

Hybrid Digital/Analog In-Memory Computing

by

Qilin Zheng

Department of Electrical and Computer Engineering  
Duke University

Defense Date: February 26, 2024

Approved:

Hai Li, Supervisor

Yiran Chen

James Morizio

Daniel Sorin

Neil Gong

Tingjun Chen

Dissertation submitted in partial fulfillment of the requirements for the degree of  
Doctor of Philosophy in the Department of Electrical and Computer Engineering  
in the Graduate School of Duke University  
2024

ABSTRACT

Hybrid Digital/Analog In-Memory Computing

by

Qilin Zheng

Department of Electrical and Computer Engineering  
Duke University

Defense Date: February 26, 2024

Approved:

Hai Li, Supervisor

Yiran Chen

James Morizio

Daniel Sorin

Neil Gong

Tingjun Chen

An abstract of a dissertation submitted in partial fulfillment of the requirements for  
the degree of Doctor of Philosophy in the Department of Electrical and Computer  
Engineering  
in the Graduate School of Duke University  
2024

Copyright © 2024 by  
Qilin Zheng

All rights reserved except the rights granted by the Creative Commons  
Attribution-Noncommercial Licence

## Abstract

The relentless advancement of deep learning applications, particularly the highly potent yet computationally intensive deep unsupervised learning models, is pushing the boundaries of what modern general-purpose CPUs and GPUs can handle in terms of computation, communication, and storage capacities. To meet these burgeoning memory and computational demands, computing systems based on In-Memory Computing (IMC), are emerging as the next frontier in computing technology. This thesis delves into my research efforts aimed at overcoming these obstacles to develop a IMC based computing system tailored for machine learning tasks, with a focus on employing a hybrid digital/analog design approach.

In the initial part of my work, I introduce a novel concept that leverages hybrid digital/analog IMC to enhance the efficiency of depth-wise convolution applications. This approach not only optimizes computational efficiency but also paves the way for more energy-efficient machine learning operations.

Following this, I expand upon the initial concept by presenting a design methodology that applies hybrid digital/analog IMC to the processing of sparse attention operators. This extension significantly improves mapping efficiency, making it a vital enhancement for the processing capabilities of deep learning models that rely heavily on attention mechanisms.

In my third piece of work, I detail the implementation strategies aimed at augmenting the power efficiency of IMC macros. By integrating hybrid digital/analog computing concepts, this implementation focuses on general-purpose neural network acceleration, showcasing a significant step forward in reducing the energy consumption of such computational processes.

Lastly, I introduce a system-level simulation tool designed for simulating general-purpose IMC based systems. This tool facilitates versatile architecture exploration, allowing for the assessment and optimization of various configurations to meet the specific needs of machine learning workloads. Through these comprehensive research efforts, this thesis contributes to the advancement of in-memory computing technologies, offering novel solutions to the challenges posed by the next generation of machine learning applications.

# Contents

Abstract . . . . .	iv
List of Tables . . . . .	ix
List of Figures . . . . .	x
Acknowledgements . . . . .	xiii
1 Introduction . . . . .	1
1.1 Main Challenge . . . . .	2
1.1.1 Flexibility . . . . .	2
1.1.2 Accuracy . . . . .	3
1.2 Thesis Contribution . . . . .	4
1.2.1 Optimization for Depth-wise Convolution . . . . .	5
1.2.2 Optimization for Sparse Attention . . . . .	5
1.2.3 Optimization for Lossless Computing . . . . .	6
2 Hybrid Digital/Analog PIM for Depth-wise CNN . . . . .	8
2.1 Backgrounds . . . . .	10
2.1.1 NVPIIM Accelerator . . . . .	10
2.1.2 Depth-wise and Point-wise Convolutions . . . . .	12
2.1.3 Depth-wise Convolution on NVPIIM . . . . .	13
2.2 MobiLattice Architecture . . . . .	14
2.2.1 Design Overview . . . . .	14
2.2.2 Mapping Strategy for Depth-wise Convolution . . . . .	16
2.2.3 Computation flow for Depth-wise Convolution . . . . .	17
2.2.4 Hybrid Digital/Analog PIM Block . . . . .	18
2.2.5 Computation Latency Analysis . . . . .	20
2.3 Evaluation Methodology . . . . .	22
2.4 Results and Discussion . . . . .	23

2.4.1	Design Overhead of NVPIM Blocks . . . . .	23
2.4.2	Performance and Memory Usage . . . . .	24
2.4.3	Impact of Weight Quantization Schemes . . . . .	25
2.4.4	Impact of ADC Resolution . . . . .	26
2.4.5	Impact of Crossbar Sizes . . . . .	28
2.4.6	Comparison to State-of-the-art Design . . . . .	29
2.5	Conclusion . . . . .	29
3	Hybrid Digital/Analog IMC for Sparse Attention . . . . .	31
3.1	Backgrounds . . . . .	34
3.1.1	Sparse Attention in Transformer . . . . .	34
3.1.2	NVPIM Basic . . . . .	35
3.1.3	Challenges . . . . .	36
3.2	Reconfigurable PIM Bank . . . . .	38
3.2.1	Design Principle . . . . .	38
3.2.2	Hybrid Digital/Analog PIM Bank . . . . .	40
3.3	Accelerator Architecture . . . . .	42
3.3.1	Architecture Overview . . . . .	42
3.3.2	Computation Flow . . . . .	44
3.3.3	Pipeline Scheme . . . . .	46
3.4	Architectural Optimization . . . . .	47
3.4.1	Greedy Scheduling Scheme . . . . .	47
3.4.2	Task Redistribution Scheme . . . . .	50
3.5	Evaluation Methodology . . . . .	52
3.5.1	Workload Setup . . . . .	52
3.5.2	Hardware Setup . . . . .	52
3.5.3	Baseline Setup . . . . .	53

3.6	Results and Discussion . . . . .	55
3.6.1	Main Results . . . . .	55
3.6.2	Ablation Study . . . . .	56
3.6.3	Scalability Analysis . . . . .	57
3.6.4	Comparison with Non-PIM Accelerators . . . . .	57
3.7	Conclusion . . . . .	58
4	Hybrid Digital/Analog IMC for lossless computing . . . . .	61
4.1	Background . . . . .	63
4.1.1	Analog IMC . . . . .	63
4.1.2	Digital IMC . . . . .	67
4.2	Design Methodology . . . . .	68
4.2.1	Partial Sum Aware Accumulation . . . . .	68
4.2.2	Macro Overview . . . . .	68
4.2.3	Nonlinear Transfer Function . . . . .	70
4.2.4	MAC in Analog Domain . . . . .	71
4.2.5	MAC in Digital Domain . . . . .	72
4.3	Experiments . . . . .	72
4.3.1	Circuit-Level Measurement Results . . . . .	72
4.3.2	Architecture-Level Simulation Results . . . . .	76
4.4	Conclusion . . . . .	76
5	Simulation Framework . . . . .	78
5.1	PIMulator-NN . . . . .	79
5.1.1	Framework Overview . . . . .	79
5.1.2	Event-Driven Simulation Core . . . . .	81
5.1.3	Architecture Description . . . . .	83
5.1.4	Example: Computational Sub-array . . . . .	85

5.1.5	Discussion: Accuracy-Related Simulation . . . . .	86
5.1.6	Limitations of PIMulator-NN . . . . .	86
5.2	Case Study . . . . .	87
5.2.1	Small-Scale DNN Accelerators . . . . .	87
5.2.2	Modeling Large-Scale DCNN Accelerator . . . . .	90
5.3	Experiment Setup . . . . .	92
5.4	Validation . . . . .	93
5.5	Key Observations . . . . .	94
5.5.1	Results for Small-Scale DNN Accelerator . . . . .	94
5.5.2	Results for Large-Scale DCNN Accelerator . . . . .	95
5.6	Conclusion . . . . .	99
6	Conclusion . . . . .	101
6.1	Summary of Contributions . . . . .	101
6.2	Future Work . . . . .	102
6.2.1	Micro architecture level . . . . .	102
6.2.2	Circuit level . . . . .	102
6.2.3	Architecture level . . . . .	103
6.3	From Academia To Industry . . . . .	103
	Bibliography . . . . .	105
	Biography . . . . .	122

## List of Tables

2.1	Neural Networks Configuration . . . . .	22
2.2	NVPIM Block Evaluation Results . . . . .	23
3.1	Area breakdown for our NVPIM bank and the conventional NVPIM bank. .	52
3.2	Hardware Configurations . . . . .	53
4.1	Comparison with State-of-the-art IMC Design . . . . .	74

## List of Figures

2.1	Illustration of NVPIM accelerator. . . . .	10
2.2	Depth-wise and point-wise convolution. . . . .	12
2.3	Data layout of depth-wise convolution on NVPIM for (a) baseline method and (b) compression method. . . . .	13
2.4	Tile Level Computation Flow for (a) Depth-Wise DCNN and (b) Standard Convolution. . . . .	15
2.5	Illustration of proposed (a) mapping strategy and (b) computation flow for depth-wise convolution. . . . .	16
2.6	Working principle for MobiLattice with (a) analog mode and (b) digital mode. Scheme of (c) ADC circuit, (d) Local PE and (e) wordline driver. . .	18
2.7	(a) Performance speedup, (b) energy saving and (c) memory space utilization for baseline settings. . . . .	24
2.8	Performance speedup of MobiLattice for different weight quantization scheme.	26
2.9	Performance speedup of MobiLattice for different ADC resolution. . . . .	27
2.10	Performance speedup comparison of MobiLattice for different crossbar sizes.	28
2.11	Performance and energy comparison to ISAAC. . . . .	29
3.1	Limited latency reduction of the previous NVPIM architecture with sparse attention models. . . . .	33
3.2	The computation process of a typical sparse attention block. . . . .	34
3.3	Illustration the inefficiency to map the (a) attention computation stage and (b) output computation stage to the conventional analog PIM bank. . . . .	37
3.4	The nested-loop expression of IP and SVM primitives on attention map computation and output computation stage. . . . .	38
3.5	Illustration of the computation path of the PIM bank for (a) VMM primitive, (b) IP primitive and (c) SVM primitive. . . . .	38
3.6	The data path of Col.MUX and MBSA in (a) analog mode and (b) digital mode. . . . .	39
3.7	The overall architecture of SparseLattice. . . . .	43
3.8	An example of data mapping and computation flow for (a) dense operation, (b) $Q \times K^T$ and (c) $\text{Attn} \times V$ . . . . .	44

3.9	Pipelined execution scheme in SparseLattice. . . . .	45
3.10	The greedy scheduling scheme, including the schematic of the attention map mask buffer and the index buffer. . . . .	48
3.11	The task redistribution scheme. . . . .	50
3.12	Speedup and the energy efficiency results on vision and NLP tasks. . . . .	54
3.13	Ablation study for latency of our designs over the baseline accelerator with different sparsity level on DeiT-Small model. . . . .	55
3.14	The latency and energy results with different number of banks for our design and ReTransformer. . . . .	57
3.15	(a) Energy-latency trade-off (b) energy breakdown for PIM and non-PIM designs with different sparsity level on DeiT-Small model. . . . .	59
4.1	Illustration of analog IMC macro, digital IMC macro and related cell design. . . . .	64
4.2	Psum value distribution of Resnet-20 on CIFAR-10 dataset. . . . .	67
4.3	The overall architecture of our proposed macro. . . . .	69
4.4	Simulated transfer function of proposed macro and conventional macro. . . . .	70
4.5	Simulated timing diagram of analog mode and digital mode computation. . . . .	71
4.6	The die photograph, layout of our proposed macro, and the area breakdown results. . . . .	73
4.7	Measured energy consumption for digital mode and analog mode. . . . .	74
4.8	Measured Shmoo plot for the proposed macro for the digital mode and analog mode. . . . .	75
4.9	Architecture simulation results for area efficiency and energy efficiency on various machine learning model. . . . .	77
5.1	Overview of PIMulator-NN framework. . . . .	80
5.2	Illustration of simulation process for PIMulator-NN . . . . .	81
5.3	Illustration of architecture description in PIMulator-NN. (a) General module description. (b) Code organization. (c) Subarray module description. . . . .	83
5.4	The illustration for MLP with (a) naive scheme, (b) intermediate data reuse scheme and (c) scheme with interconnection. . . . .	88
5.5	Illustration for CNN with (a) baseline scheme, (b) scheme with data parallelism (c) scheme with data reuse, (d) scheme with inter layer pipeline. . . . .	89

5.6	Power, performance and area results of three design schemes on MLP. . . .	94
5.7	Extracted power traces of the baseline system during processing convolution layers of VGG-13. . . . .	95
5.8	The latency and energy results for baseline design to process layer0 in VGG-13 with different degrees of parallelism and interconnect width. . . . .	96
5.9	The energy and latency results to process convolution layers with different configurations of data reuse scheme and baseline scheme. . . . .	97
5.10	The trade-off between latency and area under different data parallelism and pipeline scheme. . . . .	99

## **Acknowledgements**

Thank my advisor, Dr. Li and Dr. Chen.

# 1. Introduction

In recent years, there has been a significant increase in advanced computing tasks such as data analytics, machine learning, bioinformatics, and graphics processing. Machine learning, especially Deep Neural Networks (DNNs), has become particularly popular for their exceptional ability to surpass human performance in tasks related to video, image, speech, and natural language processing. This has led to their widespread application in real-world products and services, including speech recognition systems like Apple Siri, Google Assistant, and Amazon Alexa, image analysis tools such as Google+ image search and Facebook DeepFace, and natural language processing technologies like Google Translate and Facebook DeepText. These applications extend to search engines, recommendation systems, and more, marking a departure from traditional computing workloads due to their emphasis on simple computations across large datasets, resulting in high data transfer and memory access rates.

Traditional Von-Neumann architectures, with their separate processing and memory units, have long been the cornerstone of computing platforms. However, they struggle with performance and energy efficiency in handling these data-intensive tasks due to the significant overhead associated with moving data between the processor and memory. Addressing the bottleneck between processors and memory, integrating compute and memory components more closely has emerged as a promising solution. This integration aims to minimize frequent memory access, thus significantly boosting system performance and energy efficiency. This concept has spurred numerous academic studies and a few commercial ventures, exploring approaches such as In-Memory Computing (IMC) or Processing-In-Memory (PIM).

The adoption of commercialized and emerging memory technologies for IMC/PIM represents a groundbreaking shift towards addressing the efficiency and performance challenges posed by traditional computing architectures. These novel memory technologies, such as Spin Transfer Torque Magnetic RAM (STT-MRAM), Resistive RAM (ReRAM), and Phase Change Memory (PCM), offer unique advantages including non-volatility, high density,

and low power consumption. In contrast, the mature memory technology such as SRAM, DRAM and even storage are also widely used to implement the IMC/PIM scheme. By integrating computational capabilities directly within memory arrays, IMC/PIM leverages the inherent characteristics of these technologies to perform data-intensive computations closer to where data is stored. This architectural innovation significantly reduces the data movement between memory and processor, mitigating the bandwidth and latency bottlenecks that hamper conventional systems. As a result, IMC/PIM technologies enables more energy-efficient computing and higher performance for a wide range of applications, from machine learning algorithms to real-time data analytics, thereby heralding a new era of computing paradigms optimized for the data-centric demands of modern workloads.

## **1.1 Main Challenge**

Despite the progress made over 8-10 years of research in this domain, significant challenges remain, particularly as Deep Neural Networks (DNN) models become larger and more complex.

### **1.1.1 Flexibility**

One of the enduring challenges in the field of In-Memory Computing (IMC) and Processing-in-Memory (PIM) is the issue of programming flexibility. Initially designed IMC/PIM architectures were largely specialized, tailored to excel at particular computational tasks or to handle specific data patterns. This design philosophy, while effective for those targeted applications, significantly narrows the scope of adaptability of these architectures to the diverse array of algorithms and operations that modern Deep Neural Network (DNN) models demand. Such a specialized focus can restrict developers' abilities to tweak or fine-tune their models to run optimally on IMC/PIM platforms. This limitation not only stifles innovation but also prevents the full potential of IMC/PIM technologies from being realized across a wider range of applications.

For instance, early IMC/PIM systems might have been optimized for matrix multiplication operations, a common requirement in many DNN tasks. However, as DNN models

evolve, they increasingly incorporate complex structures such as recurrent neural networks (RNNs) for natural language processing or convolutional neural networks (CNNs) for image recognition, which require not just matrix multiplications but also a variety of other computations like convolutions, pooling, and nonlinear activations. A system optimized solely for efficient matrix multiplication might struggle with or perform suboptimally when tasked with these other types of operations. This discrepancy highlights the need for IMC/PIM architectures that are not only high-performing but also versatile and adaptable, capable of accommodating the full breadth of computational demands presented by cutting-edge DNN models without compromising on efficiency or effectiveness.

Another example is that, handling of sparse data structures presents a further example of the programming flexibility issue. Many DNN models, especially those used in natural language processing (NLP) applications, benefit from sparse data representations to manage the vast vocabulary sizes efficiently. However, an IMC/PIM system designed primarily for dense matrix operations may not efficiently support the sparse matrix operations that these models rely on. This limitation could lead developers to redesign their models around denser representations, increasing memory and computational requirements and negating some of the benefits that IMC/PIM technologies offer.

### **1.1.2 Accuracy**

Another significant challenge lies in computation accuracy. The physical operations performed by IMC/PIM architectures, particularly those involving analog components like memristors, can introduce variability and imprecision in computational results. As DNN models grow in size and complexity, the demand for high precision in computations also increases, especially for tasks requiring fine-grained accuracy such as image classification, natural language processing, and speech recognition. Ensuring that IMC/PIM systems can meet these accuracy requirements without incurring prohibitive energy or latency costs is a complex problem, requiring innovative solutions in architectural design, error correction, and compensation techniques.

Addressing these challenges demands a multi-faceted approach, combining advancements in memory technology, architectural innovations, and sophisticated software frameworks that can optimize computational accuracy while providing the flexibility needed to support a wide range of DNN models and applications. As the field of IMC/PIM continues to evolve, overcoming these obstacles will be crucial for realizing the full potential of this promising computing paradigm.

## ***1.2 Thesis Contribution***

This dissertation delves into the utilization of CMOS and emerging memory technologies to enhance In-Memory Computing (IMC) capabilities, aiming to adeptly manage contemporary Deep Neural Network (DNN) computing tasks. It particularly emphasizes on augmenting the general-purpose processing efficiency of IMC/Processing-in-Memory (PIM) systems through the innovative application of hybrid digital/analog IMC/PIM techniques. Such an approach not only broadens the scope of IMC/PIM applications but also addresses the specific challenges associated with executing complex DNN operations directly within memory arrays. The thesis meticulously explores several critical optimizations to further this goal. These include the processing of depth-wise convolution kernels, which are fundamental to efficient neural network computations, especially in mobile and edge computing environments. Additionally, it investigates the optimization of sparse attention mechanisms, crucial for the performance and scalability of transformer models that are central to many state-of-the-art natural language processing tasks. Lastly, the work targets the enhancement of high-precision Deep Convolutional Neural Network (DCNN) processing, aiming to improve the accuracy and reliability of DNN outputs. These focused optimizations are designed to leverage the unique advantages of IMC/PIM, such as reduced data movement and increased computational efficiency, thereby pushing the boundaries of what is possible with current DNN architectures and computational paradigms.

### 1.2.1 Optimization for Depth-wise Convolution

Prior works have demonstrated that using depth-wise DCNNs can achieve 90% computation complexity and model storage reduction than conventional DCNNs. Although the NVPIM architecture and the DW convolution can improve inference efficiency of DCNNs, the overall efficiency degrades seriously when mapping a depth-wise DCNN to NVPIM architecture. Existing NVPIM architectures are mainly designed for standard convolution, where the PW convolution layer in Depth-wise DCNN can be efficiently processed. Here the PW convolution layer can be regarded as a standard convolution with  $1 \times 1$  kernels. However, the DW convolution can not be executed efficiently due to the extreme low utilization of the crossbar when mapping a DW convolution layer to a NVPIM architecture as a VMM data layout. One major problem is that the feature map is only used for computation in its own channel, leading to large number of unused memory cells.

In order to enable efficient inference for both DW and PW convolution layers, we present a novel hybrid digital/analog mode NVPIM architecture. The basic idea is to introduce a digital mode to process DW convolution layers, while keeping equals mode for computation of PW convolution layers. This is the first work to address the issue of processing DW on the NVPIM architecture; ce and computation latency. This work to enable both analog and digital modes effectively on the same crossbar with a moderate overhead (less than 1%), in terms of area and power consumption.

### 1.2.2 Optimization for Sparse Attention

Attention-based neural networks have shown superior performance in a wide range of tasks. However, the computational complexity of the attention operation hinders the deployment of attention-based neural networks on resource-constrained devices. From the algorithm perspective, a better accuracy-computation complexity trade-off can be achieved by exploiting a dynamic and unstructured sparsity in the attention map matrix, where the original attention computation is converted to a sampled dense-dense matrix multiplication (SDDMM) and a sparse-dense matrix multiplication (SpMM). From the hardware per-

spective, non-volatile processing-in-memory (NVPIM)-based architecture shows its great potential to accelerate the dense model. However, the unique unstructured and dynamic sparsity pattern in the sparse attention challenges the mapping efficiency of the NVPIM architecture, as the conventional NVPIM architecture uses a vector-matrix-multiplication primitives.

To fill the gap between the NVPIM architecture and the sparse processing, in this paper, we propose SparseLattice, a NVPIM architecture to accelerate a dynamic and unstructured sparse computation in the sparse attention. We aim to improve the mapping efficiency for both SDDMM and SpMM by introducing two vector-based primitives with a reconfigurable NVPIM bank. Further, based on our reconfigurable NVPIM bank, we propose a hybrid stationary data flow, a greedy scheduling scheme and a task redistribution scheme to further improve the mapping efficiency.

### **1.2.3 Optimization for Lossless Computing**

To integrate analog IMC into general purpose NN inference accelerators, one key design metric is to support lossless MAC operations. To achieve this specific requirement, the analog accumulation process should be robust to PVT variations and enough margin should be provided for successive sensing, and an ADC with enough effective number of bits (ENOB) is required to accurately generate the accumulation results. Prior efforts to improve the robustness include investigating the usage of advanced analog computing schemes such as capacitive coupling [33]. However, to implement the capacitive coupling scheme, one or more capacitors are required to be added to each memory cell, and multiple transistors are required to implement the switches. In addition, the advanced analog computing scheme only resolves the robustness issue, while the ENOB requirement of the ADC can not be eliminated. There will still be considerable energy cost of the interface circuits to convert the analog value into digital domain.

In this chapter, we propose an alternative solution to the lossless SRAM-based analog IMC macro, instead of focusing on pure circuit level innovation to build up robust com-

puting schemes. The basic idea is to leverage the sparsity in typical NN workloads, where the computation results will mainly be small values. Simulation results show that, when 64 rows are activated in parallel, 90% accumulation results are within 8 instead of following a uniform distribution. Different from the conventional analog IMC design methodology where all accumulation results in the analog domain are treated equally, we propose an analog computing scheme based on a nonlinear transfer function which only covers an accurate computation for the low MAC value region. Significant energy efficiency improvement can be observed as the ENOB requirement for the interface ADC can be reduced.

## 2. Hybrid Digital/Analog PIM for Depth-wise CNN

Deep Convolutional Neural Networks (DCNNs) have been successfully applied to various fields, such as image classification [129]. The success of DCNNs, however, comes at the cost of high computation intensity and the use of large dataset [60]. Thus, it is generally challenging to deploy DCNNs on the resource constraint systems, such as mobile devices, edge computing nodes, smart embedded devices, etc. To overcome this challenge, researchers have been seeking efficient solutions by leveraging advancements from both hardware and DCNN algorithms.

On the one hand, researchers have developed various hardware accelerators dedicated for DCNNs. Among these approaches, Nonvolatile Processing-In-Memory (NVPIM) architecture has been considered as an attractive solution to provide highly efficient computing capability and reduce the overhead of memory access simultaneously [45]. First, the NVPIM can perform the Vector-Matrix-Multiplication (VMM) operations efficiently via analog computing on the crossbar. Second, it can substantially reduce the overhead of data movement, as the computation happens in-situ inside the memory array. Recent works have proved that using NVPIM can significantly improve energy efficiency of DCNN inference [134, 139].

On the other hand, researchers have also proposed different lightweight DCNN models to reduce the computation intensity and the number of model parameters (weights). Depth-wise DCNN is a representative lightweight design [63, 132, 41]. Its basic idea is to replace a conventional convolution layer with a combination of the *Depth-Wise (DW) convolution layer* and the *Point-Wise (PW) convolution layer*, which induce much fewer multiplication and accumulation operations. Prior works have demonstrated that using depth-wise DCNNs can achieve 90% computation complexity and model storage reduction than conventional DCNNs [63].

Although the NVPIM architecture and the DW convolution can improve inference efficiency of DCNNs, the overall efficiency degrades seriously. when mapping a depth-wise DCNN to NVPIM architecture. Existing NVPIM architectures are mainly designed for standard convolution, where the PW convolution layer in Depth-wise DCNN can be effi-

ciently processed. Here the PW convolution layer can be regarded as a standard convolution with  $1 \times 1$  kernels. However, the DW convolution can not be executed efficiently due to the extreme low utilization of the crossbar when mapping a DW convolution layer to a NVPIM architecture as a VMM data layout. One major problem is that the feature map is only used for computation in its own channel, leading to large number of unused memory cells (more details are further provided in subsection 2.1.3). In an extreme case, DW convolution has the same computation latency and memory space as a standard convolution under the same kernel size.

A straightforward solution is to apply the compression techniques on DW convolution layers to improve the memory utilization. SRE [176] has proposed an Operation Unit (OU) row compression approach to process sparse DCNN model. The utilization improvement depends on the sizes of the crossbar and the OU. However, it induces hardware overhead to support sparse processing. In addition, the efficiency of ADCs may be decreased. In some special cases, the processing latency may be longer as the compression approach deforms the data alignment in conventional approaches.

In order to enable efficient inference for both DW and PW convolution layers, we propose a novel hybrid digital/analog mode NVPIM architecture. The basic idea is to introduce a digital mode to process DW convolution layers, while keeping equals mode for computation of PW convolution layers. The key contributions of this work are listed as follows:

- To the best of our knowledge, this is the first work to address the issue of processing DW on the NVPIM architecture;
- We propose to enable both analog and digital modes effectively on the same crossbar with a moderate overhead (less than 1%), in terms of area and power consumption;
- We present comprehensive experimental results to demonstrate that our hybrid mode NVPIM architecture can achieve up to  $30\times$  speedup over traditional NVPIM approaches.

## 2.1 Backgrounds

### 2.1.1 NVPIM Accelerator

Figure 2.1 illustrates the architecture of an embedded SoC with a NVPIM accelerator. The NVPIM accelerator is composed of a group of tiles connected by a local bus. Each tile consists of a set of PIM Blocks to perform vector-matrix multiplications, an in-tile buffer to store intermediate data, and a Functional Unit (FU) to process computation other than vector-matrix multiplications. Each PIM block has one or multiple memory crossbars, input/output registers to buffer input/output data, and peripheral circuit. An accumulation unit is usually included in each PIM block to accumulate partial sum results.

Without loss of generality, we choose ReRAM as an example to demonstrate the design of NVPIM architecture [134]. Note that the same design can be extended to other NVM

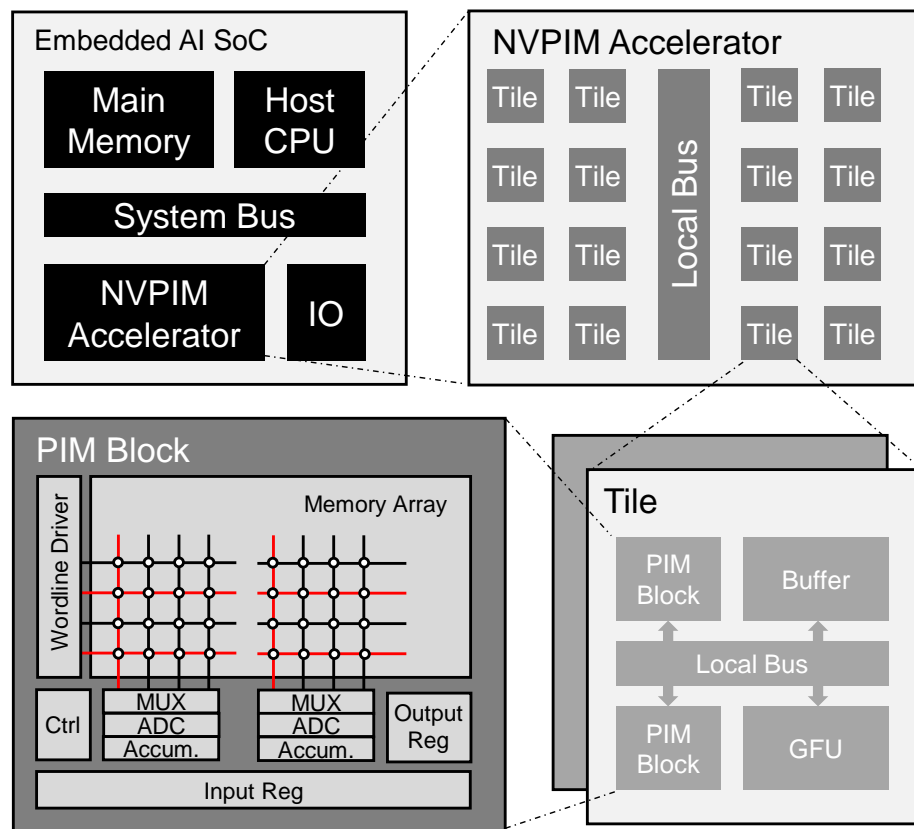


FIGURE 2.1: Illustration of NVPIM accelerator.

technologies, such as STT-MRAM [82], FeFET [30], etc. Logic values are stored as the resistance of the memory cells, which can be switched between two or more levels by applying electrical excitation with different amplitudes and duration. The NVM cells are arranged with a structure of *crossbar*, which has been widely used to perform VMM [65].

The elements of a matrix can be represented as the conductance of the NVM cells located at every cross-point of a crossbar. The vector can be represented by the voltages applied as the crossbar’s inputs. The voltages are generated by wordline driver and they could either be binary or multi-level. Then the bitline currents are collected at the outputs of the crossbar and represent the VMM result. In this design, ADC is needed to convert the signal of the analog bitline current to digital values.

Theoretically, the wordline driver could activate all wordlines of a memory crossbar. And the number of ADCs equals the column number of the memory crossbar. Thus, it only takes  $O(1)$  time complexity to perform the VMM computation. In practice, there are several limitations to reach such high computing efficiency: Firstly, simultaneously turning on many wordlines makes it hard to perform computation accurately. High bitline current would induce significant IR-drop. In addition, cell resistance variations would induce accumulated error. Thus, only a portion of wordlines could be activated at the same time. Secondly, the ADC consumes much more power and area than a memory crossbar. Consequently, one ADC needs to be shared by multiple bitlines and only a limited number of bitline currents can be sampled in one ADC cycle<sup>1</sup>.

Assuming  $2^{R_{ADC}} - 1$  wordlines can be activated simultaneously and the number of ADC is  $N_{ADC}$ . Then, the processing latency of one crossbar with row number  $N_{xb,row}$  and column number  $N_{xb,col}$  can be calculated as:

$$T_{xb} = \frac{N_{xb,row} \times N_{xb,col}}{(2^{R_{ADC}} - 1) \times N_{ADC}}. \quad (2.1)$$

---

<sup>1</sup> The ADC cycle is defined as the latency to sample an analog signal.

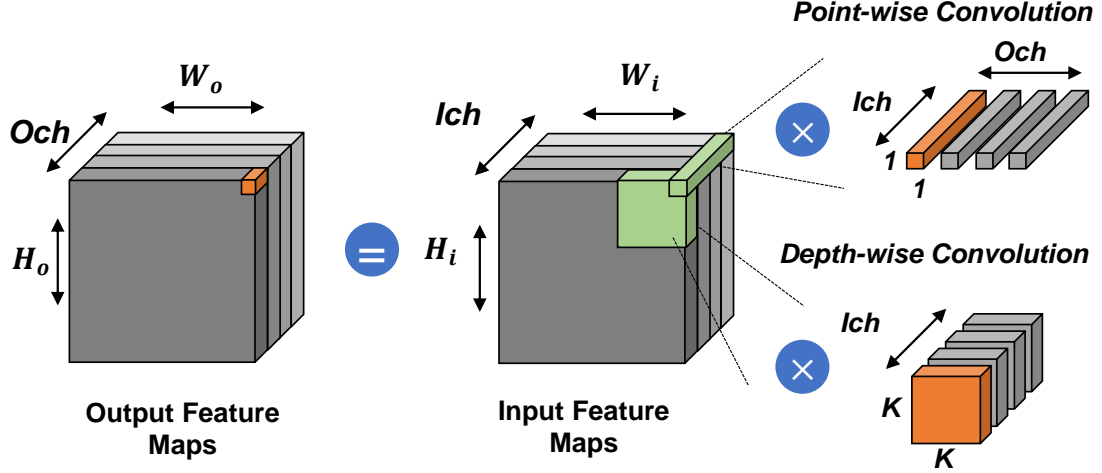


FIGURE 2.2: Depth-wise and point-wise convolution.

### 2.1.2 Depth-wise and Point-wise Convolutions

In typical depth-wise DCNNs [41, 63, 132], a standard convolution is factorized into a depth-wise convolution and a  $1 \times 1$  convolution, which is also called point-wise convolution. Depth-wise convolution applies a filter to each input channel for spatial accumulation. Point-wise convolution applies a  $1 \times 1$  standard convolution to perform inter-channel feature maps accumulation. Figure 2.2 illustrates the computation flow of depth-wise and point-wise convolution in a state-of-the-art light weighted model.

In this example, an NN layer takes a tensor  $T^{in}$  in the dimension of  $[I_{ch} \times H_i \times W_i]$  as input and produces a tensor  $T^{out}$  in the dimension of  $[O_{ch} \times H_o \times W_o]$ . Here,  $H_i/H_o$ ,  $W_i/W_o$ ,  $I_{ch}$ ,  $O_{ch}$  represent the spatial height, width, input, and output depth, respectively. In depth-wise convolution, the kernel tensor  $T^k$  is in the dimension of  $[Ch \times K \times K]$ , where  $K$  represents the kernel size. The convolution of depth-wise convolution can be written as:

$$T_{ch}^{out}(h, w) = \sum_{s=0}^{K-1} \sum_{t=0}^{K-1} T_{ch}^{in}(h+s, w+t) \times T_{ch}^k(s, t). \quad (2.2)$$

### 2.1.3 Depth-wise Convolution on NVPIIM

Figure 2.3 illustrates mapping of a depth-wise convolution on the NVPIIM architecture. For simplicity, we assume the weight is 4-bit in this example. And each ReRAM cell can store a binary value (i.e., cell level equals 2). Then, each channel of the kernel is mapped to four columns of the crossbar. Each channel is unrolled to 1-D vector along the spatial dimension and then mapped to different rows of memory array. The crossbar utilization is quite low for two reasons. Firstly, the row-axis utilization depends on the kernel’s spatial dimension, which is usually small ( $3 \times 3$  in this example). Secondly, the column-axis utilization depends on the reuse rate of channel-wise feature maps. The reuse rate of depth-wise convolution is only one, which is much lower than that of standard convolution. In the case shown in Figure 2.3(a), mapping a depth-wise convolution with  $[Ch, K, K]$  kernel configuration requires at least  $Ch \times K \times K$  rows and  $Ch$  columns of memory arrays. This mapping method would cause significant redundant memory space and unnecessary computation.

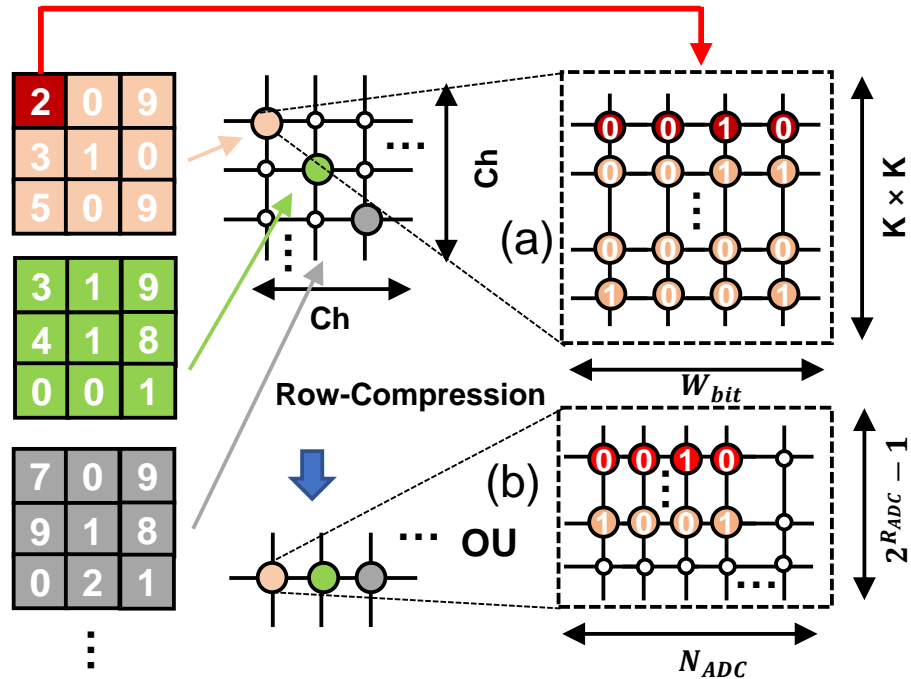


FIGURE 2.3: Data layout of depth-wise convolution on NVPIIM for (a) baseline method and (b) compression method.

Since depth-wise convolution can be viewed as a sparse convolution, a simple method to mitigate the redundant memory space is to apply weight compression techniques on NVPIM, SRE [176] has proposed an Operation Unit (OU) row compression technique to process sparse DCNN model. OU is a nominal computation unit with a size of  $N \times M$ , where  $N$  is the maximum number of the wordlines activated in parallel and the  $M$  is the number of ADCs. This OU size ensures that the computation of OU can be done in one ADC cycle. When applying OU row-compression on a sparse matrix, the original matrix is divided into many sub-matrices that are mapped into different OUs. If the elements in one row of an OU are all zero, this row can be eliminated to reduce both memory space and processing latency.

For the case shown in Figure 2.3(b), only one channel of the kernel with spatial dimension of  $3 \times 3$  and 4-bit weight can be mapped into one OU. And  $Ch$  OUs are required to map the depth-wise convolution layer. This method also has some redundant memory space if the kernel dimension can not be divided exactly by OU row size. In addition, this method would induce low ADC utilization rate. Since only one channel can be mapped in one OU, the ADCs cannot be fully utilized for computation. Thus, the maximum ADC that participate in computation equals the precision of the weights.

## **2.2 MobiLattice Architecture**

### **2.2.1 Design Overview**

Figure 2.4 depicts the tile level of the proposed MobiLattice architecture. In each tile, MobiLattice is composed of multiple PIM blocks, a functional unit (FU), and buffers. All the components are connected by a local bus. PIM blocks are the basic processing elements, operating in either digital mode or analog mode. When a PIM block is configured into analog mode, it is mapped for standard convolution or point-wise convolution by performing VMM computation. When a PIM block is configured into digital mode, it is mapped for depth-wise convolution by performing multiplication and accumulation in memory peripheral circuits. The functional unit performs miscellaneous functions such as

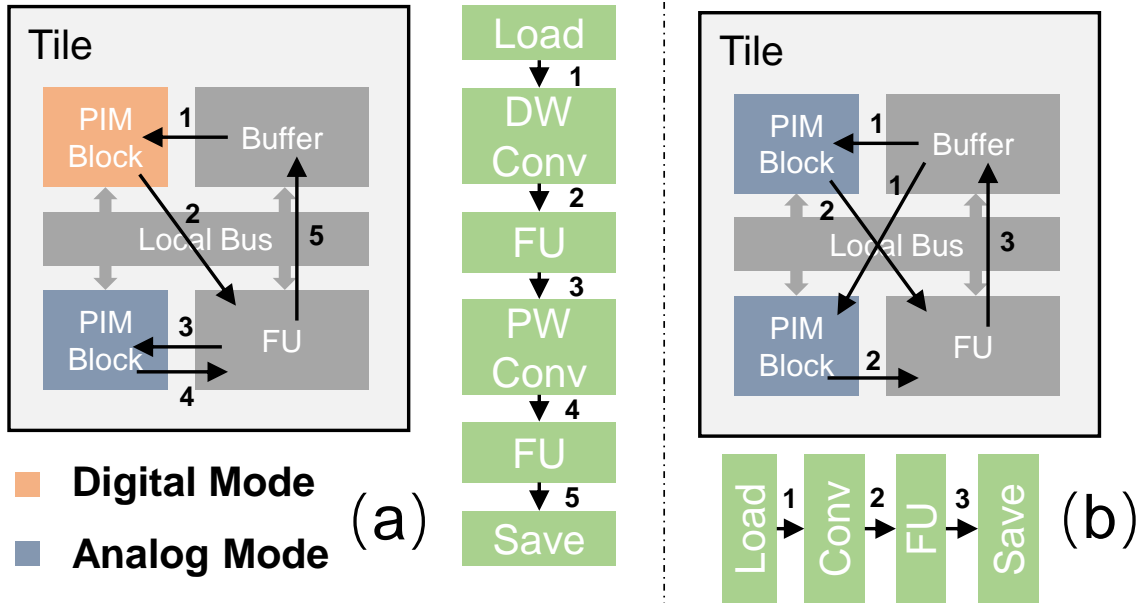


FIGURE 2.4: Tile Level Computation Flow for (a) Depth-Wise DCNN and (b) Standard Convolution.

pooling, activation, feature maps addition, etc.

For inference task, each tile processes one or several layer(s). The weights of the kernel are mapped into memory crossbars of the PIM block and the feature maps are stored in in-tile buffer. Figure 2.4(a) shows a high level computation flow of a representative convolution block in MobileNet [63]. The convolution block consists of a depth-wise layer and a point-wise layer. The feature maps data are first loaded into a NVPIM block configured to digital mode before the depth-wise convolution is performed (1). After the computation, the output data are fetched into FU for activation or max pooling operation (2). Then, the new generated intermediate feature maps are sent to a NVPIM block configured to analog mode for point-wise convolution (3). The output data are then fetched back to FU to perform activation or max pooling in the point-wise layer (4). As last, the output feature maps are saved to the buffer in this tile or in the next tile (5).

The computation flow of standard convolution is similar to lightweight convolution, as shown in Figure 2.4(b). For standard convolution, the feature maps data are loaded into a NVPIM block configured to analog mode (1) Then, the output data are fetched to FU for

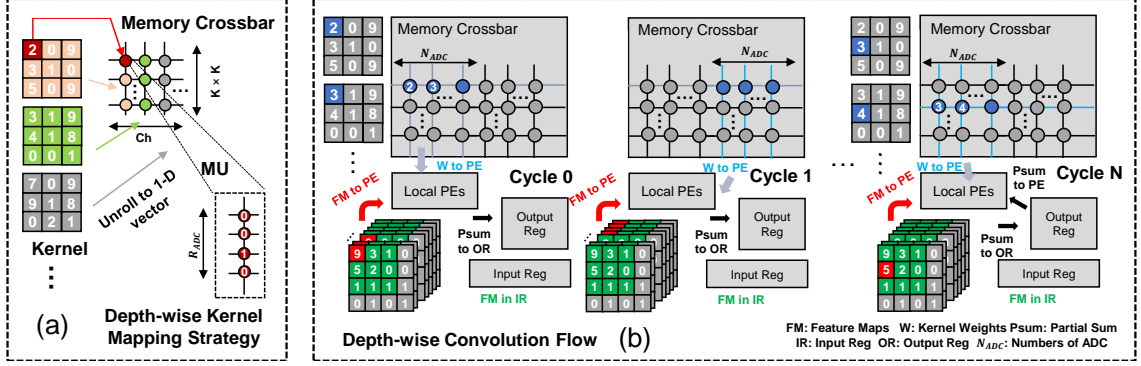


FIGURE 2.5: Illustration of proposed (a) mapping strategy and (b) computation flow for depth-wise convolution.

activation or max pooling operation (2). At last, the output feature maps are directly sent to the buffer in this tile or in the next tile (3).

## 2.2.2 Mapping Strategy for Depth-wise Convolution

We first use a simple example to explain our proposed mapping strategy. In this example, the weights of the whole depth-wise convolution layer can be mapped into one memory crossbar. The configuration of kernels can be represented by  $[Ch, K, K]$ , where  $Ch$  is the number of channels, and  $K$  is the kernel size. We still assume each memory cell can only reliably store a binary value, i.e., ‘0’ or ‘1’, to simplify the discussion. The resolution of ADC ( $R_{ADC}$ ) is set to 4.

We group  $R_{ADC}$  adjacent cells located in the same column as a Memory Unit (MU), as shown in Figure 2.5 (a). The logic value of one MU can be sampled accurately by interface circuit in an ADC cycle. Let us first discuss the case that the weight precision equals  $R_{ADC}$ . In this case, one element of the kernel can be mapped to one MU exactly. The weights in each kernel are mapped to  $K \times K$  MUs located in the same column of memory crossbar as a vector.

Each kernel is mapped into a single column of the memory crossbar. Thus,  $Ch$  columns are required in this case, as illustrated in Figure 2.5 (a).

If the precision of the kernel weights is greater than  $R_{ADC}$  (for example, 8-bit), then

one element is divided into 4-bit LSB and 4-bit MSB and mapped into two MUs located in adjunct columns. If the precision is less than  $R_{ADC}$  (for example, 2-bit)<sup>2</sup>, two elements in the same locations of two kernels are grouped and then mapped to one MU in memory crossbar. Thus, this MU based kernel mapping strategy achieves a higher memory utilization than conventional approach. In general, the mapping strategy for one layer can be written in the following equation.

$$Mem(n + R_{ADC} \times (i + K \times j), ch + n/R_{ADC}) = T_{ch}^W(i, j, n). \quad (2.3)$$

Here,  $n$  is the bit index of the weight.

The above strategy can be easily extended to the case when a depth-wise convolution layer cannot fit into one memory crossbar. The layer requires  $k \times k \times Ch$  MUs to map all its weights. They can be mapped into multiple crossbars in practice.

### 2.2.3 Computation flow for Depth-wise Convolution

When processing a depth-wise convolution, the PIM blocks works in digital mode. Weights are fetched from the crossbar and computed in the memory peripheral logics, namely, Local PEs. A basic flow is illustrated in Figure 2.5 (b).

We still use the simple example in the last subsection to explain the detailed flow. The weights have been mapped to the crossbar. The input feature maps are stored in the Input Reg. We define the number of ADCs in each PIM block as  $N_{ADC}$ . Thus, we can read out  $N_{ADC}$  MUs in each ADC cycle. These weights are sent to Local PEs. The corresponding feature maps are fetched to Local PEs at the same time to multiply with weights. Then the output results are sent to the Output Reg.

As shown in Cycle 0 and 1 of Figure 2.5 (b), the weights are read out in the row-major style. In Cycle N, the second row of MUs are read out and sent to the Local PEs for multiplication. The partial sums in the Output Reg are also fetched to the Local PEs and added with the multiplication results. Then, the accumulated results are written back to the Output Reg.

---

<sup>2</sup> We only consider that the weight precision is a power of 2

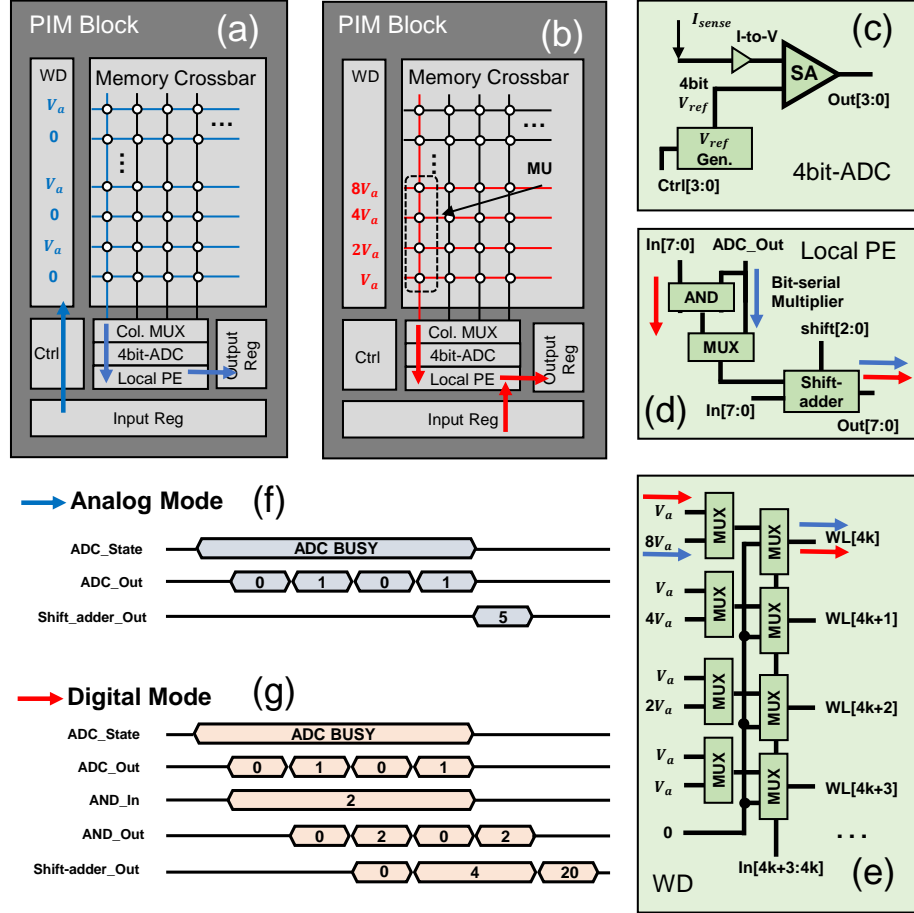


FIGURE 2.6: Working principle for MobiLattice with (a) analog mode and (b) digital mode. Scheme of (c) ADC circuit, (d) Local PE and (e) wordline driver.

## 2.2.4 Hybrid Digital/Analog PIM Block

We use the same setup in subsection 2.2.3 to explain our proposed hybrid digital/analog NVPIM block design. Figure 2.6 depicts the structure of our proposed hybrid NVPIM block, which consists of several memory crossbars, a Wordline Driver (WD), Input Register (Input Reg) for buffering input data, Output Register (Output Reg) for buffering intermediate partial sum, and related Control circuits (Ctrl). Multiple columns share one ADC through a Column Multiplexer (Col.MUX). The ADC samples the current on each column in a time-division-multiplexing manner. Here, we use a SAR-ADC [131] to balance the power and throughput. In SAR ADC, one bit can be sampled in one clock cycle. Thus, an ADC

cycle is composed of  $R_{ADC}$  cycles when the resolution of ADC is  $R_{ADC}$ . Each ADC has its own Local PE. The Local PE can be configured into either digital or analog mode. When Local PE is configured into analog mode, it performs shift-add operation to accumulate partial sum. When local PE is configured into digital mode, it performs multiplication and accumulation.

**Analog Mode:** The data path of analog mode is illustrated in Figure 2.6 (a). A bit-vector is fetched from Input Reg to WD to generate a binary voltage vector ( $V_a$  for logic value “1”) applied on each wordline. The bitline accumulated current represents the partial sum and is sampled by an ADC. The partial sum value is directly sent to Output Reg when the input bit precision is 1. Otherwise, this partial sum would be sent to the shift-adder in local PE for partial sum accumulation. Figure 2.6 (f) illustrates an example of the computation in analog mode. Generally, a 1-bit/1-bit vector dot-product with a vector length of 15 can be performed in one 4-bit ADC cycle.

**Digital Mode:** The data path of digital mode is illustrated in Figure 2.6 (b). When perform the computation, the input data are directly fetched into local PE. Different from analog mode, WD generates a multi-level voltage vector ( $1.0-8.0V_a$ ) to activate one MU. The circuit illustration of WD is shown in Figure 2.6 (e). It should be noted that this multi-level voltage reference generator is not a complex circuit module, which is also required in the ADC, as shown in Figure 2.6 (c), or memory programming circuit. Then the bitline accumulated current represents a 4-bit logic value. The weights are read out by the ADC and sent to Local PE to perform multiplication with input feature maps.

One concern is that a bit-parallel multiplier may consume too much area and power. Here, we use a bit-serial multiplier by reusing the shift-adder and adding an “AND” logic, as shown in Figure 2.6 (d). Since ADC conversion time is usually longer than the latency of multiplication using CMOS circuit, a bit-serial multiplier would not limit the computation performance. The input precision of the Local PE is 8-bit in this example, which is a common case in embedded systems. Thus eight “AND” gates and 3-bit shift value are enough for bit-serial multiplier. Since the shift-adder is also required in analog mode for

multi-bit feature maps/weight computation, the extra overhead in Local PE to support digital mode is only caused by the “AND” gates and “MUX” circuit. Figure 2.6 (g) shows a computation example of local PE in digital mode to calculate a multiplication. The first stage is ADC sampling stage. When the ADC starts, a 4-bit logic value is sampled in a serialized manner. The second stage is generating “AND” results. After an 1-bit value is sampled, the sampled bit performs “AND” with 8-bit input value to generate partial sum. The third stage is using the shift-adder to accumulate the partial sum. Generally, in one 4-bit ADC cycle, an 8-bit input can be multiplied with one 4-bit weight, or two 2-bit weights.

It is worth noting that our digital mode also works for Multi-Level Cells (MLC). For example, if a 2-bit data is stored in each cell, and the resolution of the ADC is unchanged, the number of wordlines activated in parallel can be reduced to two to read out 4-bit logic value. The other design metrics in this case remain the same as the original binary cell designs.

## 2.2.5 Computation Latency Analysis

Since the operational latency of Local PEs is usually shorter than that of the ADC, the major time consumed in the NVPIM is the latency of the crossbar and the ADC [134]. In the baseline design, the computation latency can be written as follows:

$$T = N_{xb} \times T_{xb}. \quad (2.4)$$

Here,  $N_{xb}$  is the mapped crossbar numbers and  $T_{xb}$  is the total computation latency of one crossbar and its related ADC. For a depth-wise convolution layer with a kernel configuration of  $[Ch, K, K]$ ,  $N_{xb}$  can be calculated by:

$$N_{xb} = \text{ceil}\left(\frac{Ch}{N_{Ch,xb}}\right). \quad (2.5)$$

$N_{Ch,xb}$  represents the number of kernels which can be mapped on one crossbar, and it can be expressed as:

$$N_{Ch,xb} = \min\left(\frac{N_{xb,row}}{K^2}, \frac{N_{xb,col}}{W_{bit}}\right). \quad (2.6)$$

$N_{xb,row}$ ,  $N_{xb,col}$  and  $W_{bit}$  represent the row size and the column size of a crossbar and the precision of weight, respectively. For most cases,  $\frac{N_{xb,row}}{K^2}$  is a small value. Thus, Equation 2.4 can be approximately rewritten as<sup>3</sup>:

$$\begin{aligned} T &\approx \frac{Ch \times K^2}{N_{xb,row}} \times \frac{N_{xb,row} \times N_{xb,col}}{(2^{R_{ADC}} - 1) \times N_{ADC}} \times IF_{bit} \\ &= \frac{Ch \times K^2 \times N_{xb,col}}{(2^{R_{ADC}} - 1) \times N_{ADC}} \times IF_{bit}. \end{aligned} \quad (2.7)$$

Equation 2.7 shows that the computation latency  $T$  is roughly proportional to  $N_{xb,col}$  and  $IF_{bit}$ .

For OU based compression techniques, the calculation of  $T$  is similar except  $N_{xb,col}$  needs to be substituted by OU column size, which equals  $N_{ADC}$ . We have:

$$\begin{aligned} T &\approx \frac{Ch \times K^2 \times N_{ADC}}{(2^{R_{ADC}} - 1) \times N_{ADC}} \times IF_{bit} \\ &= \frac{Ch \times K^2}{2^{R_{ADC}} - 1} \times IF_{bit}. \end{aligned} \quad (2.8)$$

Our proposed digital mode design requires  $K^2 \times R_{ADC}$  rows and  $\frac{Ch \times W_{bit}}{R_{ADC}}$  columns to store the weights, thus,  $T$  can be calculated by:

$$T = \frac{Ch \times K^2 \times W_{bit}}{R_{ADC} \times N_{ADC}}. \quad (2.9)$$

Note that in our proposed digital mode, the latency is independent to  $IF_{bit}$ , as aforementioned in subsection 2.2.4.

---

<sup>3</sup> We ignore the ceil function for simplicity.

## 2.3 Evaluation Methodology

We use ReRAM technology as an example to implement the NVPIM. The low resistance state (LRS) and the high resistance state (HRS) of the ReRAM cells are set to  $10\text{K}\Omega$  and  $100\text{K}\Omega$ , respectively. Only binary states of the ReRAM cells, i.e., ‘0’ (HRS) and ‘1’ (LRS), are used in our evaluation. We configure the memory crossbar in one PIM block with 256 rows and 256 columns. For reliability consideration, we follow the assumption in [176] that the maximum number of activated rows in parallel is 15, thus a 4-bit ADC is enough to serve as interface circuit. We extract parameters of an 8-bit ADC from ISAAC [134]. For other ADC resolutions, we scale the parameters according to this work [131]. 32 ADCs are shared in one memory crossbar. The buffer parameters are extracted from ISAAC [134]. For the analysis of overhead induced by digital mode, the digital circuits are synthesized and evaluated with Synopsys *Design Compiler* under 40nm process. The obtained power are scaled down to 32nm process to match the buffer results.

For DCNN model configuration, we evaluate the MobiLattice architecture using typical layers in a state-of-the-art depth-wise DCNN model, which is summarized in Table 2.1. We test three cases: for layer A-D, we evaluate a standalone depth-wise convolution layer. For layer E-G, we evaluate a depth-wise and point-wise block in MobileNet [63] or Xception [41]. Layer H, I are typical blocks chosen in MobileNetV2 [132], which are composed of a PW + DW + PW layer structure. The kernel size is  $3 \times 3$ , following the configuration in

Table 2.1: Neural Networks Configuration

Layer ID	Type	Ich	Och
A	DW	64	64
B	DW	128	128
C	DW	256	256
D	DW	512	512
E	DW + PW	64,64	64,128
F	DW + PW	128,128	128,128
G	DW + PW	256,256	256,512
H	PW + DW + PW	16,16,16	16,16,16
I	PW + DW + PW	32,192,192	192,192,32

Table 2.2: NVPIM Block Evaluation Results

Component	Params	Spec	Power
XB Array	number	8	9.6mW
	size	$256 \times 256$	
	bit per cell	1	
Local PE	number	$8 \times 32$	3.84mW
Input Reg	size	2KB	1.24mW
Output Reg	size	256B	0.23mW
Col. Mux	number	$8 \times 32$	0.14mW
Wordline Driver	number	$8 \times 256$	8mW
ADC	resolution	4-bits	7.6mW
	frequency	10MHz	
	number	$8 \times 32$	
<b>NVPIM Block</b>	–	–	30.65mW

MobileNet and MobileNetV2. We also use the whole model of Xception, MobileNet and MobileNetV2 to evaluate the system level performance. These NN layers capture the main features of depth-wise convolution with a broad range of model configuration setting. The data precision of both feature maps and weights are set to 8-bit for baseline evaluation setting.

In the baseline design, we use the NVPIM block structure which does not support digital mode mapping for depth-wise convolution layer. The point-wise convolution of both baseline and our proposed design is mapped into analog mode NVPIM block. In addition, we compare the performance and energy efficiency of the proposed NVPIM block and SRE [176]. To better explore the design space, we also evaluate the efficiency of the proposed digital mode for NVPIM block with various hardware configurations, i.e., different ADC resolutions and memory crossbar sizes.

## 2.4 Results and Discussion

### 2.4.1 Design Overhead of NVPIM Blocks

Table 2.2 shows the design metrics of proposed NVPIM block. Comparing to the original design that only supports analog mode, the Local PE is redesigned to support multiplication. The Local PE contributes to 10% power consumption. Note that the shift-adder

contributes to 83% power of the Local PE. Since crossbar and ADC consume most of the power, the cost to make the NVPIM block to support digital mode computation is negligible. As we have discussed in subsection 2.2.4, the bit-serial multiplier only requires several “AND” gates and shift-adder.

### 2.4.2 Performance and Memory Usage

The performance results of MobiLattice and the baseline design are compared in Figure 2.7 (a). With depth-wise convolution in digital mode, the performance speedup ranges from  $64 \sim 85\times$  for standalone depth-wise convolution layers A-D. Compared to the compression approach [176], the proposed digital mode NVPIM block can achieve about  $13\times$  acceleration. Considering typical convolution blocks in MobileNet [63], MobiLattice can achieve  $1.6 \sim 3.86\times$  speedup to process depth-wise and point-wise convolutions for layer E-G. The speedup effect is lower than standalone depth-wise convolution since we map the

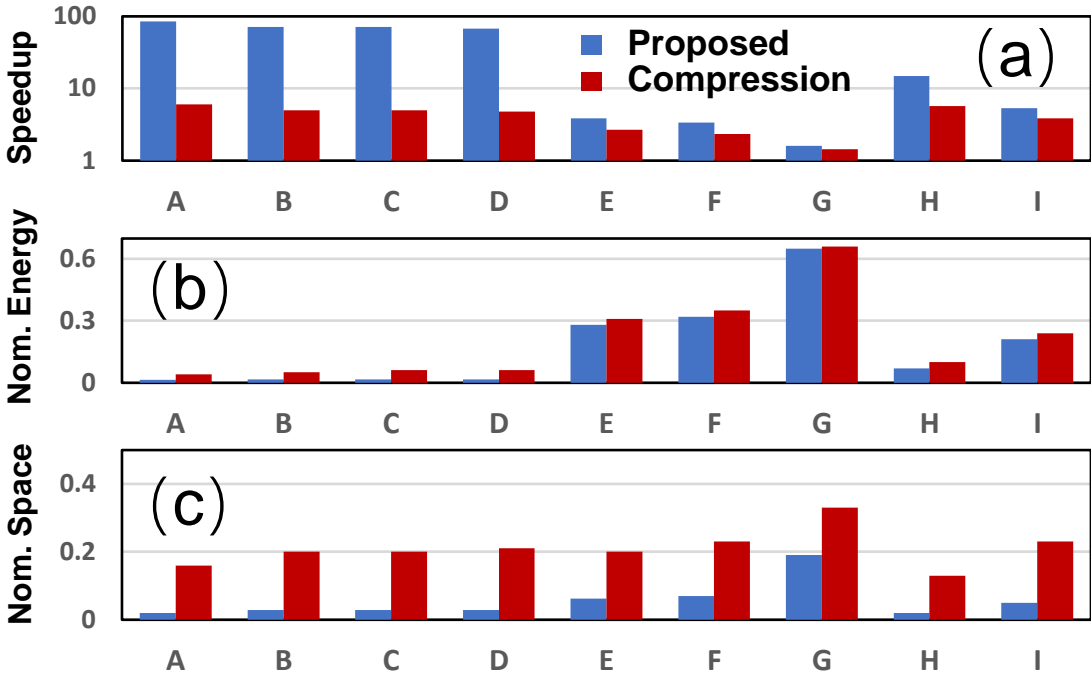


FIGURE 2.7: (a) Performance speedup, (b) energy saving and (c) memory space utilization for baseline settings.

point-wise convolution into the analog mode PIM block as baseline design. For typical convolution blocks (i.e., layer H and I) in MobileNetV2 [132], the proposed digital mode can achieve  $5.29 \sim 14.9\times$  speedup compared to the baseline design.

Both the proposed digital mode design and the compression approach can significantly reduce the processing energy. For standalone depth-wise convolution, both methods only consume 2% energy over baseline design. Considering point-wise convolution, the total energy saving rate varies between 40% – 90%.

Our proposed design can significantly reduce the memory space compared to the baseline, as shown in Figure 2.7(c): The memory space decreases by 80% on average for compression approach and over 90% by our proposed digital mode design. It is because the data layout in our proposed method is more compact and no extra memory space is required to align the kernel.

### 2.4.3 Impact of Weight Quantization Schemes

State-of-the-art DCNN models utilize quantization technique to balance the inference accuracy and model size. The weight precision varies between 1-bit and 8-bit in lightweight models. In the following subsections, we only evaluate Layers E-I from some real DCNN models.

As shown in Figure 2.8, our proposed digital mode can achieve higher performance speedup compared to both the baseline design and the compression approach. The performance speedup of compression approach keeps almost unchanged when the weight precision decreases from 8-bit to 1-bit. The maximum speedup of the proposed design is  $60.7\times$  for layer H with 1-bit weight, and the average speedup ranges between  $1.2 \sim 19.8\times$  for layers E, F, and G and between  $10 \sim 26\times$  for layers H and I, respectively. Both baseline design and compression approach can only compute depth-wise convolution in analog mode NVPIM, and the lower weight precision would induce more redundant space and lower ADC utilization rate. In contrast, our design utilizes a very compact data layout. Therefore, our design is more friendly to the quantized depth-wise convolution layer.

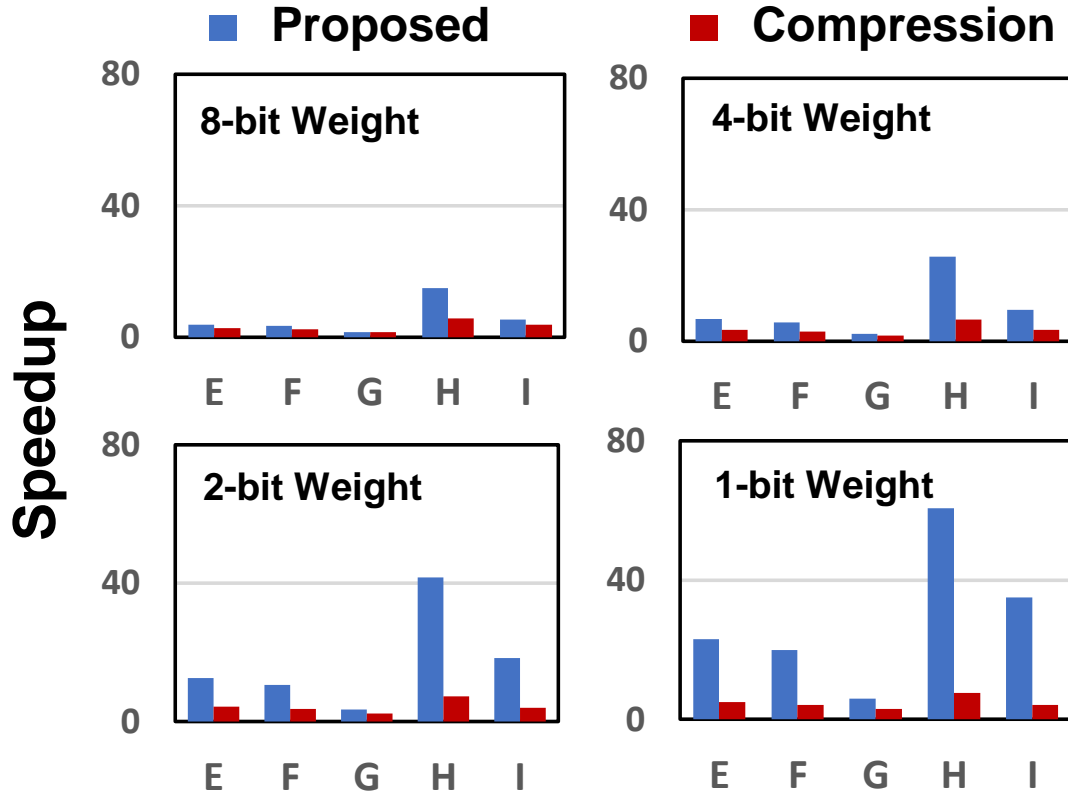


FIGURE 2.8: Performance speedup of MobiLattice for different weight quantization scheme.

#### 2.4.4 Impact of ADC Resolution

In this subsection, we present the performance of NVPIM blocks under different ADC resolution to explore the design space. We fix the weight precision to 8-bit to exclude the impact of quantization scheme.

In our baseline model, we use a 4-bit ADC to sample a bitline current which represents a 4-bit logic value or 1 bit vector inner-product with length of 15. If the resolution of ADC is higher than 4-bit, more rows can be activated in parallel to achieve a higher throughput. Figure 2.9 plots performance speedup comparison under different ADC resolution schemes. As expected, for both the proposed digital mode NVPIM design and the compression approach, the speedup degrades with the ADC having a higher resolution. For example, the

speedup decreases by  $1.2\times$  when increasing the ADC precision from 4-bit to 8-bit.

However, 8-bit ADC is an over-idealized design as mentioned in SRE [176]. In state-of-the-art ReRAM-based NVPIM design, 4-bit resolution ADC is preferred to preserve the computation accuracy. Hence, the digital mode NVPIM design is able to achieve good speedup in a practical scenario.

It is worth noting that there are other hardware parameters that may affect the ADC resolution. For example, the ADC resolution is the sum of the precision of multiple hardware components that participate in the computation, or:

$$R_{ADC} = R_{xb} + R_{in} + \log_2(N_{in}) - 2. \quad (2.10)$$

Here,  $R_{ADC}$ ,  $R_{xb}$ ,  $R_{in}$  and  $N_{in}$  represent the resolution of ADC, the bit levels in ReRAM cells, input bit levels of voltage and the number of wordlines activated in parallel, respectively.

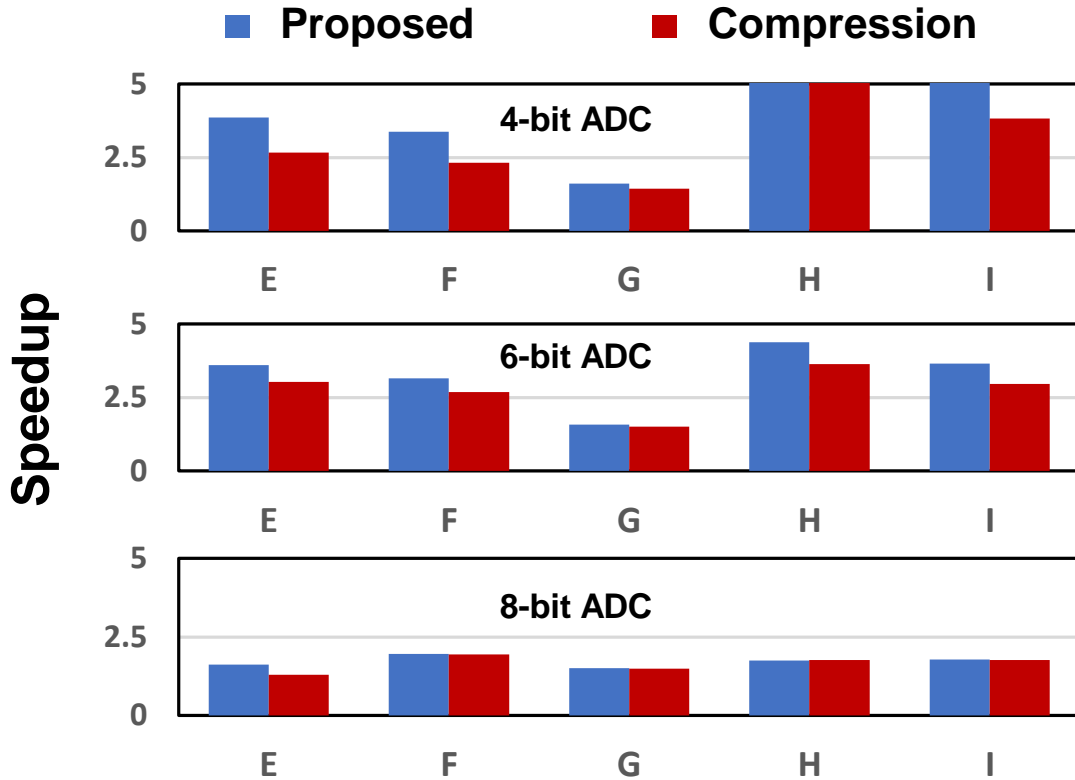


FIGURE 2.9: Performance speedup of MobiLattice for different ADC resolution.

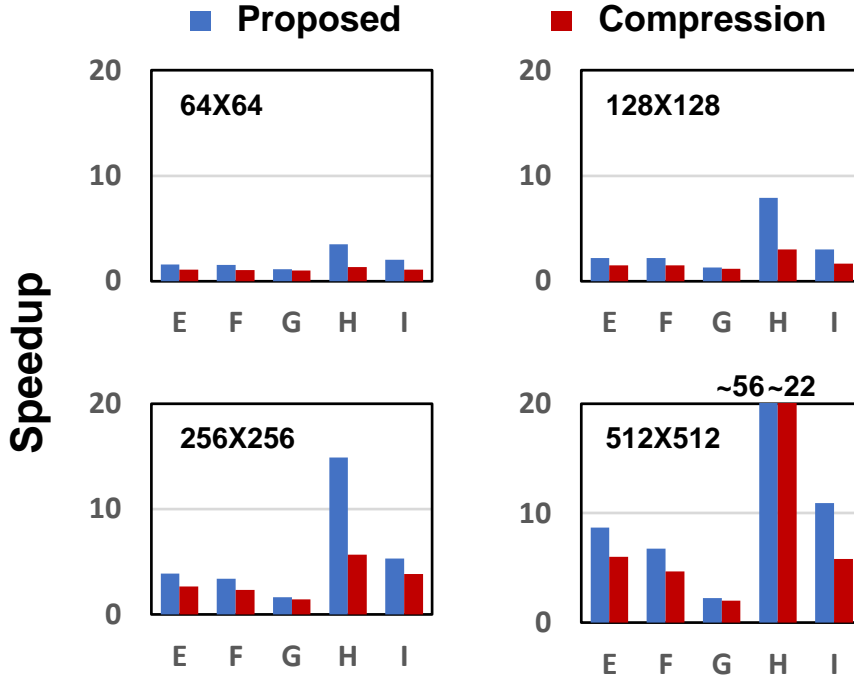


FIGURE 2.10: Performance speedup comparison of MobiLattice for different crossbar sizes.

If the bit levels of input voltage or the bit levels in ReRAM cells increase, fewer wordlines can be activated in parallel when the resolution of ADC is fixed.

### 2.4.5 Impact of Crossbar Sizes

In this subsection, we present the performance speedup under different memory crossbar sizes. The ADC resolution and weight quantization schemes follow the baseline settings. Figure 2.10 plots performance speedup comparison under different crossbar sizes. The performance speedup for both the proposed digital mode design and compression approach can benefit from larger crossbar sizes. The performance speedup can reach  $56\times$  for  $512 \times 512$  crossbar configuration. The minimum speedup is  $3.5\times$  for  $64 \times 64$  crossbar configuration. The redundant space of the baseline would increase when the depth-wise convolution layer is mapped to the crossbars with larger sizes. An extreme case is that if the crossbar configuration is only  $(9 \times 8)$ , there would be no redundant space for the baseline; its performance, hence, would be higher than the proposed digital mode design.

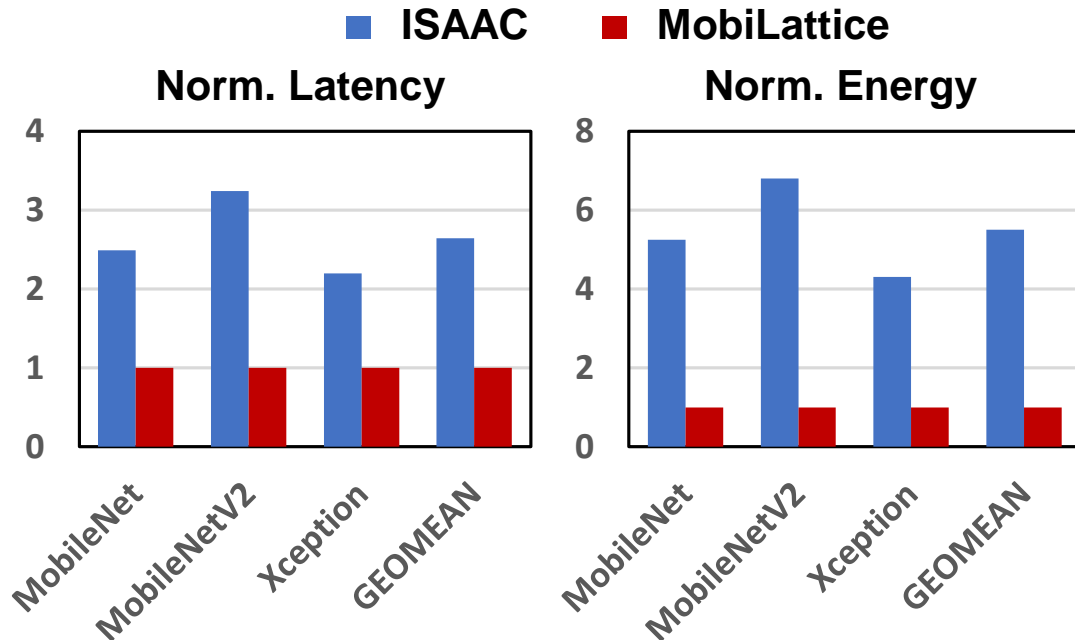


FIGURE 2.11: Performance and energy comparison to ISAAC.

### 2.4.6 Comparison to State-of-the-art Design

ISAAC [134] is a representative state-of-the-art NVPIM architecture which only supports analog mode computation. Figure 2.11 illustrates the performance comparison on three depth-wise DCNN models. As shown in this figure, MobiLattice can achieve 2.2 – 3.2 $\times$  (2.64 $\times$  on average) speedup on these depth-wise DCNN models than ISAAC. MobiLattice can achieve 0.14 ~ 0.22 $\times$  energy saving compared to ISAAC.

## 2.5 Conclusion

In this paper, we propose a NVPIM architecture, called MobiLattice, to improve the inference efficiency of depth-wise DCNNs. MobiLattice can support both digital mode and analog mode computation with negligible hardware overhead. The computation of depth-wise convolution can be processed in digital mode using a unique computation flow to mitigate the redundant memory space. MobiLattice can speedup the processing of typical depth-wise DCNNs by 2 – 5 $\times$  on average and up to 30 $\times$  for some extreme quantization

schemes, compared to conventional designs with only analog mode, i.e., ISAAC.

### 3. Hybrid Digital/Analog IMC for Sparse Attention

With the development of machine learning, attention-based neural networks are being viewed as the next generation solution for computer vision (CV) [51], natural language processing (NLP) [47] and other wide range of applications including hardware security [28, 26, 25]. In a typical attention-based model, for example, *transformer* [74], each element of the input sequence (i.e., token) is embedded into a feature vector. Before the self-attention operation, the input vectors are first projected into query (Q), key (K) and value (V) matrices through linear transformation. Then, self-attention is operated in two consecutive stages. In the first stage, attention computation is done, where an attention map is generated by multiplying Q and K, and goes through the softmax operation for normalization. In the second stage, the output computation is done, where the normalized attention map is multiplied by the V matrix to generate the output. The attention operation has a high computation cost that increases quadratically with the length of the input sequence. This computational complexity hinders the deployment of attention-based model to resource-constrained devices like wireless implant [1].

From the algorithm perspective, researchers propose several methods to reduce computation complexity and memory footprint of the attention operation. One representative method is sparse attention [11, 69], where redundant attention scores are identified and eliminated, generating a sparse attention map. The computation complexity can be significantly reduced by exploring the sparsity in the attention map. Among the sparsity patterns, the dynamic unstructured sparse attention could achieve an optimal trade-off between computation savings and accuracy since the sparsity pattern is tailored for each input sample with no restrictions. The computation complexity can be significantly reduced as the sparsity level could reach 90%..

From the hardware perspective, researchers have developed various hardware architectures to support the attention-based models [108, 124, 59, 153]. Among the different approaches, previous works, such as ReTransformer [178], reveal the superior advantages of non-volatile processing-in-memory (NVPIM) based architectures on boosting the through-

put and energy efficiency of attention-based models. The NVPIM architecture can perform the vector-matrix multiplication (VMM) operations efficiently via analog computing on the memory array. Furthermore, it can substantially reduce the overhead of data movement, as the computation happens *in-situ* inside the memory array. Although the NVPIM architecture is promising for dense computation, it is difficult to support the dynamic unstructured sparsity pattern for its irregular and unpredictable nature. Some existing works, such as ReCom [76], exploit the sparse processing on the NVPIM architecture with a coarse-grain structured pruning method for convolutional neural networks (CNNs). SRE [176] extends the method in ReCom by applying a structured pruning method at the bit-level for both activations and weights. However, no existing work explores the approach to support a dynamic and unstructured sparsity pattern for attention-based model in the NVPIM architecture.

As shown in Figure 3.1, directly mapping a *transformer* model with dynamic and unstructured sparse patterns to the ReTransformer architecture will only induce about  $1.59\times$  ( $1.0\times$ ) speedup under 95% (67%) sparsity level. Even equipping the baseline with the method proposed in SRE, the speedup only reaches about  $1.84\times$  ( $1.11\times$ ) under the same sparsity level, which is far from the theoretical savings. As the native processing granularity of NVPIM is matrix, the utilization rate of the NVPIM banks will drop significantly under the sparse scenario. Furthermore, there is a significant workload imbalance in both attention map computation and output computation due to the strong locality of the attention map. Thus, the bank that finishes earlier has to wait for all banks, further lowering the overall utilization rate.

To fill this gap and release the potential of the NVPIM architecture to process dynamic and unstructured sparse patterns, we propose SparseLattice, a NVPIM architecture with a reconfigurable NVPIM bank to accelerate the sparse attention-based model. To the best of our knowledge, our design is the first work to address the challenging unstructured and dynamic sparsity processing on the NVPIM architecture. We use an architecture-circuit co-design method to improve the intra-bank mapping efficiency of both SDDMM and SpMM.

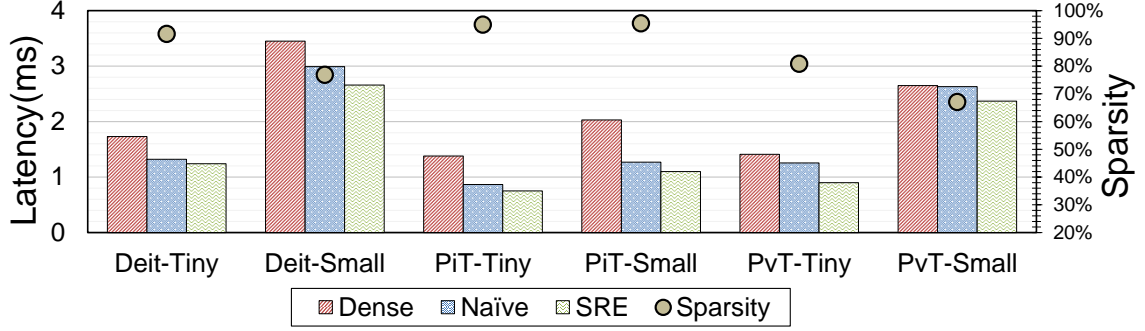


FIGURE 3.1: Limited latency reduction of the previous NVPIM architecture with sparse attention models.

The basic idea is to provide two additional vector-based computation primitives to support the sparse computation pattern by reusing the interface circuits in the conventional NVPIM bank. Further, based on our bank-level innovation, we further present a hybrid stationary dataflow to achieve a pipelined processing to hide the latency of the SDDMM stage and the SpMM stage. We also present two schemes to further improve the inter-bank mapping efficiency, including a greedy scheduling scheme for SDDMM, and a task redistribution scheme for SpMM. The evaluation result shows that our design achieves up to  $12.36\times$  performance improvement compared with the conventional NVPIM architecture with an up to  $3.4\times$  energy efficiency improvement. In addition, our design also achieves up to  $8.6\times$  energy efficiency improvement over the non-PIM design without sacrificing the processing latency. In summary, the contributions can be summarized as follows:

- We propose SparseLattice, a NVPIM architecture with a reconfigurable NVPIM bank that features dense processing with conventional matrix-based primitives and sparse processing with specialized vector-based primitives. By introducing a hybrid analog/digital computation mode with peripheral circuit reuse, our reconfigurable PIM bank improves the efficiency of intra-bank mapping with inconsiderable design overhead (Section. 3.2).
- Based on our innovation at the bank level, we also present a hybrid stationary data

flow which enables a pipelined processing of different stages while minimizing the data movement (Section. 3.3).

- We further propose a greedy scheduling scheme and a task redistribution scheme to improve the efficiency of inter-bank mapping considering the sparse pattern of the attention map. Based on the proposed scheme, further speedup can be achieved for large scale NVPIM architecture (Section. 3.4).

### 3.1 Backgrounds

#### 3.1.1 Sparse Attention in Transformer

Attention-based neural networks, especially *Transformers* [47, 51], have been used in CV and NLP tasks thanks to their capability of capturing semantic dependencies among word tokens. The *transformer* model utilizes self-attention, where the input sequence of token embeddings  $X_{in}$  is first converted into a queries ( $Q$ ), keys ( $K$ ) and value ( $V$ ) matrices by multiplying them with three weight matrices  $W^Q$ ,  $W^K$  and  $W^V$ . The output is computed as a weighted sum of the values, where the weight of each value is computed by a similarity function of the query with the corresponding key. We compute the matrix of outputs using

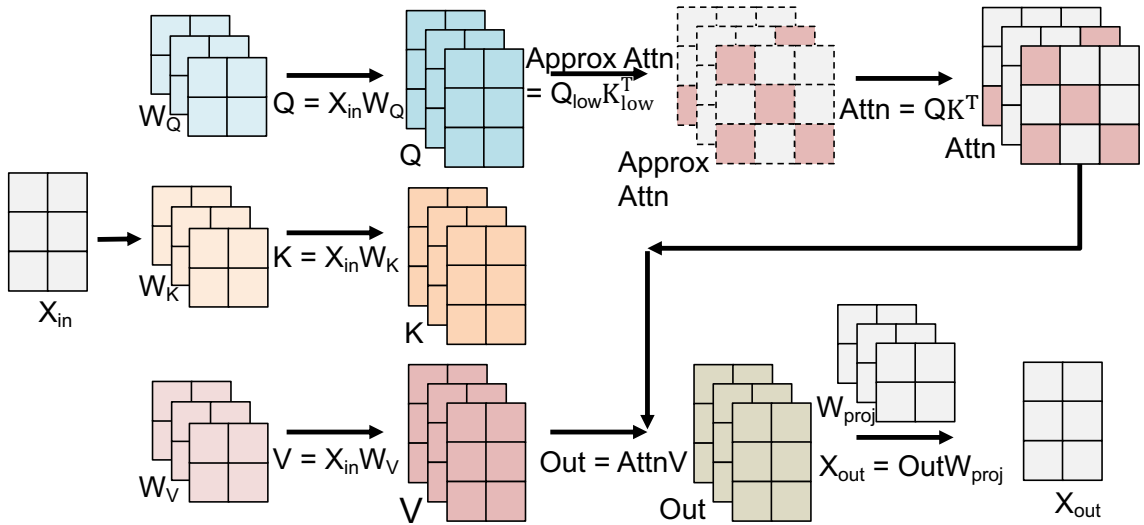


FIGURE 3.2: The computation process of a typical sparse attention block.

the scaled dot-product attention layer:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (3.1)$$

Here,  $d_k$  is a pre-defined constant and  $QK^T/\sqrt{d_k}$  is also called *attention map*.

Researchers attempted to reduce the computation related to the attention map by exploiting the sparsity [43, 11, 153]. By eliminating unimportant attention scores, sparse attention could maintain the important correlation between tokens while significantly reducing computational complexity. With a sparse attention map, matrix multiplication between  $Q$  and  $K^T$  is converted into sampled dense-dense matrix multiplication (SDDMM) while the weighted sum of values turns into a sparse-dense matrix multiplication (SpMM). The computation process of a sparse attention block is illustrated in Figure 3.2. One representative approach proposed in [108] adopts a precision gating method to generate the attention mask. An approximated attention map is computed by performing dense multiplication with low-precision  $Q$  and  $K$  matrices. Then, the attention mask is generated by comparing the approximated attention map with a predefined threshold. In addition to precision gating, [124] adopts a low-rank decomposition method to generate the approximated attention map. In summary, to utilize the attention mask and reduce computational cost, we need to design specialized hardware units to efficiently support SDDMM and SpMM operations.

### 3.1.2 NVPIM Basic

In this paper, we refer processing-in-memory (PIM) as the design where the computation is performed by the memory array. We select ReRAM as our target device because of its high density and energy efficiency, as shown in previous designs for other neural networks [35, 134]. Typically, the memory cells are organized into arrays with the crossbar structure. The PIM array supports VMM as the computation primitive. The elements of the matrix are stored in the memory array as the conductance of the memory cells. The vector is first fed into the input register of the bank. Then, each element of the vector is converted into

the voltage generated by the wordline driver and applied to the corresponding wordlines of the array. The accumulation result can be represented as the current on the bitline. The current is then converted to digital domain by an analog digital converter (ADC) or multi-bit sense amplifier (MBSA) for the subsequent process at the local PE. The final results are stored in the output register for further accumulation or external access.

For practical NVPIM designs, the bank-level throughput is limited by some factors including area/power budget and processing/programming variations. The input parallelism is determined by the number of wordlines that can be turned on in parallel. Turning on too many wordlines may lead to non-negligible errors in the accumulation current, resulting in the distortion of the output. The output parallelism is bounded by the maximum number of ADCs that can fit into the area and power budget of the design. Since the ADCs, especially high-precision variants, are expensive in terms of both area and power, it is impractical to activate too many columns in parallel. For example, in state-of-the-art designs [71], the memory array size is  $512 \times 512$ . For each cycle, only 8 wordlines are turned on, while only 64 bitlines are sensed in parallel instead of activating the whole array simultaneously.

### 3.1.3 Challenges

Despite the practical parallelism constraints, prior work SRE [176] finds that the NVPIM bank with smaller parallelism could deliver similar throughput with the idealized design in [134]. On the one hand, the smaller parallelism could lead to a smaller area by eliminating high precision ADC, so more banks could be available under same area budget. On the other hand, the smaller parallelism of the NVPIM bank enables to use a structural pruning method to avoid storing and computing the filter with zero weights, instead of activating the whole memory array, However, the success to support structural sparsity does not transfer to the sparse operations in the attention-based model, like SpMM and SDDMM. For one NVPIM bank, the sparse computation pattern leads to low utilization rate, which originates from rigid input/output dimensions of the VMM primitive. We can abstract each PIM bank into a  $M \times N$  PE array which computes the inner product

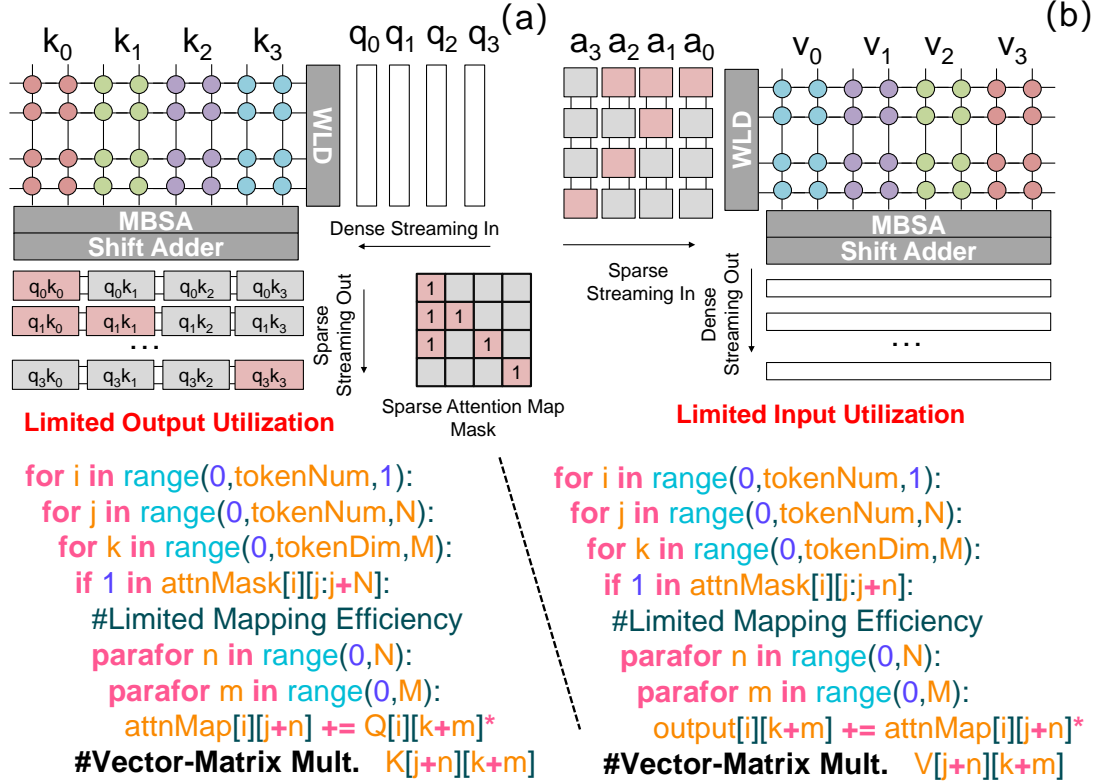


FIGURE 3.3: Illustration the inefficiency to map the (a) attention computation stage and (b) output computation stage to the conventional analog PIM bank.

between a  $M \times N$  matrix and a  $1 \times M$  vector in each cycle. As shown in Figure 3.3(a), in the attention map computation stage, the  $k$  vectors are stored in the memory array, and the  $Q$  vectors are streamed into the NVPIIM bank, where the output attention map matrix is streamed out. At each cycle,  $M$  elements in a  $Q$  vector can be computed with  $N$   $K$  vectors. In this case, the NVPIIM bank is fully utilized only when  $N$  continuous indexes in the attention map mask is valid ( $\text{attnMask}[i][j:j+N]$  is all '1'). In another word, the output dimension of the NVPIIM bank cannot be fully filled, which will reduce the effective output parallelism. Similarly, for the output computation stage (shown in Figure 3.3(b)), the  $V$  vectors are stored in the memory array and the sparse attention map matrix is streamed into the NVPIIM bank. At each cycle,  $M$  elements in a row of the attention map are computed with  $N$   $V$  vectors. We need to ensure  $N$  continuous indexes in the attention map mask are valid ( $\text{attnMask}[i][j:j+N]$  is all '1') to fully utilize the PE array. In another word, the input

### Attention Map Computation

```

for i in range(0,tokenNum,1):
  for j in range(0,tokenNum,1):
    for k in range(0,tokenDim,M*N):
      if attnMask[i][j] == 1:
        #Good Mapping Efficiency
        parafor n in range(0,M*N):
          attnMap[i][j] += Q[i][k+n]*
        #Vector-Vector IP  K[j][k+n]

```

### Output Computation

```

for i in range(0,tokenNum,1):
  for j in range(0,tokenNum,1):
    for k in range(0,tokenDim,M*N):
      if attnMask[i][j] == 1:
        #Good Mapping Efficiency
        parafor n in range(0,M*N):
          output[i][j+n] += attnMap[i][j]*
        #Scalar-Vector Mult.  V[k][k+n]

```

FIGURE 3.4: The nested-loop expression of IP and SVM primitives on attention map computation and output computation stage.

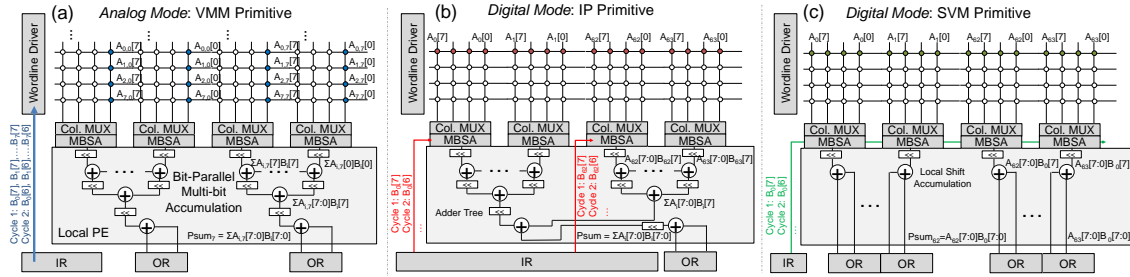


FIGURE 3.5: Illustration of the computation path of the PIM bank for (a) VMM primitive, (b) IP primitive and (c) SVM primitive.

dimension is not fully filled with a reduction of the effective input parallelism. In general, for current NVPIM architecture, the computation primitive provided by the NVPIM bank is not suitable for sparse processing in the attention model.

## 3.2 Reconfigurable PIM Bank

### 3.2.1 Design Principle

In this section, we present the key innovation to address the mapping inefficiency problem. As discussed before, the conventional NVPIM bank only provides the primitive of VMM with fixed input and output parallelism. To improve the mapping efficiency, the basic idea is to collapse the input or output dimensions according to the presence of sparsity. We still abstract each PIM bank into a  $M \times N$  PE array. However, we assume that the PE

array can be dynamically configured to compute the inner product between two  $1 \times MN$  vectors or multiplication between one scalar and one  $1 \times MN$  vectors. As shown in Figure 3.4, in the attention map computation stage, the output dimension is sparse. Instead of mapping the sparse dimension into the spatial parallelism, we map the sparse dimension in a temporal loop. For each computation cycle, we compute one non-zero value in the sparse attention map from the IP between one  $q$  and  $k$  vectors ( $\text{attnMask}[i][j]=1$ ). The computation corresponding to the pruned attention scores can be skipped. Since token dimension is a relatively large value (e.g. 64, 128) and  $M, N$  is a small value (e.g. 8), the  $M \times N$  PE array can be fully utilized. Similarly, when the input dimension is sparse, the input is viewed as a scalar and each computation cycle produces one output vector corresponding to the product of the non-zero attention map and one  $V$  vector. For each computation cycle, we feed one non-zero value in the sparse attention map as a scalar and perform a SVM between the scalar and one specific  $v$  vector. The computation corresponding to the zero-value attention map can be skipped.

Following this micro-arch design principle, we need to have some requirement of the NVPIM bank design methodology, where each bank should be reconfigured to support one of these three primitives. In SVM mode, the PIM bank receives a scalar as the input, multiplies it by the vector stored in the bank, and produces a vector as the output. In the IP mode the PIM bank receives a vector as the input, computes the dot-product of

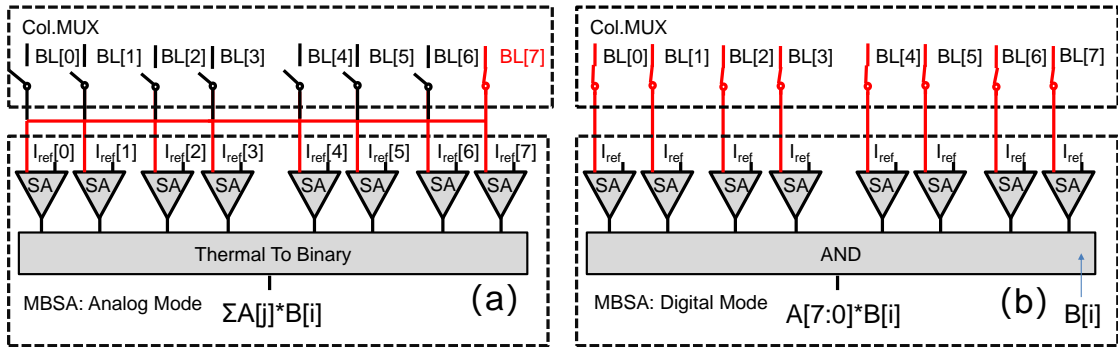


FIGURE 3.6: The data path of Col.MUX and MBSA in (a) analog mode and (b) digital mode.

the input vector and the stored vector, and produces a scalar as the output. In addition, the throughput delivered by each bank should be maintained as it in the original design. Thus, we need to scale up the vector dimension in these two primitives. For example, if the original PIM bank provides the VMM primitive of the shape  $M \times N$  (i.e., input length is  $M$ , output length is  $N$ ), our reconfigurable PIM bank design will provide the SVM primitive with the shape of  $1 \times MN$ , or the IP primitive with the shape of  $MN \times 1$ .

### 3.2.2 Hybrid Digital/Analog PIM Bank

The proposed PIM bank contains an I/O register (IR/OR), wordline drivers (WLD), one or multiple memory array(s), read out circuit (including column multiplex (Col.MUXs) and MBSAs), and a local processing element (PE). As aforementioned, the PIM bank in our design should support three primitives, VMM, SVM, and IP. The VMM is the primitive provided by conventional PIM designs. Since the accumulation is performed in the analog domain, we refer to the configuration as analog mode. For SVM and IP, the accumulation happens outside the memory array in the digital domain. Thus we denote these configurations as digital modes. The objective of our PIM bank design is to support the reconfiguration between the analog and two digital modes, and between the SVM and the IP within the digital modes.

Before discussing how to reconfigure the PIM bank, we discuss the available resources in the conventional analog PIM bank. Figure 3.5 (a) shows the bank-level architecture and the detailed computation process of the conventional analog PIM bank. Here, we denote the  $M \times N$  matrix stored in the memory array as matrix  $A$ , the  $M \times 1$  input vector as vector  $B$ , and we assume that the precision of both  $A$  and  $B$  are 8-bit. Matrix  $A$  is assigned to the corresponding  $M$  rows and  $N \times 8$  columns in the memory array. The bits for one element in  $A$  is interleaved to different columns with a stride of 8 to ensure the parallel computation. In each cycle, one bit of each element in  $B$  is fetched from the input register to the wordline driver that activates  $M$  wordlines in parallel to take in these  $M$  1-bit inputs. The current at each bitline will represent a bit-wise multiplication and accumulation (MAC)

result between B and one column of A. Along each bitline, one MBSA converts the bitline current to a multi-bit digital value. In total,  $N \times 8$  MBSAs are required to generate  $N \times 8$  bit-wise MAC results. For every 8 MBSAs, the bit-wise MAC results are further shifted and accumulated through an 8-to-1 adder tree and then sent to the shift-accumulator collecting the partial sum. To perform an  $M \times N$  8b VMM in 8 MBSA cycles (we denote the MBSA cycle as the latency of one sensing operation), we need  $N \times 8$  MBSAs,  $N$  8-to-1 adder trees and  $N$  shift-accumulators. The total number of adders is  $N \times 8$ . For the interface circuits, each MBSA requires  $M$  levels to accurately sample the computation results in the analog domain, thus, each MBSA contains  $M$  sense amplifiers (SAs) with  $M$  different reference levels, as shown in Figure 3.6 (a). The Col.Mul is set to select a specific bitline to the MBSA, to implement the stride-8 bit-interleaved scheme. A thermal-to-binary decoder is needed to generate a data with binary format for further processing.

The basic idea of our proposed reconfigurable PIM bank is to implement the digital modes (SVM and IP) with an alternative near-memory computation path. By reusing the resources available in the conventional analog PIM bank, we are able to minimize the area overhead incurred by supporting the digital modes. The computing paradigm of the IP primitive is shown in Figure 3.5 (b). The key to support IP primitive is to configure the MBSA into multiple SAs by providing same reference voltage to each SA, as shown in Figure 3.6 (b) (At the same time, the thermal-to-binary decoder is by passed). The Col.Mux is set to connect 8 bitlines to select multiple bitlines in parallel to multiple SAs. Here, we denote the vector stored in the memory array as vector A, and the input vector as vector B. Both vectors have the shape of  $MN \times 1$ . Each bit in one element of A is directly stored in 8 consecutive bitlines ensure the parallel read. Before the computation starts, the vector A is first read out by activating one specific wordline. In this case,  $N \times 8$   $M$ -level MBSAs can be used to sense  $N \times 8 \times M$  bitlines in parallel. We could read out vector A with length of  $MN$ , where each element in the memory array is 8 bit. To reuse the adders in the analog mode, we still feed the vector B bit-by-bit to the MBSA.  $M \times N$  element-wise multiplication results can be generated by performing an AND operation between the sensed vector A and

each bit of vector B. As the length of vector B is  $MN$ , we need an  $MN$ -to-1 adder tree to generate the MAC result by accumulating the element-wise multiplication results. We need one shift-accumulator to accumulate the bit-wise MAC results corresponding to different input bits. Thus, the total number of the adders is  $MN$ .

The computing paradigm of the SVM primitive is similar to the IP primitive, as shown in Figure 3.5 (c). Here, we denote the vector stored in the memory array as vector A and the input scalar as B. The MBSA is also configured into multiple SAs and a vector A with length  $MN$  is read out. The scalar B is broadcast bit-by-bit to the all MBSAs, and then an AND operation is performed to generate  $MN$  8b element-wise multiplication results. In this case, we require  $MN$  shift-accumulators to accumulate the bit-wise MAC results corresponding to different input bits.

The two digital modes we introduce could maintain the original throughput in the analog mode, if we ensure that  $M$  equals the precision of the operands ( $N \times 8$  adders V.S.  $M \times N$  adders). Considering the IR-drop, resistance variation problem in the ReRAM device and interface circuit complexity,  $M$  usually equals to 8 [71, 19]. Since we reused most of the components in the conventional PIM bank, the introduction of digital modes incurs negligible area overhead while maintaining the original throughput.

### **3.3 Accelerator Architecture**

#### **3.3.1 Architecture Overview**

Figure 3.7 depicts the overall architecture of SparseLattice. We organize the aforementioned reconfigurable PIM bank into tiles. Each tile can communicate with the external memory or the other tiles through a shared interface. Within each tile, apart from the reconfigurable PIM banks, we also built a local buffer, a functional unit, and dense/sparse schedulers. The sparse attention computation can be decomposed into multiple dense and sparse matrix computation stages. The PIM banks in the tile are used for both dense and sparse matrix computation. The analog mode is used to perform a VMM for the dense matrix operation, and the digital modes are used to perform IP or SVM for the sparse matrix

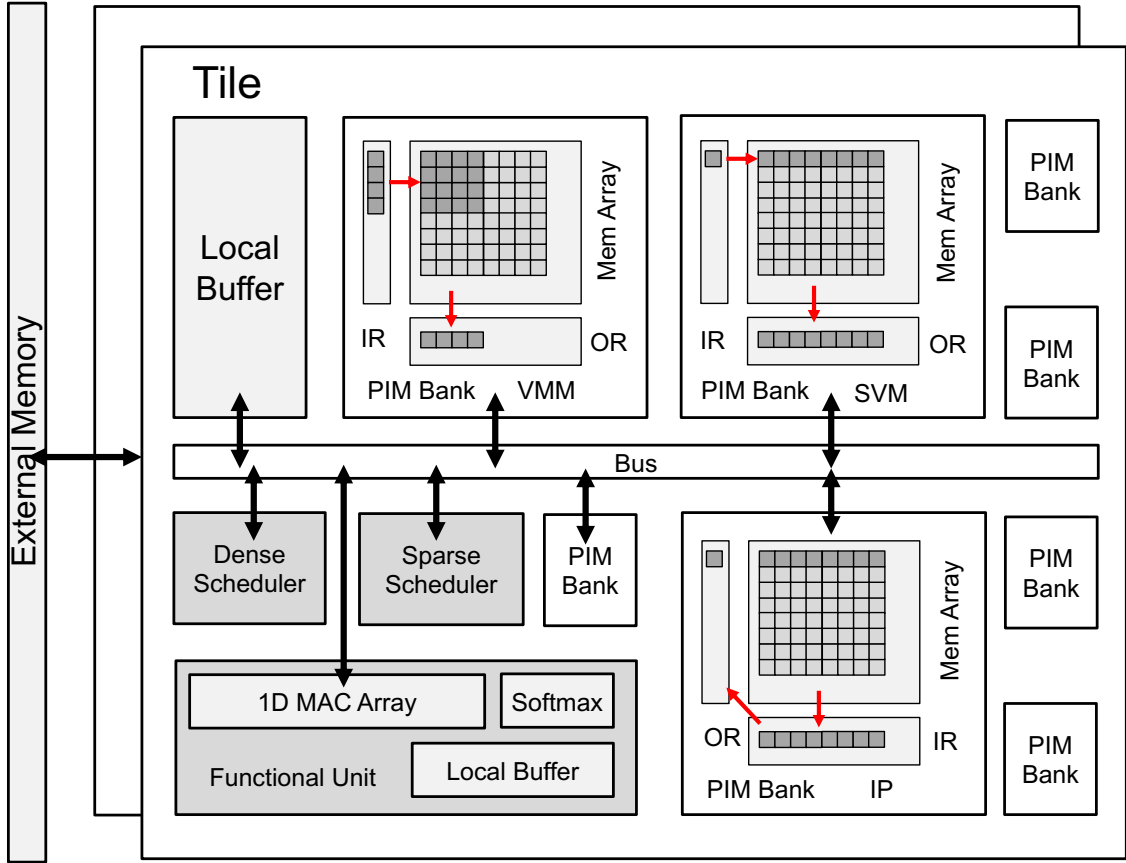


FIGURE 3.7: The overall architecture of SparseLattice.

computation. The functional unit is used to perform softmax and normalization operations. Intermediate data, such as embeddings ( $X$ ), are stored in the local buffer. Finally, a dense and a sparse scheduler are used to orchestrate the computation flow by moving the data and assigning the computation task to each PIM bank.

Our design features a hybrid stationary dataflow that distributes the computation of different stages into different PIM banks to fully utilize the computation resources. Moreover, we could enable a dedicated pipelined execution scheme among different stages to reduce processing latency.

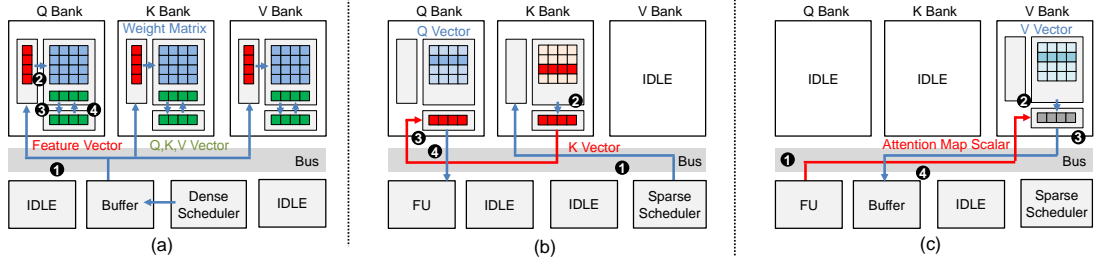


FIGURE 3.8: An example of data mapping and computation flow for (a) dense operation, (b)  $Q \times K^T$  and (c)  $\text{Attn} \times V$ .

### 3.3.2 Computation Flow

We would like to use a simple example to demonstrate how our proposed design executes both dense and sparse computation in the sparse attention operation. For simplicity, we assume that there are three banks available.

#### 3.3.2.1 Dense Computation

The dense operation includes the QKV generation layer, the prediction of the attention map, the projection layer, and the FFN layer (linear layers). Since they share the same operation (matrix multiplication) with different input/output shapes, we use the QKV generation layer as an example. Before starting the calculation, we divide the PIM banks into Q, K, V bank, and the weight matrices ( $W_q, W_k, W_v$ ) are assigned to the corresponding bank. The computation is performed in token order. According to the token index, an input feature vector is fetched from the local buffer and sent to all Q, K, V PIM banks. Then, all Q, K, V banks are configured into the analog mode to perform a VMM computation. The dense scheduler triggers Q, K, V banks to perform the VMM between the input feature vector and the weight matrices, and the results are temporarily stored at the output register inside each PIM bank. At the end of the dense computation, the results are directly stored back to the memory array in the PIM bank for further sparse computation. The mapping and computation flow of the other dense computation are the similar to the QKV generation layer.

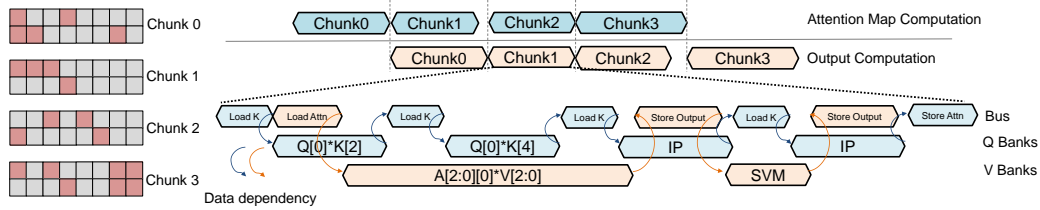


FIGURE 3.9: Pipelined execution scheme in SparseLattice.

### 3.3.2.2 Attention Map Computation

As mentioned previously, the attention map computation stage is a SDDMM pattern. Each output attention score is the result of a dot-product of a pair of query and key vectors. After the QKV generation layer, the Q and K matrices are stored at the Q banks and K banks. We use a “Q stationary” dataflow, where the Q banks are used as the computation bank and the K banks as the memory bank. The mask of the sparse attention map stored at the sparse scheduler. According to indices of K vectors corresponding to non-zero attention score, the sparse scheduler issues the memory access requests to the K bank. The K vector stored in the K bank will be gathered to the sparse scheduler and further scattered to the corresponding Q banks. Then, the Q banks are configured into the vector-vector IP mode, and the sparse scheduler triggers the computation of the corresponding Q bank. According to the Q index, a specific Q vector is selected in the memory array and a vector-vector IP between the Q vector and the K vector is computed to generate one attention score. The generated results are further sent back to the function unit for softmax, and temporarily stored in the attention map buffer in the function unit.

### 3.3.2.3 Output Computation

Having generated the sparse attention map, the next stage is the output computation stage. In this stage, we obtain the result of the attention operation by multiplying the attention map with the V matrix. To reduce the data movement, we use a ‘V stationary’ dataflow. According to the index of V vectors corresponding to non-zero attention scores, the sparse scheduler fetches the attention map from the function unit to the V banks.

Then, the V banks are configured into the SVM mode, and the sparse scheduler triggers the computation. According to the non-zero index of the attention map, a specific V vector is selected in the memory array, and a scalar-vector multiplication is computed. Then, we iterate over all attention score to generates the final output features which are further sent back to the local buffer.

### 3.3.3 Pipeline Scheme

In the attention map computation stage, the Q and K banks are used for data access or computation while the V banks are idle. In contrast, in the output computation stage, only V bank is used for data access or computation. Thus, there is an opportunity to form a pipeline to hide the latency.

Figure 3.9 shows the pipeline scheme in our design. The pipeline contains 2 stages, including the attention map computation and the output computation. We group multiple rows in the attention map into one chunk, and split the attention map into multiple chunks. We first trigger the attention map computation of the first chunk in the attention map. When the attention map of the first chunk is generated, we trigger the output computation of the first chunk and the attention map computation of the second chunk at the same time. As different chunks may contain different numbers of non-zero indexes, we design a pipeline that adapts to the varying stage latency. Two individual controllers are designed inside the sparse scheduler to manage the computation flow for attention map computation and output computation. The computation of the subsequent stage is triggered only when both the attention map computation and the output computation of the previous stage are completed. Other than latency reduction, a chunk level pipeline could also reduce the attention map storage requirement. Instead of buffering the whole attention map, we only need to buffer the attention map for two chunks.

We illustrate an example of the proposed pipeline in Figure 3.9. Chunk 2 proceeds to the attention map computation stage while chunk 1 in the output computation stage. To avoid the bus conflict, we interleave the memory accesses of the two stages. For example, in

the first cycle, the bus is used to gather  $K$  vectors from  $K$  bank and scatter the  $K$  vector to corresponding  $Q$  bank. After the gather/scatter operation is completed, the bus is released for output computation. Then, we could switch the bus to move the data to the  $V$  bank for the output computation stage. The latency of data movement on the shared bus can be ignored comparing with the computation latency in this pipeline interleaving scheme.

### **3.4 Architectural Optimization**

In the previous section, we discussed the mapping and computation flow with only three PIM banks. However, the actual workload requires the coordination of more PIM banks. In this section, we would like to discuss the key innovations to improve the inter-bank mapping efficiency (i.e., bank-level parallelism). As the sparsity pattern of the attention mask is unpredictable, it is sub-optimal to directly map the sparse model into multi-banks due to the workload imbalance problem. Moreover, since the attention map computation stage and the output computation stage are pipelined, we need to accelerate both stages to achieve a performance gain. We propose a dedicated scheduling scheme for each stage to address the imbalanced workload issue caused by sparsity.

#### **3.4.1 Greedy Scheduling Scheme**

When we have multiple banks, we split the original  $Q$  matrix in the token number dimension and map the sub-matrix to each  $Q$  bank. During the attention map computation stage, the scheduler would fetch multiple  $K$  vectors to different  $Q$  banks to exploit a token-level parallelism. Figure 3.10 (a) shows an example where the token-level parallelism is 4. Before the computation starts, the attention map mask is stored in the mask buffer. To reduce the storage requirement, we compress the attention map mask by eliminating the all-zero columns. We use a customized buffer array to store the attention mask and an index buffer to store the indices of the key vectors. Prior work [178] uses a serial scheduling scheme that issues tasks key by key. In each cycle, one index of the key vector in the index buffer is selected and the corresponding key vector is sent to corresponding  $Q$  banks with non-zero attention map mask. However, since not all  $Q$  banks can utilize the selected vector

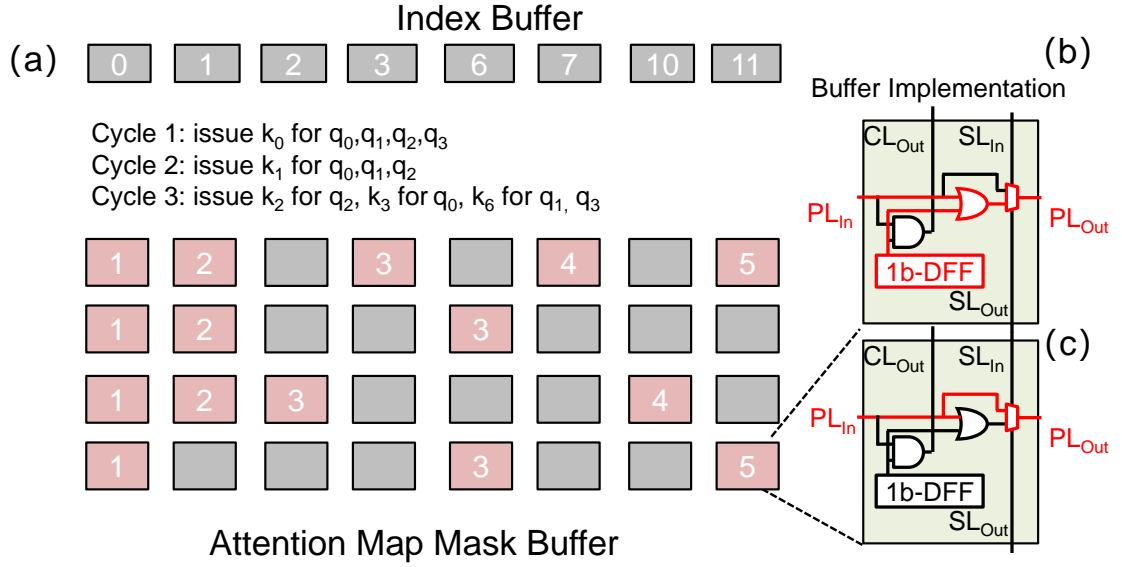


FIGURE 3.10: The greedy scheduling scheme, including the schematic of the attention map mask buffer and the index buffer.

in sparse scenario, this scheme will result in a low utilization rate of Q banks.

Our proposed scheduling scheme aims to explore the possibility of processing multiple keys in parallel. As shown in Figure 3.10 (a),  $k_2$ ,  $k_3$  and  $k_6$  can be computed in parallel, since the Q vectors they used are located in different banks. On the contrary,  $k_1$  and  $k_6$  cannot be computed in parallel since they both require  $q_2$  for the computation. Generating a global optimal scheduling result is a dynamic programming problem with high computational cost. Thus, we use a greedy scheduling method to reduce the hardware cost to generate a near-optimal scheduling result. In each iteration, we first issue a task to involving the first key in the index buffer. Then, we traverse all the other keys in the index buffer and check if there is a bank conflict with the scheduled keys. The computation of new keys can be scheduled if no bank conflict occurs. The scheduled keys will be removed from the index buffer and the mask buffer for the next iteration. The pseudo-code representation of the scheduling scheme is shown in Algorithm 1.

Figure 3.10 (b) and (c) depicts the implementation of our proposed scheduling scheme.

---

**Algorithm 1** Greedy Scheduling

---

**Require:** Q bank parallelism  $M$ , a  $M \times N$  attention map mask and an key index list with  $N$  indices

**Ensure:** A computation order queue with a set of issued query, key pairs

BusyQbanks  $\leftarrow$  zero vector with length of  $M$

**while** key index list is not empty **do**

  Issue & remove keyIndexList[0]

  BusyQbanks  $\leftarrow$  attnMapMask[0]

**for**  $i < \text{length}(\text{keyIndexList})$  **do**

**if** 1 in BusyQbanks & attnMapMask **then**

      continue

**else**

      Issue & remove keyIndexList[i]

      BusyQbanks  $\leftarrow$  BusyQbanks + attnMapMask[i]

**end if**

**end for**

**end while**

---

The key component of our scheduler consists of a buffer array and a scheduling logic. The buffer array is organized by  $K \times N$ , where  $N$  denotes to the number of available Q banks and  $K$  indicates the maximum token number. Each cell in the array contains a 1b DFF and conflict checking logics, connected with propagation line (PL), selection line (SL) and conflict line (CL). For each iteration, we propagate a bit vector with length of  $N$  horizontally to the memory array through  $N$  PLs. The  $i$ th bit of this vector represents the availability of  $i$ th Q bank. A simple ‘AND’ operation is performed between the bit vector in the PLs and the attention map mask stored at the DFF to check if there is a conflict between the scheduled keys with the new key. If there is no conflict, the new key vector will be scheduled and the status vector is updated by performing an ‘OR’ operation with the attention map mask (The data path is shown in Figure 3.11(b). Then, the sparse attention map mask of this key is reset to zero to indicated that this key has been scheduled. If there is a conflict, the status vector is directly propagate to the next column of the memory array through PL without any update (The data path is shown in Figure 3.10 (c). The scheduling process is repeated when all the tokens in the attention map buffer are scheduled.

When we increase the token-level parallelism  $N$ , the size of sparse attention map buffer

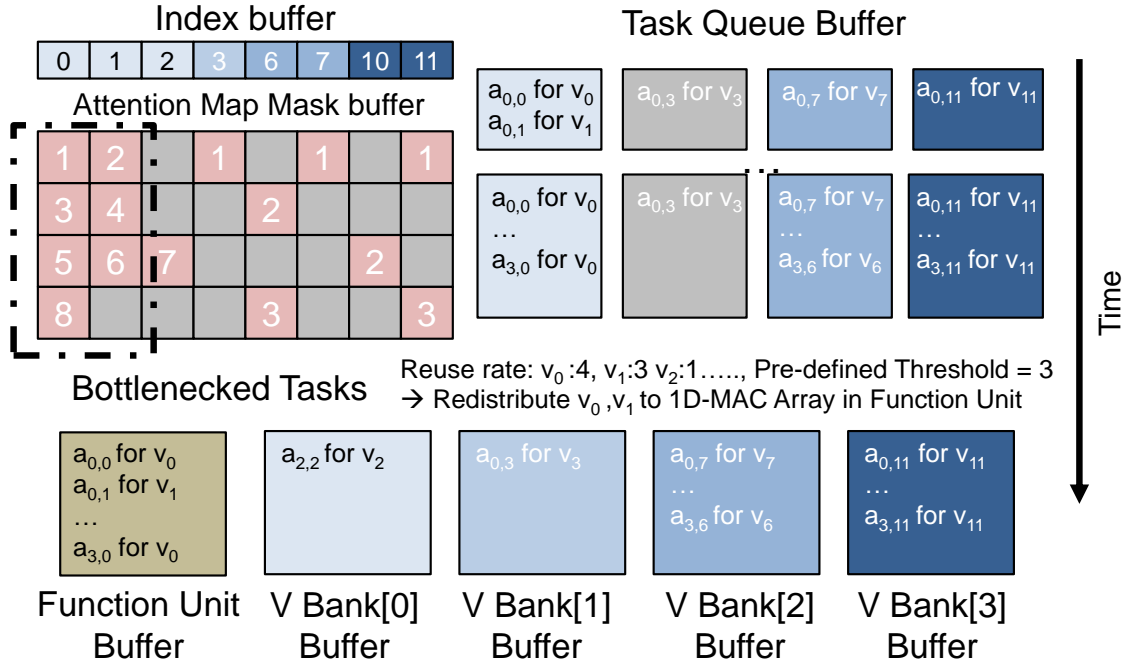


FIGURE 3.11: The task redistribution scheme.

increases linearly. It is worth noting that existing works [124] present an out-of-order scheduling method, which could result in higher utilization than serial scheduling. However, the buffer and scheduling cost increase exponentially in the out-of-order scheduling method when  $N$  increases. As shown in [124], the token-level parallelism cannot be greater than 4, otherwise the scheduling cost will offset its benefit. This method can not be used in the PIM based architecture, as we may have an opportunity to scale up the number of available PIM banks with a large token-level parallelism.

### 3.4.2 Task Redistribution Scheme

Similar to the Q bank-level parallelism, we could also exploit a token-level parallelism by splitting the original V matrix into multiple sub-matrices and mapping them into different V banks. During the output computation stage, the scheduler moves the attention map to different V banks to exploit token-level parallelism. The scheduling process is shown in Figure 3.11. We design a task queue buffer for each V bank, and then the attention

map mask is readout row-by-row. The non-zero index for each V bank is sent to the corresponding task queue buffer, and the attention map is fetched from the function unit to each V bank for the SVM computation. After the results are generated, the data are then moved to the buffer for the next layer.

We observe that a specific pattern where multiple elements in the same column of the attention map dominates the execution time, as we can not execute two SVMs in parallel if the vectors are in the same PIM Bank. This observation originates from the locality of the attention map. If one token is important, the sparsity of that attention map column is relatively lower than the other tokens. An interesting point is that, the pattern which bottlenecks the execution of PIM-based architecture is more suitable for the conventional digital accelerator. As the bottleneck pattern is two or more elements multiplied by the identical token V vector, the token V vector can be reused across multiple attention map elements.

Based on this observation, our task redistribution scheme is designed by leveraging the in-memory processing and near-memory processing with a 1-D MAC array in the function unit. The scheduler estimates the workload cost by calculating the reuse number of each token vector and determines whether a token should be computed in the V PIM Bank or MAC array. If the reuse number of one token is greater than a pre-defined threshold (2 in the example), we will send the V vector to the near-memory MAC array for processing, otherwise the computation remains in the V PIM Bank. For the computation assign to MAC array, the V vector is read out and feed to the function unit. Otherwise the attention map elements are sent to the V PIM Bank from the attention map buffer.

The main benefit of this scheme comes from a fact that the MAC array has lower latency than NV-PIM bank. Thus, redistributing a small amount of workload to the MAC array could reduce the overall latency. Nevertheless, the threshold should be profiled at the compile time to avoid to schedule to much tasks to the MAC array.

## 3.5 Evaluation Methodology

### 3.5.1 Workload Setup

We evaluate our design on 3 vision *Transformer* models and 1 NLP *Transformer* model. For vision tasks, we use 6 models including DeiT (DeiT Tiny, DeiT Small) [147], PVT (PVT Tiny, PVT Small) [154] and PiT (PiT Tiny)[56] on ImageNet [46] dataset, where the input size is  $3 \times 224 \times 224$  and the patch size is  $16 \times 16$ . For the language tasks, we use BERT [47] for GLUE [152], SQuAD v1.1 [126], and CLOTH [167] benchmarks. We use the precision gating method proposed in [108] to generate the attention map mask. The other layer such as linear layer and FFN layers are quantized to 8 bit for both activation and weights. Each vision model is re-trained 60 epochs after quantization and sparsification, to recover the accuracy loss caused by the gating. The language models are fine-tuned on each downstream task based on pretrained BERT model until converge.

### 3.5.2 Hardware Setup

For the NVPIM bank, we use the memory array model from [134]. The energy and area for the adder in local PE is extracted from synthesized results with 28 nm standard cell library. For the analog component such as MBSA, we implement the circuit in transistor level and extract the area and energy consumption from the schematics. The NVPIM bank level area breakdown is shown in Table I. As the MBSA and the memory array consumes

Table 3.1: Area breakdown for our NVPIM bank and the conventional NVPIM bank.

Component	Params.	Spec.	Area(mm <sup>2</sup> )	
			Con.	Ours
Memory Array	Size	$512 \times 512$	0.0128	0.0128
	Number	4		
Local PE	No. of Adders	64	0.0078	0.015
Input Register	Size	2K Byte	0.0021	0.0021
Output Register	Size	256 Byte	0.00077	0.00077
8-level MBSA	Number	64	0.01	0.01
Total	–	–	0.033	0.041

Table 3.2: Hardware Configurations

Component	Params	Spec
NVPIM Bank	Throughput	64 MACs/14 ns
	Comp.Energy	93 fJ/MAC
	Capacity	128 kB
	Mem. Energy	2.3 pJ/Byte
	Number	72
Function Unit	Throughput	64 MACs/ cycle
	Comp.Energy	113 fJ/MAC
Local Buffer	Capacity	512 kB
	Mem.Energy	4.5 pJ/Byte
Local Bus	Bandwidth	512 Byte/cycle

most of the area, our reconfigurable NVPIM bank design could only introduce about 20% area overhead.

The hardware configuration is shown in Table 3.2. The results of the function unit is extracted from [124], and the results of the local buffer is extracted from CACTI [114]. Each PIM Bank could deliver about 9.14 GOPS throughput with 128 kB memory capacity. We set 72 banks in each tile, which could deliver 658 GOPS and 9 MB memory capacity. The number of tiles ranges from 1 to 16, depending on the model size. Based on bank-level results, we further implement a cycle-accurate simulator to capture the total performance and energy consumption for each workload. The simulator is designed to capture the features including memory access conflict, bus conflict, PIM bank conflict and pipelined scheme to guarantee the accuracy of the simulation results.

### 3.5.3 Baseline Setup

For the baseline NVPIM accelerators, we quantitatively compare our design with ReTransformer [178], as implemented with our cycle-accurate simulator. ReTransformer is designed for dense attention computation, so we apply the method proposed in SRE [176] to support the sparse attention. To make a fair comparison, we use the same PIM bank level parameters for the baseline. We also compare our design with two non-PIM based designs. Generally, it is hard to make a fair comparison between NVPIM based design and non-PIM

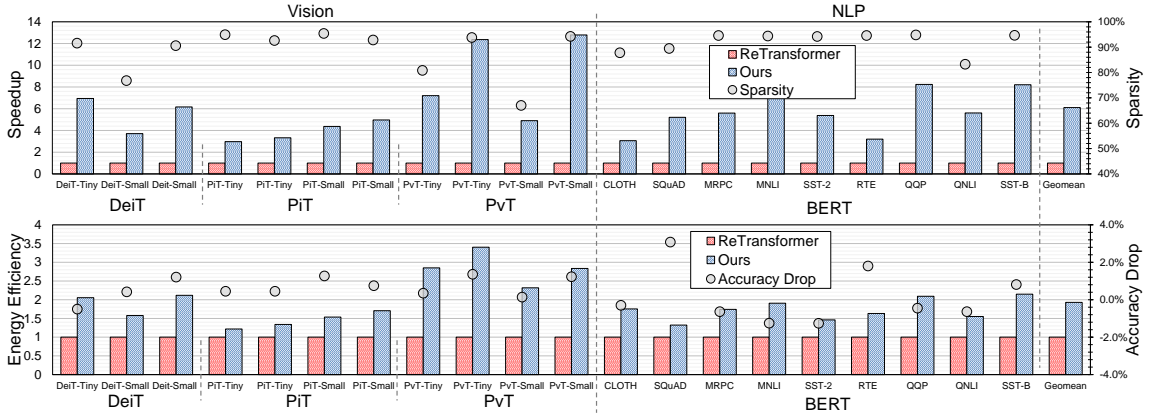


FIGURE 3.12: Speedup and the energy efficiency results on vision and NLP tasks.

based design, since its computation principle is fundamentally different. We make several assumptions to make a fair comparison. Firstly, we set the memory access energy/latency and the computation energy/latency to be same as the NVPIM based designs. Secondly, we select a proper design budget so that both non-PIM and NVPIM designs could deliver the same peak throughput. Thirdly, we ignore energy and latency cost of communication and schedule in non-PIM designs. Thus, we could capture the general latency/energy trends instead of absolute value, and our assumption ensures the results from non-PIM design are more optimistic.

We choose DOTA [124] and Sanger [108] as they use similar sparse algorithm. For these non-PIM based designs, we develop a behavior model to evaluate the latency and energy cost. For DOTA, we scale down the number of lanes in the original design (2TOPS) to match the peak throughput with our baseline non-PIM accelerator (512GOPS), while for Sanger, we directly choose the original hardware configuration since it delivers similar effective throughput (529GOPS). We also scale-down our baseline design to achieve similar peak throughput to 1 tile (658 GOPS).

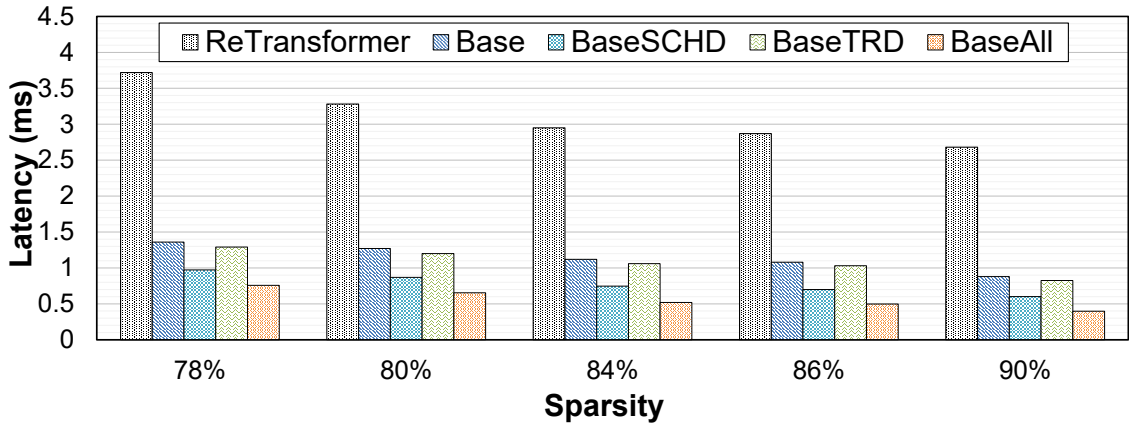


FIGURE 3.13: Ablation study for latency of our designs over the baseline accelerator with different sparsity level on DeiT-Small model.

## 3.6 Results and Discussion

### 3.6.1 Main Results

We separately depict the sparsity and accuracy result as grey dots in Figure 3.12. For most of the models, the sparse attention algorithm incurs negligible accuracy loss ( $< 1\%$ ) while archiving a relatively high sparsity ratio in attention score ( $> 85\%$ ). Sparse attention could even improve the accuracy on simple downstream tasks like SST-2, MNLI, CLOTH, and QNLI, since the pruning process reduce the overfitting of the baseline model. We illustrate the improvement of our proposed design over the baseline NVPIM accelerator in Figure 3.12. The result shows that our proposed design could achieve a speedup ranging from  $2.95\times$  to  $12.36\times$  over the baseline design. The speedup mainly comes from two aspects. Our reconfigurable NVPIM bank could improve the intra-bank utilization rate, while our proposed architectural optimization techniques could improve the inter-bank utilization rate. In terms of energy, our design could achieve a  $1.2\times$  to  $3.4\times$  energy efficiency improvement comparing with the baseline accelerator, as shown in Figure 3.12. The energy efficiency improvement comes from the improvement of the intra-bank utilization rate. Similar as the speedup, we also observe a larger energy efficiency improvement when the sparsity level increases.

### 3.6.2 Ablation Study

To further understand the source of the performance improvement, we perform several ablation studies to identify the contribution of each proposed component. We choose DeiT-Small model with different sparsity levels as a benchmark. Three versions of our proposed design are further implemented. Here, **Base** denotes the design that replaces the PIM banks in the baseline design with our reconfigurable PIM bank. **Base+SCHD** further adds the greedy scheduling scheme. **Base+TRD** adds the task redistribution scheme to the **Base** design. **Base+All** denotes our finalized version with all the architectural optimization. The latency results are shown in Figure 3.13. The reconfigurable NVPIM bank design could bring about 2.2 to 3.3 $\times$  latency reduction to the baseline design (ReTransformer to **Base**). The speedup increases with the sparsity level. For the model with a sparsity level of 78%, we could achieve 2.2 $\times$  speedup, while the speed up increases to 3.3 $\times$  as the sparsity level reaches 90%. This latency reduction comes from the improved PIM bank utilization rate. ReTransformer fails to convert the sparsity into the reduction of computation. In other words, with a higher sparsity level, the bank utilization rate of ReTransformer decreases while the overall latency almost remains the same. In contrast, in our design, the processing latency can be reduced nearly linearly scaled with the sparsity level.

Considering the architectural optimization to improve the inter-banks utilization rate, the greedy scheduling scheme could further provide a near 1.8 $\times$  speedup (**Base** to **BaseSCHD**) by reducing the latency of attention map computation stage. The task redistribution scheme could provide around 1.5 $\times$  speedup (**BaseSCHD** to **BaseAll**) from output computation stage. When we combine both techniques, the effective throughput could be further improved to 2.75 $\times$  (**Base** to **BaseAll**). This is because the output computation stages becomes the bottleneck when the attention map computation stage latency has been reduced by the greedy-based scheduling techniques.

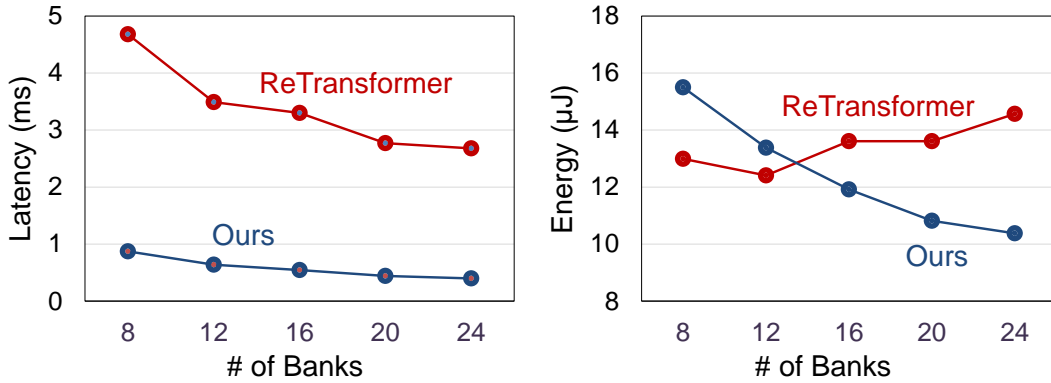


FIGURE 3.14: The latency and energy results with different number of banks for our design and ReTransformer.

### 3.6.3 Scalability Analysis

We investigate the bank-level scalability when we have more available banks to improve the token-level parallelism for both attention map computation stage and output computation stage. The result is shown in Figure 3.14. The processing latency can be reduced about 50% when we increase the bank-level parallelism from 8 to 24, while the latency reduction is only 30% for the baseline design. Moreover, the gap between output design and baseline expands with more banks. The results also show a reduction of energy consumption when we use more banks in the proposed design, while the baseline design consumes more energy. When we increase the token-level parallelism, the token reuse rate can be increased by reusing the K vector for different Q vectors. Thus, we could achieve a lower energy consumption in terms of memory access cost. In contrast, the baseline accelerator consumes more energy in the computation as the inter-bank efficiency is further reduced by mapping the Q, K, V matrices into more banks.

### 3.6.4 Comparison with Non-PIM Accelerators

The energy-latency trade-off of both PIM and non-PIM accelerator is shown in Figure 3.15 (a). Generally, the non-PIM designs could achieve lower latency than ReTransformer, while the NVPIM-based designs could achieve a higher energy efficiency. In con-

trast, our proposed design could achieve a better energy-latency trade-off, where both energy consumption and latency is relatively low. For example, our design could achieve up to  $3.8\times$  and  $8.6\times$  energy reduction comparing with Sanger and DOTA, respectively, while the processing latency is in the same level. However, Retransformer could deliver  $3.71\times$  and  $8.5\times$  energy reduction at the cost of  $4.8\times$  and  $3.3\times$  latency overhead, respectively.

To further understand the energy efficiency improvement, we present an energy consumption breakdown of our design, ReTransformer and non-PIM designs. Figure 3.15 (b) shows the energy breakdown of four designs to process a DeiT-Small model with different sparsity levels. For all sparsity levels, our design consumes less energy on computation. The lower computation energy comes from a higher array utilization rate. However, our design requires higher memory access energy due to smaller data reuse opportunities while using a vector-based processing scheme.

Comparing with Sanger, our design could achieve both computation and memory access energy reduction. Sanger uses a systolic-array-based processing scheme and its mapping efficiency is relatively low for unstructured sparse processing. The computation energy reduction comes from the higher utilization rate in our design. The memory access energy reduction comes from our hybrid stationary dataflow, where the data movement of Q and V is reduced. In contrast, Sanger uses an attention map stationary dataflow, thus, during the computation, all the Q, K, V vectors should be fed to the systolic array multiple times. Comparing with DOTA, our design achieves a similar computation energy with a large reduction of memory access energy. DOTA uses 1-D SIMD-based processing scheme and all the Q, K, V vectors require to be fetched multiple times during the computation. In addition, the limited token parallelism in DOTA reduces the reuse opportunity of K vectors.

### **3.7 Conclusion**

In this work, we propose SparseLattice, which aims at solving the challenge of the sparse attention on NVPIM architecture with a dynamic and unstructured pattern. We first propose a reconfigurable NVPIM bank with vector-based primitives to improve the intra-bank

utilization rate for the SDDMM and SpMM computation. Based on our bank-level innovation, we design a hybrid stationary dataflow which enables a pipelined processing scheme to hide the computation latency of the SDDMM stage and SpMM stage. Furthermore, we propose a greedy scheduling scheme and a task redistribution scheme to accelerate the SDDMM stage and the SpMM stage by improving the inter-bank utilization rate. Our proposed design could achieve up to  $12.36\times$  performance improvement over conventional NVPIM architecture (ReTransformer), while delivering a up to  $3.4\times$  energy efficiency improvement on a range of representative benchmarks. In addition, SparseLattice could also

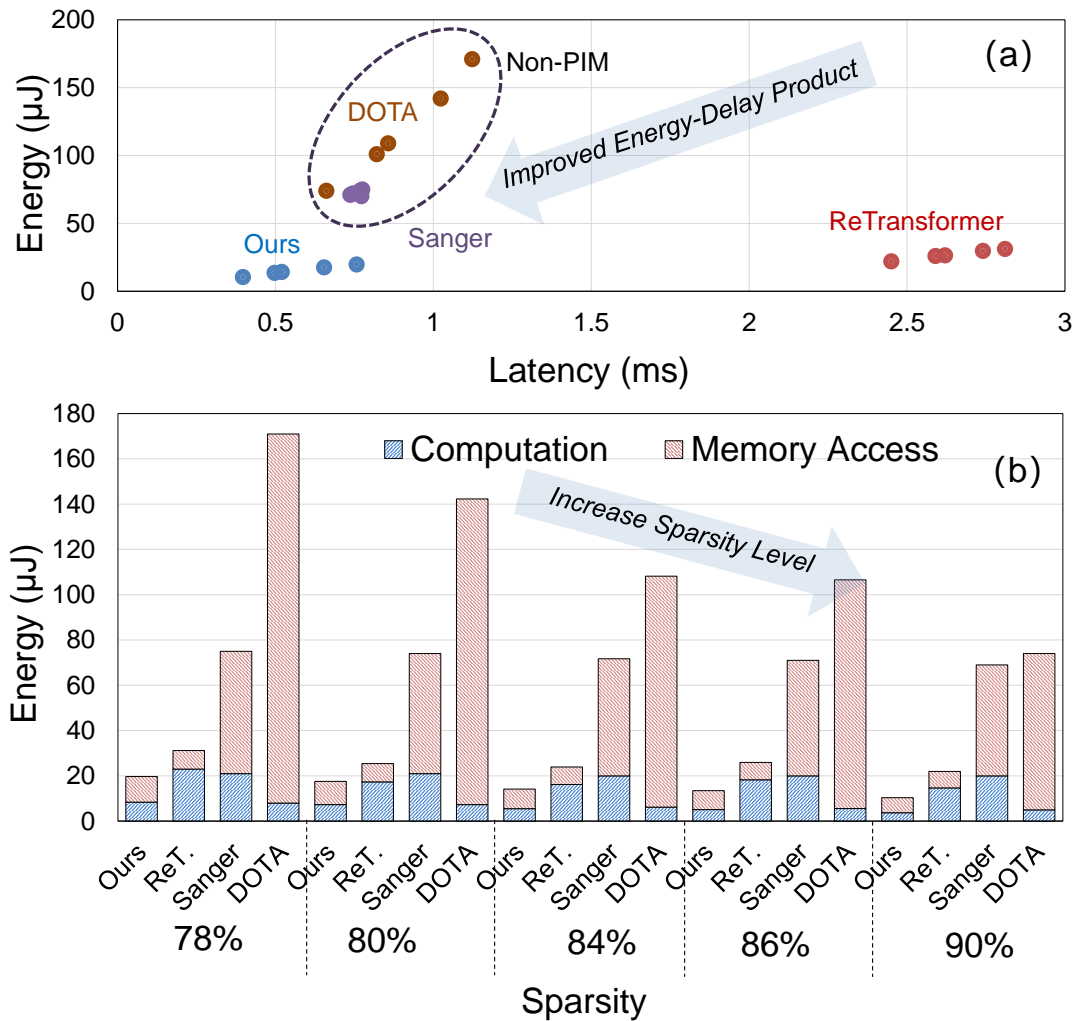


FIGURE 3.15: (a) Energy-latency trade-off (b) energy breakdown for PIM and non-PIM designs with different sparsity level on DeiT-Small model.

achieve up to  $8.6\times$  energy efficiency improvement over non-PIM design without sacrificing the processing latency.

## 4. Hybrid Digital/Analog IMC for lossless computing

In-memory-computing (IMC) technology has attracted a lot of attention in recent years as a solution for low power neural network processing [189, 193, 172, 150, 62, 97, 187, 159]. In particular, static random access memory (SRAM) based analog in-memory computing schemes are showing great potential to improve the energy efficiency by moving the key operation, multiplication and accumulation (MAC) into analog domain [137, 13, 33, 39, 135, 80, 40]. Typically, the IMC macro supports vector-matrix multiplication (VMM) as the computation primitive. The elements of the matrix are stored in the memory array as logical values, and two or more transistors are added to the memory cell to convert the logical values to the analog currents in the SRAM array. Then, each element of the vector is converted into the voltage generated by the wordline driver and applied to the corresponding wordline of the array. The accumulation result can be represented as the accumulated current on the bitlines or other customized accumulation lines. The current is then converted to digital domain by an analog-to-digital converter (ADC) for the subsequent process. Since multiple wordlines are activated in parallel, the normalized throughput and energy efficiency can be extremely improved comparing with conventional digital MAC unit. However, the high efficiency of the analog IMC computing scheme sacrifices the computation accuracy, due to non-ideal effect of the analog computing units and interface ADCs. Consequently, existing analog IMC macros are usually used for highly customized neural networks [177, 188], such as binary, ternary neural networks. The extreme quantization scheme in these networks reduces the accuracy of the workloads, which challenges the utilization of analog IMC in the general purpose neural networks (NN) inference accelerators.

To integrate analog IMC into general purpose NN inference accelerators, one key design metric is to support lossless MAC operations. To achieve this specific requirement, the analog accumulation process should be robust to PVT variations and enough margin should be provided for successive sensing, and an ADC with enough effective number of bits (ENOB) is required to accurately generate the accumulation results. Prior efforts to improve the robustness include investigating the usage of advanced analog computing schemes such as

capacitive coupling [33, 92]. However, to implement the capacitive coupling scheme, one or more capacitors are required to be added to each memory cell, and multiple transistors are required to implement the switches. For example, [92] uses 10T-1C SRAM cell to implement the in-memory charge sharing scheme. The computation accuracy is improved at the expense of memory density. In addition, the advanced analog computing scheme only resolves the robustness issue, while the ENOB requirement of the ADC can not be eliminated. There will still be considerable energy cost of the interface circuits to convert the analog value into digital domain.

An alternative approach to solve the problem is to design a fully-digital IMC macro. The basic digital IMC implementations is based on local AND computation and a in-memory adder trees. The local AND is implemented by a NOR gate with two reversed inputs to improve the area efficiency. A 4T-NOR gate is added to each memory cell, and the output of the NOR gate is connected to a local adder tree, which counts the bits "1" of the 4T-NOR gate and generate a bit-wise accumulation results. The results are sent to the same local PE for successive post processing, which is same as the analog IMC macro. Since all the computation is in the digital domain, the proposed IMC scheme is robust to PVT variations with arbitrary bitwidth. However, this specific scheme has large hardware overhead due to additional digital circuits, where the local adder tree consumes about 10× area compared to the memory cell, so the memory density will be extremely degraded. In summary, to achieve higher energy and area efficiency of analog IMC macros, we need to find a solution that can improve the sensing margin and reduce the ENOB requirements of the ADC at the same time.

In this paper, we propose an alternative solution to the lossless SRAM-based analog IMC macro, instead of focusing on pure circuit level innovation to build up robust computing schemes. The basic idea is to leverage the sparsity in typical NN workloads, where the computation results will mainly be small values. Simulation results show that, when 64 rows are activated in parallel, 90% accumulation results are within 8 instead of following a uniform distribution. Different from the conventional analog IMC design methodology

where all accumulation results in the analog domain are treated equally, we propose an analog computing scheme based on a nonlinear transfer function which only covers an accurate computation for the low MAC value region. Significant energy efficiency improvement can be observed as the ENOB requirement for the interface ADC can be reduced. Our contributions can be summarized as follows:

- We propose a partial sum distribution aware computation scheme with a nonlinear transfer function to improve the sensing margin by  $7\times$  as well as a 3bit ENOB reduction for the interface ADC.
- We further propose a hybrid digital/analog computation scheme with an additional digital data path to provide lossless computation capability with only 13% area overhead.
- We verify our ideas on a fabricated silicon in 65nm technology. The result show that the energy efficiency can be improved by  $2.92\times$  higher than conventional current domain IMC macros.

## **4.1 Background**

### **4.1.1 Analog IMC**

Analog IMC is characterized by the use of specific memory technology to perform MAC operations. It leverages the values stored in memory to directly modulate analog input signals into weighted analog output signals. A typical analog IMC macro contains a wordline driver, one or multiple memory arrays, readout circuitry (including column multiplexers (Col. MUXs) and ADC), and a shift adder for multi-bit accumulation. Depending on the computing scheme, the design methodology of the interface circuits and memory cells may vary slightly. Here, we review two representative approaches to implement analog domain computation.

#### **4.1.1.1 Current Domain**

One representative approach to implement the analog computation is based on 8T memory cell [137]. The logical value is stored in the conventional 6T cells and two additional

transistor is used as a current source. We first present the basic computation scheme where the input and weights are both 1 bit. Before the computation is performed, each RBL is first pre-charged to the supply voltage  $V_{DD}$ . When the computation starts, the input vector is encoded as a binary value and a pulse will be applied on the read wordline (RWL) in the macro. When the logical value stored in the memory array is 1 and the RWL is driven to high voltage for a short period ( $T_0$ ), 2 serial-connected NMOS will form a current source to discharge the RBL (assume the current is  $I_{DS}$ ). The final voltage at RBL depends on the number of activated 2T current-sources which represents the bit-wise multiplication

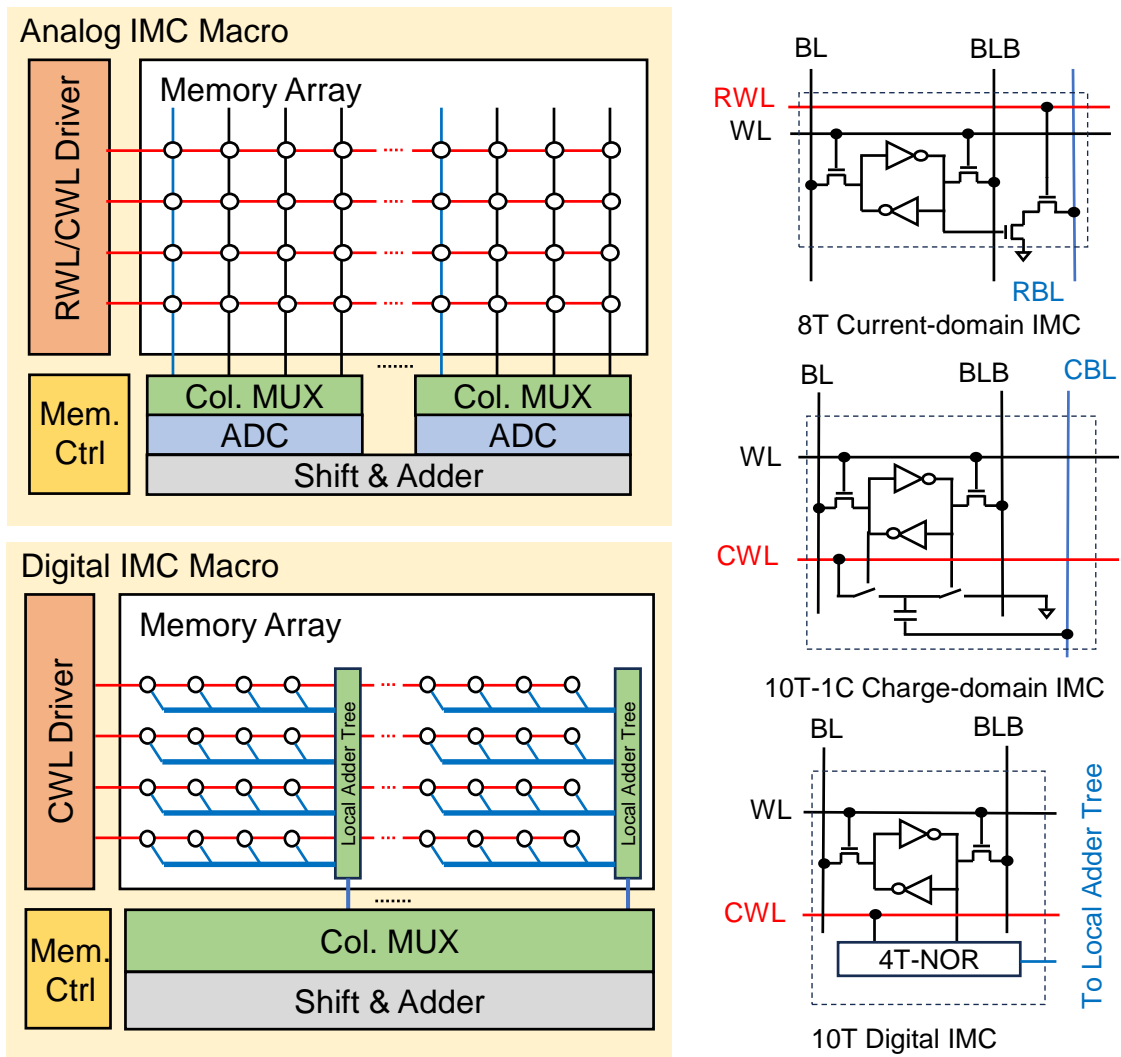


FIGURE 4.1: Illustration of analog IMC macro, digital IMC macro and related cell design.

and accumulation results. The voltage at RBL will then be sampled and held on a column capacitor ( $C_{RBL}$ ), and then converted by ADCs located in each column. The relationship between voltage at RBL and desired output (transfer function between analog output and input vector) can be written as follows:

$$V_{RBL} = V_{DD} - \int_0^{T_0} \frac{\sum W_i X_i * I_{DS} * dt}{C_{RBL}} \quad (4.1)$$

If  $V_{RBL}$  is large enough to keep  $I_{DS}$  as a constant, the equation can be reformulated as follows:

$$V_{RBL} = V_{DD} - \frac{\sum W_i X_i * I_{DS} * T_0}{C_{RBL}}. \quad (4.2)$$

Here  $W_i$  and  $X_i$  represent the logical values of weights and inputs. To deal with the multi-bit operands, the input vector and weight matrices are stored as a 2's complementary value. Consequently, we need to further perform a post accumulation to add the ADC output from different column. A shift-adder is used to add the results at different column for multi-bit accumulation purpose.

$$P_{sum} = \sum \sum Dout_{i,j} 2^{i+j} * (-1)^{i==7|j==7} \quad (4.3)$$

The -1 term is used to implement negative weights or input.

Assume the analog computing results are represented as voltage with a full range of  $V_{Full}$ , the vector length to be accumulated in the analog domain is  $2^N$ , and the bitwidth of the input vector is 1. In this case, the possible analog accumulation results would range from  $0 - 2^N$ , and the sensing margin will be  $V_{Full}/2^N$ . As shown in the transfer function, two terms will significantly generate variations to affect the sensing margin. The first term is the current generated by each cell ( $I_{DS}$ ), which will be affected by threshold voltage variations. The second term is the pulse period  $T_0$ , which will be affected by the pullup/pulldown drivability of the driver in the RWL. In a practical design example,  $V_{Full}$  is about 0.7V and  $N$  is 6. The sensing margin will be around 10mV and an ADC with 6-bit ENOB is required. To achieve this sensing margin, the  $3\sigma$  variation of  $I_{DS}$  and  $T_0$  should be less than

2%, However, simulation results show that the  $3\sigma$   $I_{DS}$  variation can reach 20% under 1.0V voltage supply at 65nm with  $W/L=240n/120n$ . Despite the current domain accumulation scheme is simple, it is generally challenging to implement lossless computation based on current domain accumulation.

#### 4.1.1.2 Charge Domain

Another representative approach is based on capacitive coupling scheme [92]. The memory cell is shown in Figure 4.1. Before the computation is performed, each CBL is first pre-discharged to the ground. When the computation start, the input vector is encoded as a binary value and a DC voltage will be applied on the read wordline (CWL) in the macro. When the logical value stored in the memory array is 1 and the CWL is driven to high voltage, charges  $V_{CWL}/C_i$  will be accumulated on the CBL, while there will be zero charge if CWL is 0 or logical value stored in the memory cell is 0. The final voltage at CBL will depend on the total charge which represents the bit-wise multiplication and accumulation results. The successive conversion and post processing is similar as current domain. The relationship between voltage at CBL and desired output (transfer function analog output and input vector) can be written as follows:

$$V_{CBL} = \frac{\sum Q_i V_{CWL_i} * C_i}{\sum C_i}. \quad (4.4)$$

Here, the major term that affects the computing accuracy is the capacitance of  $C_i$ , and  $V_{CWL}$  can be designed with an accurate voltage reference. As pointed out in [33], the capacitance variance can be reduced to 1% with a 2fF customized metal-oxide-metal capacitor. The charge domain design provides some potential to implement lossless analog domain IMC. However, the area and energy efficiency is relatively lower than current domain as multiple capacitors are added to each CWL to ignore the parasitic capacitance, and the complexity of the cell structure is large due to the presence of switches.

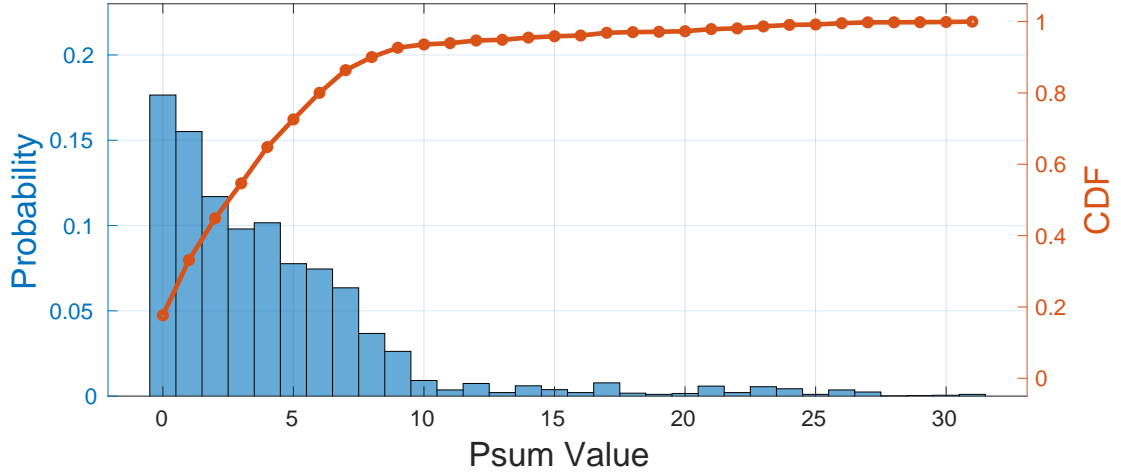


FIGURE 4.2: Psum value distribution of Resnet-20 on CIFAR-10 dataset.

### 4.1.2 Digital IMC

To overcome the ADC overhead and inaccurate computation of analog IMCs, researchers also presented multiple digital implementation of the IMC macro [36, 172]. In the literature, one representative approach of digital IMC implementations is based on local adder trees. The local AND operation is implemented by a NOR gate with two reversed inputs to improve the area efficiency. As shown in Figure 4.1, in the digital IMC, a 4T-NOR gate is added to each memory cell, and the output of the NOR gate is connected to a local adder tree. Each local adder tree counts the bits "1" of the 4T-NOR gate and generate a bit-wise accumulation results. The results are sent to the same local PE for successive post processing, which is same as the analog IMC macro. Since all the computation is in the digital domain, the proposed IMC scheme is robust to PVT variations with arbitrary bitwidth. However, this scheme has one main drawback. The local adder tree consumes about  $10\times$  area overhead compared to the memory cell, so the memory density will be reduced extremely.

## **4.2 Design Methodology**

### **4.2.1 Partial Sum Aware Accumulation**

In a typical neural network model, the activations and weights are naturally sparse. Thus, the bit-wise accumulation results are usually small due to the sparsity. To understand the partial sum distribution, we choose ResNet-20 model on CIFAR-10 dataset, and calculate the partial sum value distribution under an assumption that 64 rows are activated in parallel. As shown in Figure 4.2, over 90% MAC results at each bitline are within 8 when 64 rows are activated in parallel. Based on this observation, we are motivated to design two different data paths for the low MAC value results (less than a specific threshold value) and high MAC value results (higher than a specific threshold value). Here, we choose the Low/High threshold as 8 to achieve a better trade-off between efficiency and implementation cost. For the low MAC value case, we can directly use the conventional analog IMC design method to implement the MAC operation in current domain. Since the low MAC value is less than 8, we only need to differentiate 8 possible states instead of 64 states in conventional analog IMC design. The sensing margin can be increased to 100mV when the full range is 0.7V, and the ENOB requirement of the ADC will be reduced to 3 bits. For the high MAC value case, we can design an additional data path based on digital IMC. Since only less than 10% MAC value is high, we can minimize the area overhead by sharing the digital path among a number of columns.

### **4.2.2 Macro Overview**

Figure 4.3 shows the overall architecture of the proposed macro. Our macro is based on conventional analog current domain IMC design, with several moderate modifications. The proposed macro contains a wordline decoder and drivers, read/write interface (R/W Interface), several multi-bit sense amplifiers (MBSAs), one RWL driver and one or multiple memory array(s). Beyond the standard macro components, our macro has three key modifications to support the hybrid analog-digital computing mode. Firstly, we add an additional digital data path for high MAC value computing, motivated by the digital IMC

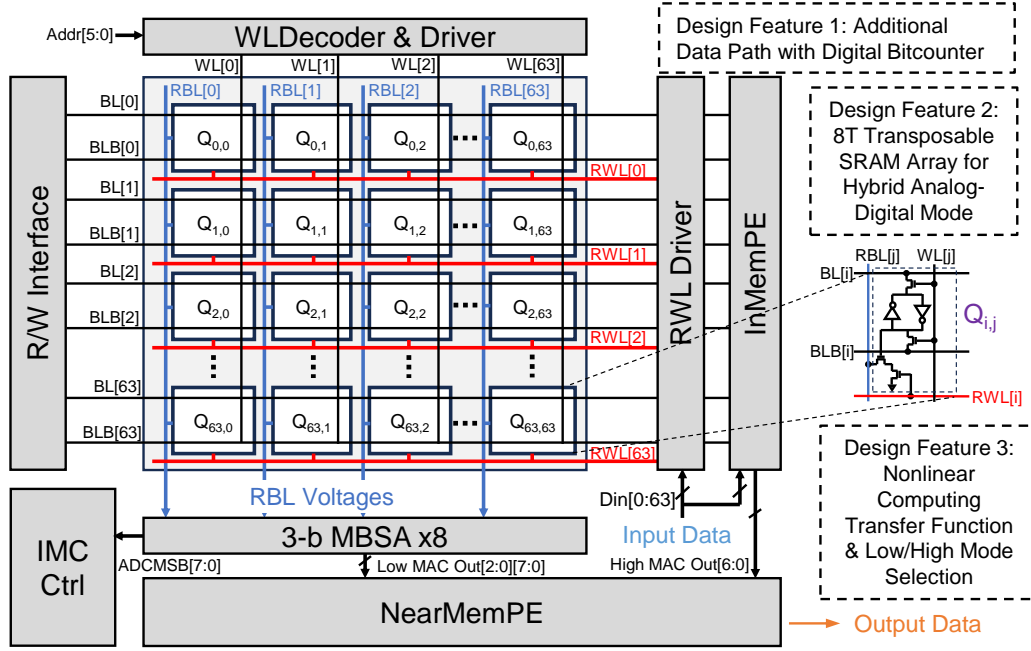


FIGURE 4.3: The overall architecture of our proposed macro.

computing scheme [191], to provide a simple and robust digital accumulation for high MAC value. The InMemPE is composed of several AND gates, a digital bitcounter, which is used to perform a lossless bit-wise MAC no matter what the MAC value is. One input for the InMemPE is the input vector and the other input is directly connected to BL/BLB. Secondly, the memory cell in the proposed design is implemented as a transposed 8T cell, where the RWL(RBL) is vertical to the standard WL(BL/BLB). This transposing design enables two separated data paths for both analog and digital computing. Thirdly, the IMC controller (IMC Ctrl) circuit is designed with additional functionalities to switch the analog-digital computing mode. The ADCMSB signal connected to the IMC Ctrl is used to indicate whether a specific MBSA detects a high MAC value. The NearMemPE is still a shift-adder to receive the low MAC results from the MBSAs, with an additional data path to receive the high MAC output from the InMemPE.

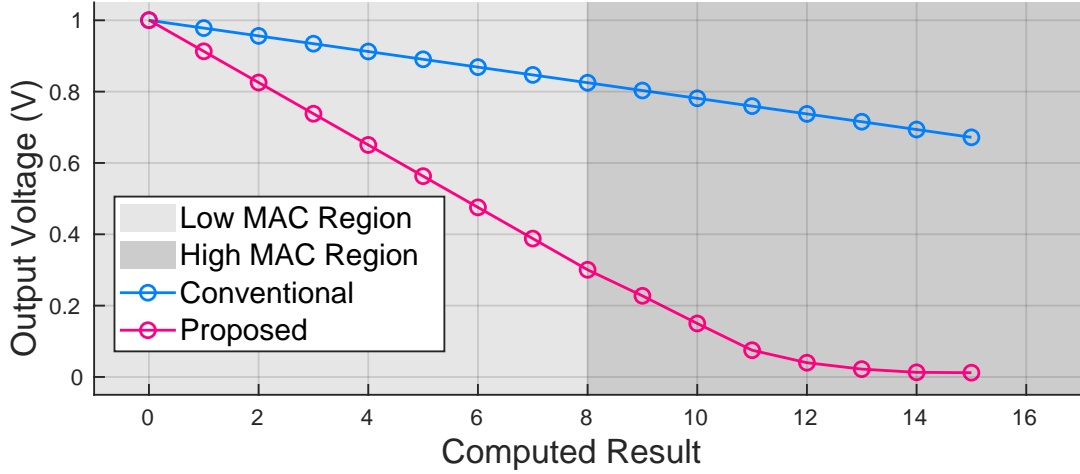


FIGURE 4.4: Simulated transfer function of proposed macro and conventional macro.

### 4.2.3 Nonlinear Transfer Function

Instead of forcing the desired output and analog voltages at RBL to be linear, we make the transfer function into two regions, as shown in the following equations:

$$V_{RBL} = \begin{cases} V_{DD} - \frac{\sum W_i X_i * I_{DS} * T_0}{C_{RBL}}, & \sum W_i X_i < N \\ 0, & \sum W_i X_i \geq N \end{cases} \quad (4.5)$$

Here,  $N$  is a pre-defined threshold value to differentiate low and high MAC regions. For the low MAC region, we keep the original linear transfer function, where the desired computing results are linear to the voltage generated at RBL. For the high MAC region, we make the output voltage to a near zero value. Figure 4.4 shows the simulated transfer function of our proposed IMC macro and conventional IMC macro. Since we only need to convert the MAC results in the low MAC region accurately, the sensing margin can be improved by up to  $7 \times$  comparing with conventional design with linear transfer function.

It is worth noting that the nonlinear transfer function of the analog computation scheme is naturally supported. The only modification is to reduce the value of  $C_{RBL}$  or increase the value of  $I_{DS}$  and  $T_0$ . In this case, the  $I_{DS}$  can be designed as a constant only less than  $N$  current sources are activated, denote to the low MAC value region. Otherwise, the  $I_{DS}$

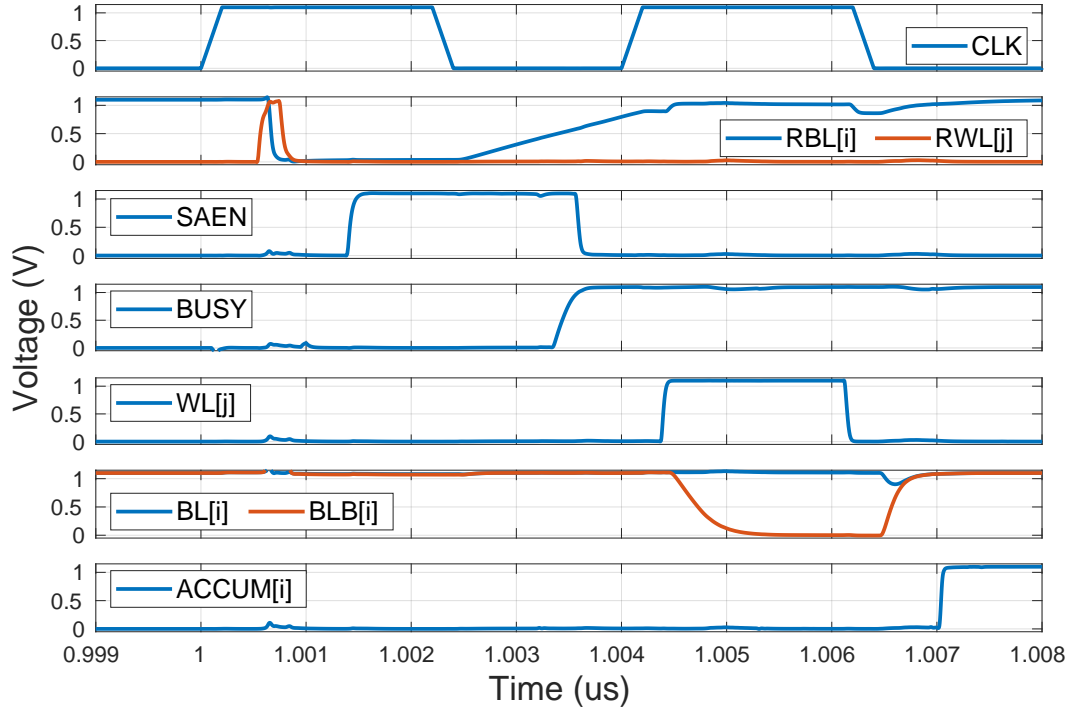


FIGURE 4.5: Simulated timing diagram of analog mode and digital mode computation.

will naturally goes to near zero as the current source turns to the linear region.

#### 4.2.4 MAC in Analog Domain

Before the computing starts, the RBL is pre-charged to  $V_{DD}$ . A input bit vector is sent to the RWL driver, and the trigger clock will generate a short pulse on each RWL if the corresponding input bit is 1. The current at each RBL will discharge the capacitor  $C_{RBL}$  to a certain voltage level, following the nonlinear transfer function we implemented. After the voltage at  $C_{RBL}$  is stable, a positive edge of the sense-enabling signal (SAEN) will trigger the interface circuits to sense the  $V_{RBL}$ . we use a MBSA with 7 different voltage reference to differentiate 8 different voltage level generated by the analog current summation. An additional comparator with another voltage reference adjusted to differentiate whether the output voltage crosses the ranges of low MAC value. If the output is within the low MAC

region, the converted digital output will be directly sent to the NearMemPE to perform the bit-wise accumulation. Otherwise, the column index will be sent to the IMC Ctrl for successive MAC in digital domain.

The first cycle in Figure 4.5 shows a simulated timing diagram of proposed analog computing scheme when the computation results fall into a high MAC value region. After SAEN is triggered, the digital components (BUSY) will be activated due to a high MAC value is detected. In this case, the computation of the next cycle will go to digital domain for high MAC value. Otherwise, the next cycle will be used for analog computation for the other columns or input bit vector, and there will be no throughput loss.

### **4.2.5 MAC in Digital Domain**

When a high MAC value is detected, the IMC Ctrl circuit will store the column index of the high MAC value column and switch the macro into the digital mode. The second cycle in Figure 4.5 illustrates the timing diagram of the digital computing scheme. A specific address will be generated in the IMC Ctrl to activate the wordline driver based on the column index detected to compute a high MAC value. Once the wordline is activated, the weight data stored in one column of the memory array will be read out through BL/BLB discharging process. The input vector is directly sent to the InMemPE to the weight data at BL/BLB will be sent to the InMemPE to perform a local digital computing with the input vector. At the same time, the NearMemPE is configured to receive the 7-bit output from the InMemPE. IMC Ctrl also generates the shift value for this accumulation depending on the column index.

## **4.3 Experiments**

### **4.3.1 Circuit-Level Measurement Results**

To evaluate the proposed methodology, we fabricated the SRAM IMC macro in GP 65nm technology node. The die photo and the detailed layout of the macro are shown in Figure 4.6. The proposed macro occupies  $0.025 \text{ mm}^2$ , with a  $2.4 \times 0.9 \text{ um}^2$  8T SRAM bitcell. Most of the area is the memory array (51%), and the digital/analog interface only occupies

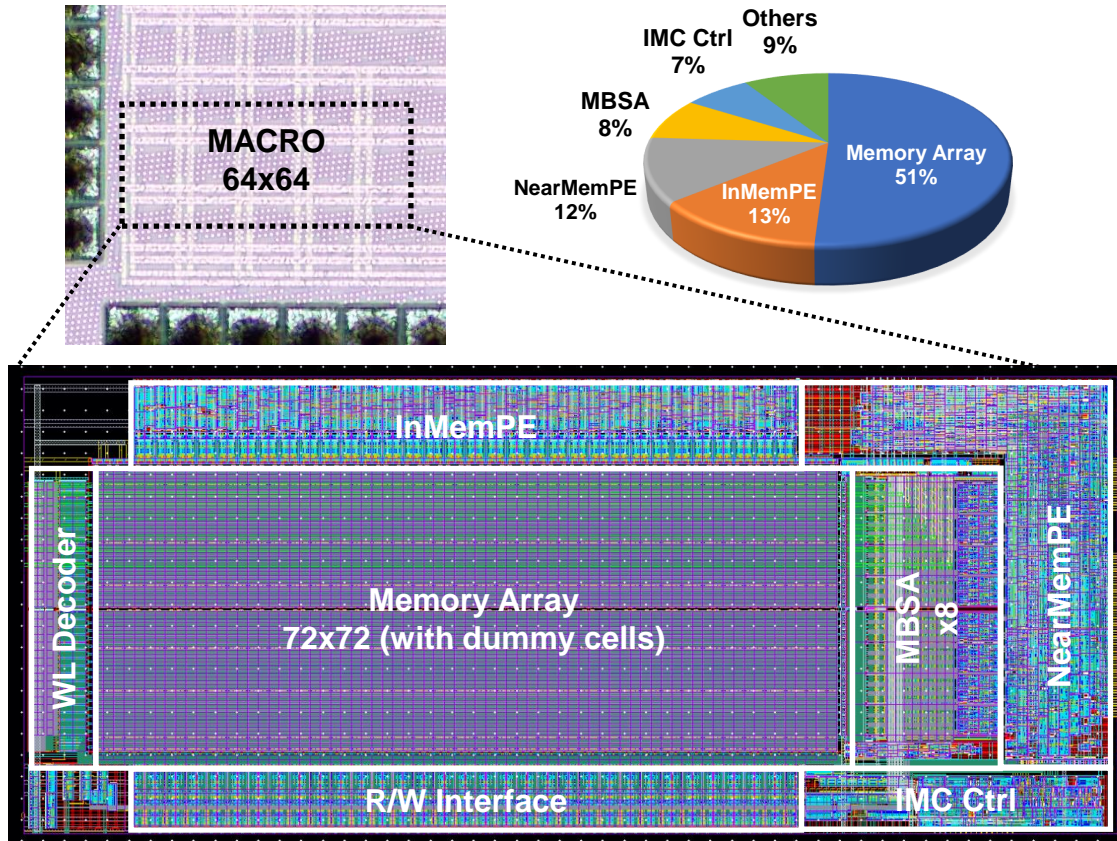


FIGURE 4.6: The die photograph, layout of our proposed macro, and the area breakdown results.

$0.002 \text{ mm}^2$ , and it takes about 8% of the whole macro, since we reduced the ENOB to only 3bit. Both of the InMemPE and the NearMemPE occupy 13% of the whole macro.

#### 4.3.1.1 Energy Consumption

Figure 4.7 depicts the measured energy consumption for both digital mode and analog mode. The average energy consumption of the digital mode is higher than the analog mode. When the supply voltage is 1.0V, the digital mode requires about 450fJ to execute an 8-bit MAC, where it will be reduced to 154 fJ when the supply voltage is 0.7V. The effective energy efficiency is 2.22-6.89 TOPS/W for the digital mode computation. In contrast, the analog mode can provide significant energy reduction comparing with the digital mode. The energy consumption is 101 and 39.2 fJ when the supply voltage is 1.0 and 0.7 V,

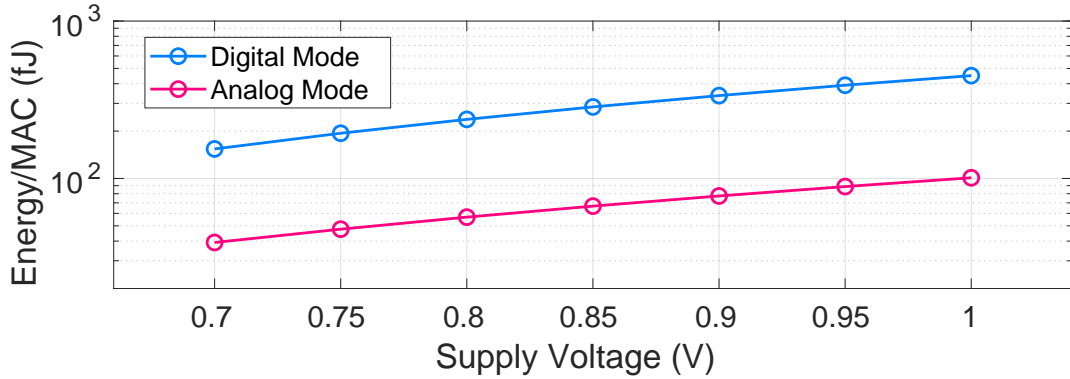


FIGURE 4.7: Measured energy consumption for digital mode and analog mode.

Table 4.1: Comparison with State-of-the-art IMC Design

Work	This work	JSSC19 [137]	JSSC21 [136]	ISSCC22 [172]	JSSC19 [13]
Technology Node	65	55	28	28	65
Speed (ns)	4	10.2	8.4	3.3	150
Area Efficiency ( $GOPS/mm^2$ )	160	572	120	200	126
8b Lossless	Yes	No	Yes	Yes	No
Energy Efficiency (TOPS/W)	25.5 (8b-8b)	18.37 (4b-5b)	16.63 (8b-8b)	27.4 (8b-8b)	28.1 (8b-1b)
Cell Structure	8T	T8T	6T+LCC	6T+LCC	10T-1C
Computing Scheme	hybrid current-digital	current	current	digital	charge

respectively. In this case, the effective energy efficiency in this mode is around 9.9-25.5 TOPS/W.

#### 4.3.1.2 Speed

Figure 4.8 presents the measured shmoo plot for the proposed macro. The digital mode computation could achieve up to 2.3ns access time without any computation loss under 1.0V voltage supply. Even with the supply voltage at 0.6V, the access time could still achieve 4.9 ns to provide an accurate computing results. For the analog mode, the macro can reach 3.7ns under 1.0V supply voltage and 6ns under 0.7V supply voltage.

#### 4.3.1.3 Comparative Results

Table 4.1 shows the comparative results with state-of-the-art IMC design. We first comparing our design with similar technology node. [137] is a conventional current domain

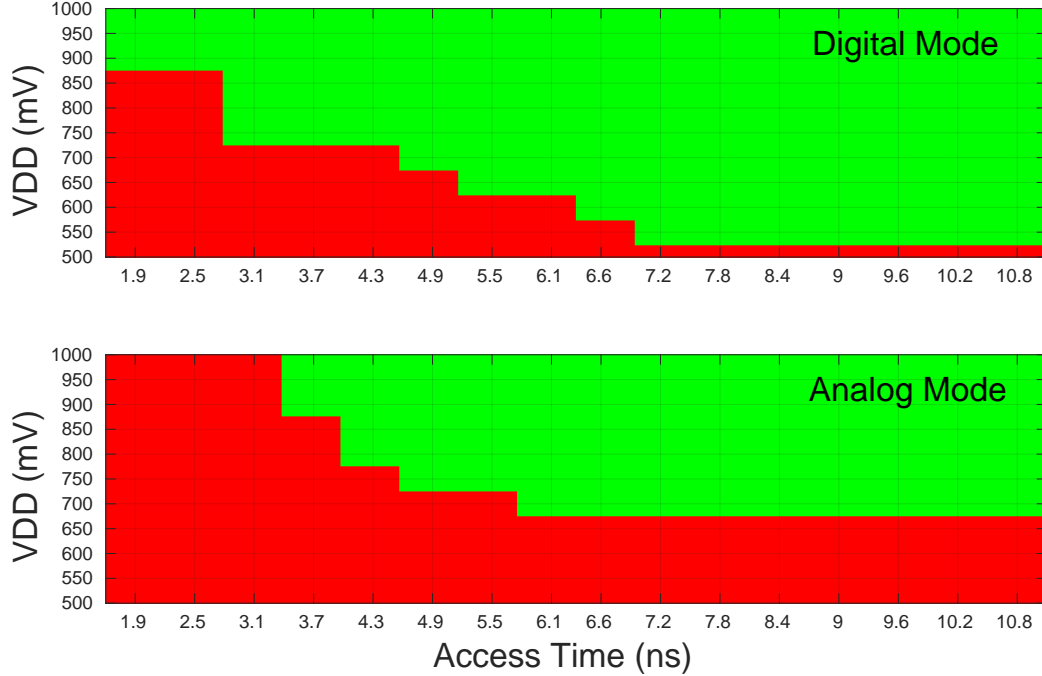


FIGURE 4.8: Measured Shmoo plot for the proposed macro for the digital mode and analog mode.

design with a linear transfer function. Our design can achieve higher energy efficiency because our design does not require an ADC with large ENOB. The design in [137] shows higher area efficiency, which sacrifices the capability to provide a 8b lossless computation. Comparing with charge domain design in [13], our design also achieves higher energy efficiency and higher area efficiency due to the current-domain computation scheme.

Other than comparing with similar technology node, we also compare our design with some designs implemented in advanced nodes, [136] and [172]. [136] is an extended version of [137] and their computing scheme is similar. The After normalized to 65nm technology node, our design can achieve  $2.65 \times$  higher energy efficiency and  $5.33 \times$  area area efficiency than this scheme. The current domain implementation has a low energy and area efficiency as a result of an interface ADC with a 6bit ENOB to support lossless computation. [172] is one representative work of digital IMC. All the analog IMC macro can achieve higher energy efficiency and memory density, as the local adder tree extremely increases the area overhead. In particular, our design could reach around  $2 \times$  energy efficiency improvement

over the fully digital design.

### 4.3.2 Architecture-Level Simulation Results

To understand the performance/energy consumption of the proposed macro in real neural networks accelerator, we further perform several architecture-level evaluation based on our circuit-level evaluation results. The computation flow and weight mapping scheme follows the principle shown in [189] and [193] for convolution neural networks and mobile convolution neural networks. We use an event-driven simulation framework in [190] to generate the area normalized throughput (denote to area efficiency) and energy efficiency for running 6 different models trained on CIFAR-10 and ImageNet datasets, based on the circuit-level evaluation results. To make a fair comparison, we a fully analog IMC macro in current domain in [136], and a fully digital IMC macro in [172] to build the same architecture, and scale our area efficiency and energy efficiency results into the same technology node.

The architecture-level simulation results are shown in Figure 4.9. The architecture based on our proposed IMC macro could achieve  $2.52\times$  to  $3.84\times$  improvement of the area efficiency comparing with the architecture based on a fully analog IMC macro. In terms of the energy efficiency, the architecture based on our proposed IMC macro could achieve  $2.14\times$  to  $2.92\times$  improvement comparing with the architecture based on a fully analog IMC macro.

## 4.4 Conclusion

In this paper, we propose a new design methodology to implement lossless 8b MAC operation with a hybrid digital/analog computing scheme. Considering the sparsity in machine learning workloads, we first propose a nonlinear transfer function of the analog accumulation to reduce the ENOB and improve the sensing margin of the conventional current domain IMC scheme for low MAC values. To deal with high MAC values, we further propose a in memory digital computing scheme to compensate the error generated by analog computing. Chip measurement results show that the proposed macro could

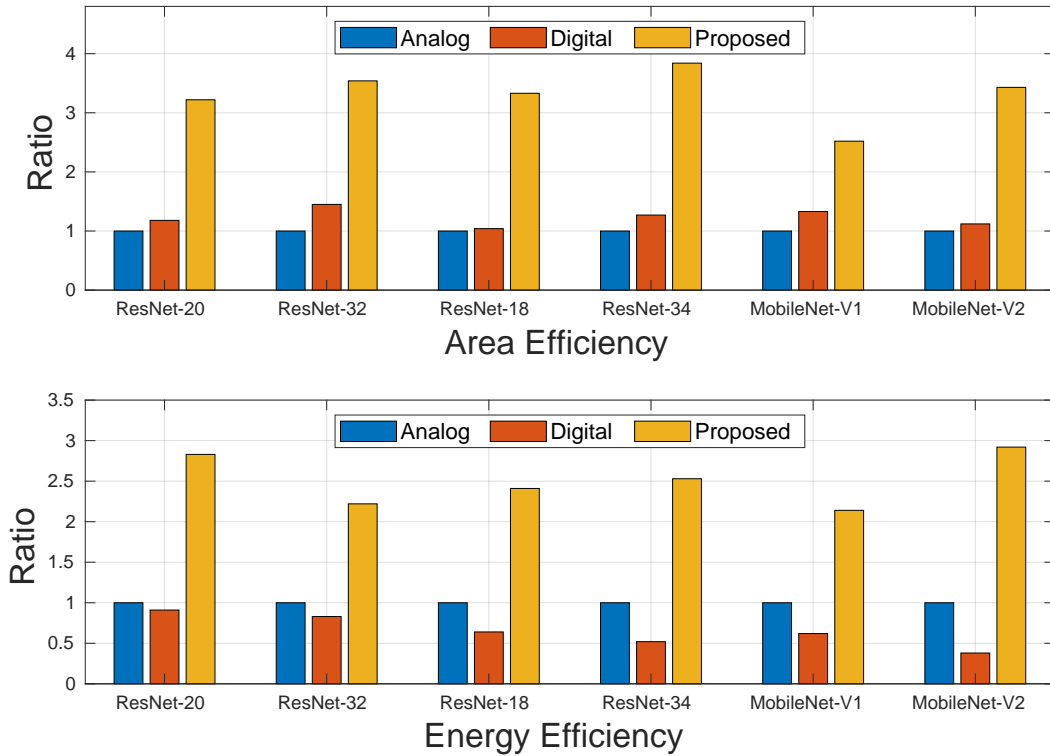


FIGURE 4.9: Architecture simulation results for area efficiency and energy efficiency on various machine learning model.

achieve  $160 \text{ GOPS}/\text{mm}^2$  area efficiency and  $25.5 \text{ TOPS}/\text{W}$  for  $8\text{b}/8\text{b}$  matrix computation.

The architectural-level evaluation for real workloads shows that the proposed macro can achieve up to  $2.92\times$  higher energy efficiency than conventional analog IMC designs.

## 5. Simulation Framework

For large-scale PIM-based accelerators, massive factors may affect the performance, energy consumption and chip area. In order to generate an optimized design, it is critical to perform an accurate early-stage end-to-end simulation. Harmonica [106] is a framework to simulate small-scale memristor-based PIM engines for very simple neuromorphic algorithms. NeuroSIM [24] and RxNN [75] focus on circuit-level simulation and provide technologies for sub-array modeling. These three frameworks are accurate for macro-level simulation by configuring the detailed physical parameters of transistors and memory devices. However, they lack some architectural level design description and they cannot accurately model the memory access and data communication patterns. MNSIM [163] and DNN+NeuroSIM [121] are behaviour-level frameworks to simulate large-scale PIM based DNN accelerators. In these two frameworks, accelerator architecture is pre-defined with a dedicated performance model. Both frameworks cannot provide a flexible architecture description and a precise architecture model. Thus, if the users want to change the architecture, they should rewrite the source code and build up their own performance model.

In this paper, we present PIMulator-NN, an end-to-end framework to evaluate a PIM-based accelerator for DNN applications. PIMulator-NN uses an event-driven simulation mechanism, and extracts the common features of accelerator modules as templates. Thus, it enables users to define the inter-module connections and customized execution flow of an architecture. PIMulator-NN also provides a front-end to convert the DNN models into software control flow to enable software/hardware co-simulation. In addition, PIMulator-NN integrates some mainstream circuit-level simulators (e.g., NeuroSim [24], CACTI [114], etc.) to generate more accurate timing, energy and area results. Compared to existing frameworks, PIMulator-NN is the first framework which enables users to flexibly define the architectural design details, such as module parameters, pipeline schemes, and parallelism schemes, to cover most state-of-the-art PIM architectures. The contributions of PIMulator-NN are summarized as follows:

- To the best of our knowledge, PIMulator-NN is the first event-driven framework to

allow users to perform cross-level simulation for PIM-based DNN accelerators.

- By modeling some existing PIM designs, we find that some architectural optimization is sensitive to interconnection configuration. These details are hard to be captured by prior performance model based estimations. PIMulator-NN’s event-driven simulation mechanism is able to generate important trace results such as throughput traces and power traces.
- As most mainstream simulators face the challenge to model the PIM-based accelerators flexibly, we provide an architecture template based event-driven simulation mechanism to model the design details of PIM-based accelerators. This feature would enable users to simulate their own architecture quickly.

It is worth noticing that researchers from different areas may benefit from PIMulator-NN. First, PIMulator-NN provides software/hardware interfaces to allow algorithm researchers to investigate the performance of their DNN models on PIM-based accelerators. Second, architecture researchers can use PIMulator-NN to build up their customized PIM-based accelerators and evaluate the performance. Third, for circuit designers, PIMulator-NN is suitable for them to benchmark high-level performance results by customizing the back-end circuit simulators.

## **5.1 PIMulator-NN**

### **5.1.1 Framework Overview**

Figure 5.1 shows the overview of PIMulator-NN. The framework is composed of (1) a front-end to parse the user-defined DNN model and generate the software configurations, (2) an architecture description file to define the modules, interconnection and dataflow, (3) an Event-Driven Simulation Core (EDSC) to perform event-driven simulation and (4) a back-end to provide detailed estimations on latency, energy, and area.

Before the simulation, the user should provide back-end configurations, DNN models and their architectures. Back-end configurations include several physical parameters to set up the back-end simulation tool. The key components that users should provide to

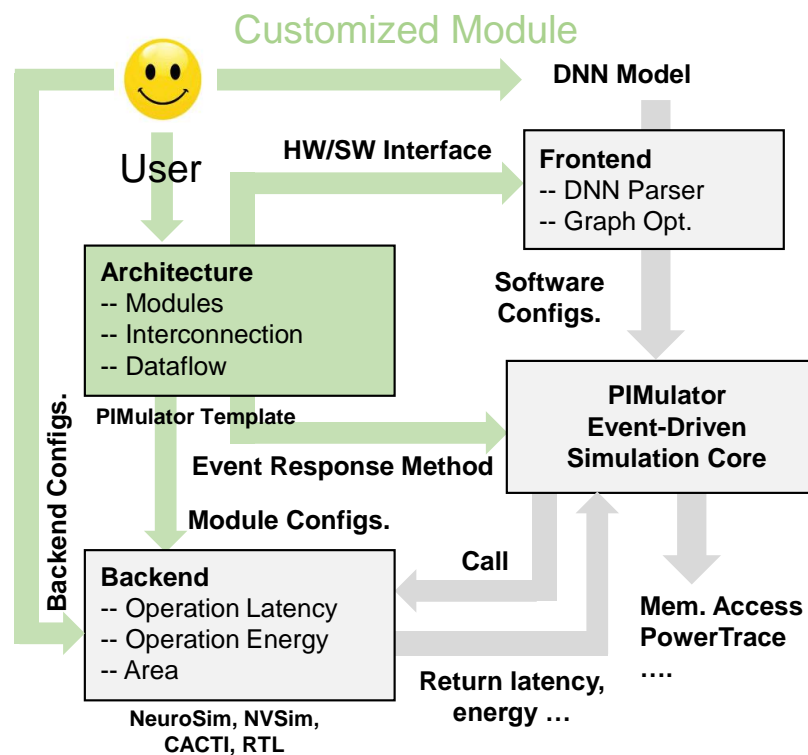


FIGURE 5.1: Overview of PIMulator-NN framework.

PIMulator-NN are the architecture description files. The details of architecture descriptions are presented in Section 5.1.3. Then the user can import their DNN models and feed the models into front-end. After that, a start event is generated, and the simulation starts.

During the simulation, the EDSC would generate a set of new events by analyzing the current event being processed. The event response method is predefined in the architecture description files. This event response method determines the dataflow and execution schedule of the accelerator. To generate accurate latency and energy results, the EDSC would call the back-end simulator for reference. Once an event that the user are interested in has been processed, or all events have been processed, the simulation terminates and the latency/energy results would be recorded.



action is *START*, the proc defined in the event is called for execution. When the action is *STOP*, the proc is forced to stop by setting the state of proc to *IDLE*. By default, other actions are not required by EDSC. Users can extend *CheckAction* function by adding responses to *RECEIVED*, *FINISH* or other user defined actions.

#### **5.1.2.2 CheckProcState**

Once an action is *START*, the next step is to check the state of the proc. EDSC can further process the event when the state of proc is *IDLE*, and the state of the proc changes to *BUSY*. Otherwise, the event would be rescheduled to next timestamp and processed later. This “*IDLE-BUSY*” mechanism is used to model the conflicts (e.g., bus conflicts, port conflicts during memory access, etc.) in the hardware system.

#### **5.1.2.3 AddReceived**

An event with *RECEIVED* action is added to the global event queue. Then the caller is informed that this event has been received. The goal of this “*START-RECEIVED*” mechanism is to model the communication between procs (e.g., hands-shaking protocols).

#### **5.1.2.4 Execution**

The execution function of one proc is the the key part for users to define an architecture. Execution function is in charge of performing operations of the module, such as, vector-matrix multiplication of a sub-array and data copy of a memory. In addition, the data flow management is also defined in the execution function by adding some new events to the global event queue. To generate accurate energy and latency results, back-end simulator is called. The argument of the operation (e.g. length for data access) is passed to back-end simulator, then the energy and latency results are calculated and passed back to execution function. It is essential for PIM to model the non-ideal effects in analog circuits. In this case, the input data can also be passed to back-end simulator as an argument, and the non-ideal effects would be added in the circuit-level evaluation in the back-end simulator. For customized modules without back-end simulator, users can define virtual results of energy

and latency to support EDSC. To model architecture dataflow, users could also define some event generation methods to call other procs according to the current operation and state of the hardware. All these events are added to the global event queue, and will be processed in later simulation steps.

### 5.1.2.5 AddStop

Once the execution is done, EDSC would automatically generate a new event to set the state of proc to IDLE. The action of this event is STOP, and the time is obtained by adding current timestamp and the latency generated from back-end simulator. This “START-STOP” mechanism could also be used to evaluate the module utilization during the event-driven simulation.

### 5.1.3 Architecture Description

In this subsection, we would like to introduce the architecture description methodology to support arbitrary PIM architecture in PIMulator-NN. The basic component of PIMulator-NN is a module (refer to “SimObj” in the source code). Module is composed of a set of sub-modules, the interconnection definition among submodules, a set of ports and several processes. Different from the modeling philosophy in MNSIM [163], the intercon-

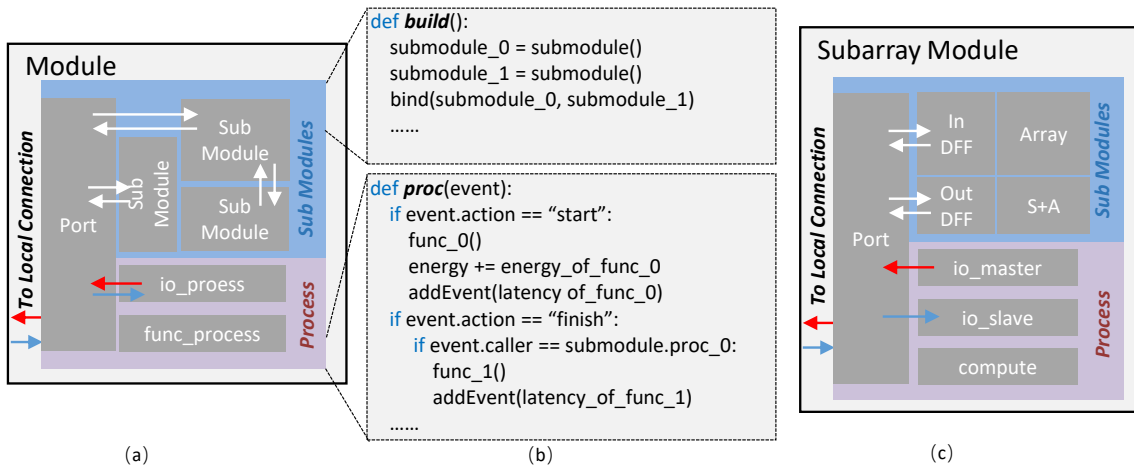


FIGURE 5.3: Illustration of architecture description in PIMulator-NN. (a) General module description. (b) Code organization. (c) Subarray module description.

nection definition in PIMulator-NN is much more flexible, to support arbitrary user-defined interconnection schemes.

To model the architecture details like dataflow and pipeline scheme, we would like to introduce a concept called process in PIMulator-NN. Process is used for event-driven simulation, and one module has at least one process to model its function. In one module, one process is independent to the others. Thus, several processes can run in parallel. The content of one process is the execution function mentioned in Section 5.1.2, and a wrapper to support EDSC simulation. According to the execution function, one process would perform a certain function, and generate a set of new events to call other processes in the same module or other modules. The details of the architecture description are described as follows:

#### 5.1.3.1 Port

Port class is used to model the inter-module data communication and interconnection. Port is automatically generated while binding two modules. Users can also define ports in the module definition, and assign memory space to this port.

#### 5.1.3.2 Sub-modules

The building of sub-modules are defined in **buildHW** function. As shown in Figure 5.3 (b), users can define arbitrary modules by instantiating sub-module class. After instantiation, the user should bind the sub-modules to form the interconnection.

#### 5.1.3.3 Process

The content of one process that needs user definitions is the execution function mentioned in Section. 5.1.2. The other functions (e.g., **checkAction**) are wrapped and packaged in EDSC. As shown in Figure 5.3 (a), each module has two types of processes. `Io_process` is used to model inter-module data communication behaviours, and `func_process` is employed to model inner-module functional behaviours or data communication behaviours. Figure 5.3 (b) shows an example to model an architecture. Users are expected to define the

execution flow of START action. In this example, once the example proc is called, func\_0 is executed, and related energy/latency are calculated. If there are some sub-modules in the module, an event with START action can be generated to call the processes of sub-modules. In this case, users are expected to define the execution flow of FINISH action. FINISH action is called from sub-modules to inform the top module that the sub-module process just called has finished. In the example, once the submodule.proc\_0 is done, func\_1 would be executed.

#### 5.1.4 Example: Computational Sub-array

Figure 5.3 (c) illustrates the architectural description of a computational sub-array. By default, each sub-array is composed of an input buffer (InDFF), an output buffer (OutDFF), and a memory array with corresponding port to perform data interaction outside the module. The data of input buffer and output buffer can be read out or written in through local connections.

The sub-array can be configured into either a master module or a slave module. The data in InDFF and OutDFF can be directly accessed through local connections by calling the **ioslave** process. While serving as a master module, a sub-array can raise a load/save request to read/write data through local connections.

The computation behavior is defined in compute process. Before the computation starts, the input vector should be loaded into the InDFF by raising an **iomaster** process. When the computation process starts, DAC converts the digital values in InDFF into voltages applied to each word-line. After summing up the current of each bit-line, ADC converts the analog currents into digital values, and saves the results in the OutDFF. At the same time, the computation process finishes. The latency and energy results can be generated from the back-end simulator. In current version, the sub-array model does not support partially-activated design. All word-lines should be activated in parallel, and all the weights stored in the array need to participate in computation. This follows the assumptions in NeuroSIM, which serves as the back-end simulator in current PIMulator-NN.

In addition, PIMulator-NN only supports matrix-matrix multiplication function of a sub-array. The modeling of on-chip weight updating is not considered in current version. We assume that weights are pre-mapped into the memory arrays before running simulation, and we avoid modeling the read/write behaviours of memory array. Thus, sub-arrays only serve as computation modules. The **ioslave** process can not perform data access inside the array as a standard memory module.

### 5.1.5 Discussion: Accuracy-Related Simulation

An advantage of PIMulator-NN is the fast event-driven simulation. Instead of simulating the hardware behaviour cycle by cycle as previous works, it can quickly produce energy, latency, and area results. Besides these metrics, PIMulator-NN also supports accuracy-related simulation, e.g., it can evaluate the inference accuracy when considering non-ideal effects of devices. Such simulation requires sacrifice of simulation speed, because the concrete calculation should be conducted. We introduce how to do accuracy-related simulation with PIMulator-NN as follows.

Recall that each module has functions called **proc** to simulate its behaviours. Usually, it only calculates the latency and energy to conduct this behaviour. However, if the user wants to simulate accuracy, **proc** can also simulate the concrete calculation, instead of only producing latency and energy results. The inputs and outputs for the simulation can be transferred using the events in PIMulator-NN. Recall that each event has a parameter **proc**, which represents the **proc** of a module this event should call. The event also contains parameters as the arguments of the **proc**, which can include the inputs and outputs of the **proc**. When executing the **proc** to simulate real calculations, any backend can be used to simulate the non-ideal effects.

### 5.1.6 Limitations of PIMulator-NN

PIMulator-NN could provide a detailed architecture description and a flexible interface for SW/HW co-design. However, there are still several limitations that should be addressed in the future version, as follows.

### **5.1.6.1 Compiler**

PIMulator-NN does not support automatic mapping of a DNN model to an arbitrary architecture. The front-end is only available for users who are using our architecture templates integrated in PIMulator-NN. For a customized architecture, the user should define the mapping function and control flow manually to perform an end-to-end simulation.

### **5.1.6.2 Modeling Control Logic**

For power and area evaluation, the overhead of control logic could not be modeled at current version. As user can define an arbitrary architecture, it is hard to calculate the accurate results of control logic without RTL implementation. Thus, all control logics are virtually implemented, and PIMulator-NN evaluates the functional modules and interconnection only.

### **5.1.6.3 Modeling of Interconnect Scheme**

For current version, we only support bus modeling of the PIM based accelerator. Due to the difficulties of NoC modeling, we did not cover this part of simulation. The user should customize the NoC function based on PIMulator template for interconnection if they would like to model the impact of NoC for large scale PIM Accelerator simulation.

## **5.2 Case Study**

In this section, we present several examples to demonstrate the architectural details of a PIM based NN Accelerator design. We first present a simple Multi-Layer Perceptron (MLP) to demonstrate the interconnection schemes and data flow schemes. These examples show the usage of PIMulator-NN as an architecture/circuit co-simulation platform.

### **5.2.1 Small-Scale DNN Accelerators**

In this section, we present several simple PIM-based designs to illustrate the flexibility of PIMulator-NN. Figure 5.4 illustrates three design examples of an accelerator for simple DNN Model. In this example, we choose a Multi-Layer Perceptron (MLP) with one hidden layer and one output layer. To simplify the case, we assume the weights of each layer

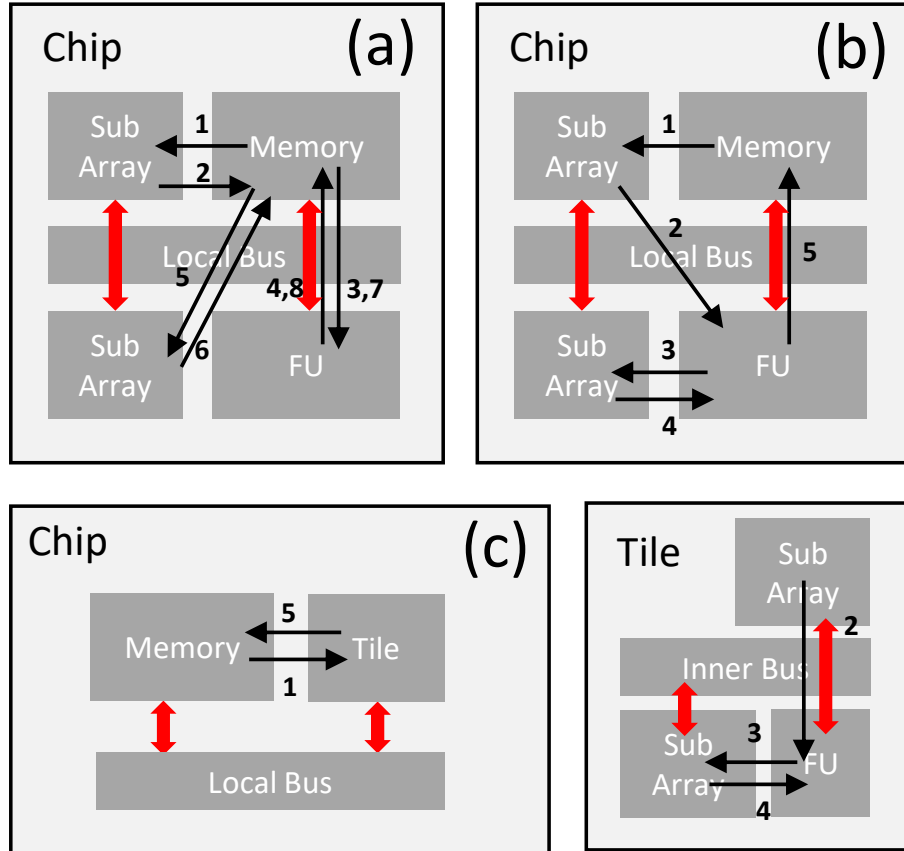


FIGURE 5.4: The illustration for MLP with (a) naive scheme, (b) intermediate data reuse scheme and (c) scheme with interconnection.

can fit into one sub-array exactly. All the three designs are composed of one memory to store intermediate data, two sub-arrays to compute matrix multiplications, and one FU for activation functions. In (a) and (b), all modules are connected by a local bus, and sub-arrays have their local connections with FU in (c). Input vector is pre-loaded into the memory and fetched into sub-arrays to perform matrix multiplication. Then the results are fed into FU to perform activation. The final results are written back to the memory.

For the naive scheme, all intermediate results are stored in the memory without any data reuse. The corresponding data flow is presented in Figure 5.4 (a). Input vector is directly fetched into the sub-array where the weights of the first layer are stored (1). Then the sub-array performs computation and writes the results back to the memory (2). Since



bus between sub-arrays and FU is added, which makes the local bus free of intermediate data transmission.

## 5.2.2 Modeling Large-Scale DCNN Accelerator

In this section, we present several PIM-based designs for accelerating convolution layers. The architectures are shown in Figure 5.5 (a). The overall architecture follows the design principles in ISAAC [134]. The accelerator contains several tiles, connected with a global bus. In each tile, there are multiple IMAs, max-pooling unit, activation unit, a vector adder, and I/O buffers. The IMA contains several memory sub-arrays, and related local I/O buffers.

We first present the baseline scheme. In the baseline scheme, one CNN layer is operated at a time, and one layer of the model is mapped to one tile. The 4-D kernels with dimensions of  $(Kh, Kw, Ich, Och)$  are flattened into a 2-D matrix with dimensions of  $(Kh \times Kw \times Ich, Och)$ . Then the 2-D matrix is split into several sub-matrices, according to the maximum size of one IMA. Each sub-matrix is mapped into one IMA, and the partial results generated by the IMA are added together in the vector adder. In the baseline scheme, we do not explore the sub-array level data reuse. Thus, there is no inter sub-array data communication during the matrix multiplication.

To further boost the performance and energy efficiency, we present three unique techniques for PIM based accelerator in existing works by exploring data parallelism and data reuse, and pipeline.

### 5.2.2.1 Data Parallelism

Data parallelism is employed in ISAAC [134] to perform the convolution in parallel. In PIM-based architecture, weights of a layer are duplicated multiple times and mapped into different IMAs. Feature maps are split spatially and fed into different IMAs to perform convolution in parallel. We define the degree of data parallelism as the number of replications of the weights. Figure 5.5 (b) shows an example to illustrate the concept of data parallelism. In this example, FU is omitted for simplicity. We assume the kernels of

this layer can be exactly mapped into one IMA. For the case without data parallelism, all kernels are mapped into one IMA. Input feature maps and output feature maps are transferred between this IMA and the buffer. When data parallelism is 2, the weight kernel is duplicated for two times and mapped into 2 IMAs. Input feature maps are also partitioned into two parts, and processed by two IMAs in parallel.

### 5.2.2.2 Data Reuse

The goal of applying data reuse is to reduce the memory access cost by exploring the opportunity to fetch the data from low cost memory directly. In CMOS-based accelerators, input feature maps, weights, and partial sums are stored in the main memory. Thus, the design space to explore is pretty large. Input reuse, weight reuse, and output reuse are all possible for CMOS-based accelerators. However, in PIM-based architectures, all the weights are pre-mapped into the sub-arrays, which makes weights naturally reused. To the best of our knowledge, few PIM-based design employs output reuse. The major data reuse scheme applied in PIM-based designs is input reuse, which is presented in previous works [189]. Figure 5.5 (a) shows the IMA level design principle to handle the input reuse scheme. The sub-array in IMA forms a 2-D array, with interconnect with adjunct sub-arrays. Data can be directly fetched into the InDFF in each sub-array from input register (IR), or be streamed into the InDFF from adjunct sub-array. With this unique interconnection, sub-array level data reuse is achieved to reduce the memory access overheads.

Figure 5.5 (c) shows an example to compute a conv2x2 with such sub-array level data reuse. The 4-D kernels with dimension of  $(K_h, K_w, I_ch, O_ch)$  are split into a  $K_h \times K_w$  2-D matrix with dimension of  $(I_ch, O_ch)$  and mapped into each sub-array. If the matrix dimension  $(I_ch, O_ch)$  is greater than the maximum matrix size provided by a sub-array, the matrix would be further split and mapped to different sub-arrays according to the principles of baseline scheme. The elements of input feature maps would be fetched into corresponding sub-array before the computation starts (IFM[:,0][0]~IFM[:,1][1] are mapped into XB[0][0]~XB[1][1]). Each sub-array would generate a partial sum with size

of  $1 \times 1 \times O_c h$ , and these partial sums are summed up together. At the next cycle, input feature maps can be reused by streaming the input feature maps located into the InDFF of subarray (IFM[:,0][1]~IFM[:,1][2] are mapped into XB[0][0]4~XB[1][1]). In this way, only 1/2 input data are accessed from IR, and the other 1/2 input data are reused from adjunct sub-arrays. With input reuse, the number of memory accesses required by each input element decreases from  $K_h \times K_w$  to  $\min(K_h, K_w)$ .

### 5.2.2.3 Pipeline

The goal of inter-layer pipeline is to reduce the overall processing latency and the memory footprint. The example in Figure 5.5 (d) shows an example for this inter-layer pipeline. In non-pipeline scheme, each convolution layer is executed when the previous layer finishes. Considering the data dependency, the convolution layer could be early executed when all input data is ready. Thus, there would be some overlap between layers. In addition, the capacity of data buffer could also be reduced. Once the feature map data of previous layer has been processed for  $K_h \times K_w$  times, it could be thrown away since this data is no longer used. Generally, in a conv2x2, the minimum data buffer for each layer is  $(H + 2) * I_{ch}$ , and when this data buffer is fulfilled, the execution could start without waiting all the input feature maps are ready.

## 5.3 Experiment Setup

We implement the architecture mentioned in Section 5.2 with PIMulator-NN. For small scale architecture, there are two sub-arrays, one memory buffer and one FU. The width of local bus and inner bus is 8 bytes and 128 bytes, respectively. The size of each sub-array is  $256 \times 256$ , and the size of memory buffer is 512 bytes. The NN model is a simple MLP, which consists of one input layer with a  $256 \times 256$  weight matrix and one output layer with a  $256 \times 256$  weight matrix. For activation functions, we implement a ReLU activation module in FU.

For large scale architecture, there are 16 tiles in the accelerator. In each tile, there are 18 IMAs, one input buffer, one output buffer, and one FU. The size of input buffer

and output buffer is 64KB and 32KB, respectively. FU can perform max pooling, vector accumulation and ReLU activation. In each IMA, there are 16 sub-arrays (each  $256 \times 256$ ), one input register and one output register. The input register and output register are both 1KB. For the architecture with sub-array level data reuse, a VPE is added to each IMA to perform vector accumulation across sub-arrays. We choose a simple VGG-13 model as the benchmark for CIFAR-10 dataset. In this case study, we only consider the convolution layer because convolution consumes the most time during DCNN inference.

We build the SRAM and sub-array with NVSIM and NeuroSIM to generate circuit-level results. The technology node is set to 32nm, and the clock frequency of the system is set to 1GHz. Note that the area of sub-array generated from NeuroSIM is greater than the original results in ISAAC ( $13083 \mu m^2$ ). The reason is that NeuroSIM uses 1T1R structure as memory cell, but ISAAC uses 1R structure. Since 1R structure is not mature currently, our estimation results are more practical.

## **5.4 Validation**

To ensure the simulation results are reasonable, we validate the simulation output based on low level simulation results. Generally, it is hard to validate the simulator as there are limited resources for real large scale IMC/PIM based chips. Thus, we mainly relies on RTL and power simulation to generate the performance, latency results and the power results, while the area is directly achieved from RTL synthesis process.

For the performance and latency validation, we implement the architecture mentioned above in RTL and simulate the proposed workloads. The performance and latency comes from the total cycle count and the clock frequency. The non-digital blocks, such as IMC/PIM macro, are implemented as a behaviour function which is parameterized of the computing latency.

The power validation is based on post-synthesis power simulation based on Design Compiler and PrimePower. The non-digital blocks are implemented as a behaviour function which is parameterized of several power states depends on whether the macro is activated.

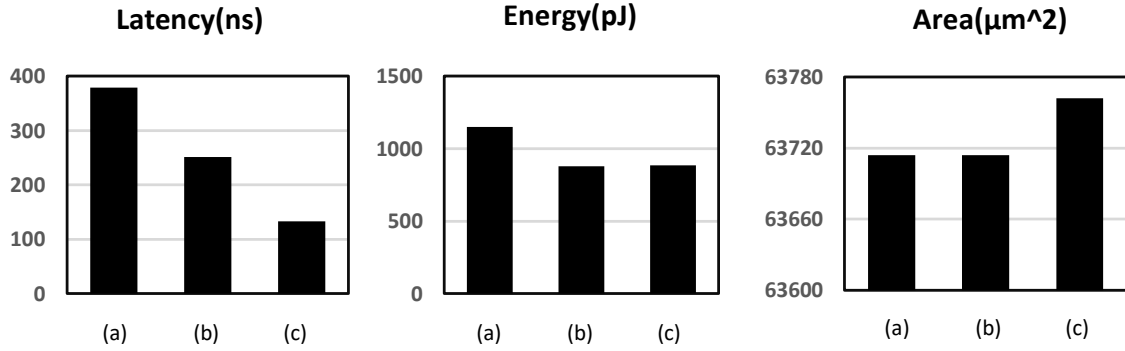


FIGURE 5.6: Power, performance and area results of three design schemes on MLP.

## 5.5 Key Observations

In this section, we would like to present some key observations from simulation results of PIMulator-NN. These results are generated by this event-driven simulation framework, and these results can not be accurately modeled by prior performance model estimation. These key observations are beneficial to help the architects to design PIM-based NN accelerators with higher performance and lower power consumption.

### 5.5.1 Results for Small-Scale DNN Accelerator

#### 5.5.1.1 Impacts of Execution Flow

**For PIM-based architecture, the execution flow impacts the results, even when architecture is the same.** Figure 5.6 shows the area, energy and latency results for three designs. As expected, the processing latency of scheme (a) without operation fusion is the highest (379 ns). The energy for this scheme is also the highest (884 pJ). This is because that all the intermediate data would be saved to memory. These saving operations would induce multiple memory accesses. Since scheme (b) with operation fusion benefits from the memory access reduction, both processing latency and energy are lower.

#### 5.5.1.2 Impacts of Interconnect

**High bandwidth on local bus would significantly improve the performance with considerable area overheads.** Scheme (c) achieves the lowest processing latency (133 ns) among

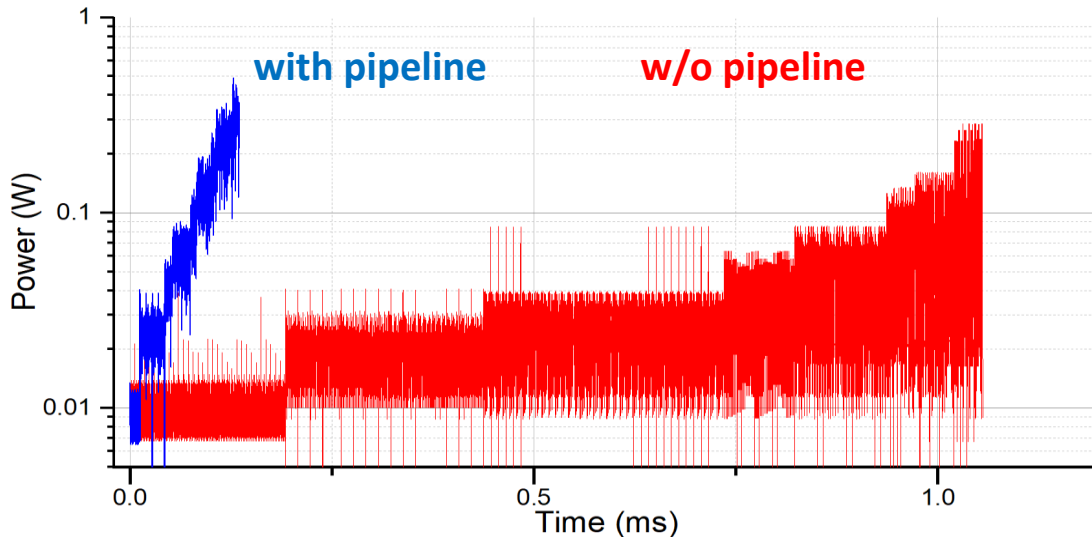


FIGURE 5.7: Extracted power traces of the baseline system during processing convolution layers of VGG-13.

the three schemes. The latency reduction mainly comes from the high-bandwidth inner bus. Since the system bus would route long distance, its width is usually smaller than inner bus. However, a high-bandwidth inner bus usually consumes larger energy and takes extra area. Thus, the energy and area of scheme (c) are slightly greater (less than 1%) than those of scheme (b).

## 5.5.2 Results for Large-Scale DCNN Accelerator

### 5.5.2.1 Power

The power results should be carefully simulated for different layer configurations. We found that it is not accurate to estimate the power by applying a simple power model, and power results would be overestimated. In simple power models, established in ISAAC, the overall power is accumulated among all sub-modules. Thus, it is hard to analyze the power traces for different model configurations. Power traces are critical to analyze the energy consumption of PIM-based architecture. Though sub-array computation dominates the overall energy consumption, not all the sub-arrays are activated in parallel due to data dependency and memory access latency. This case gets worse for smaller models. Figure 5.7

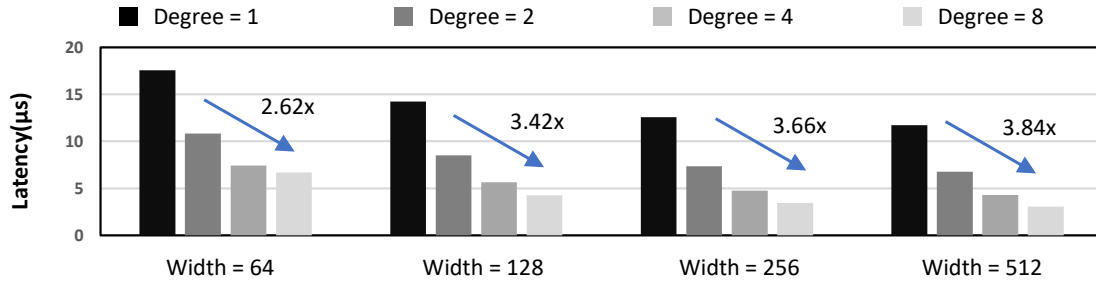


FIGURE 5.8: The latency and energy results for baseline design to process layer0 in VGG-13 with different degrees of parallelism and interconnect width.

shows a power trace generated from the event-driven simulation of PIMulator-NN. For layer 0, the average power consumption is about 10mW, and it could reach 650mW to process layer 8 and layer 9. However, in ISAAC, the power consumption in simple power model is fixed to 330mW for each tile. For layers with smaller kernel size, the power consumption is over estimated sine not all the sub-arrays are activated at the same time. In conclusion, PIMulator-NN could offer users with detailed trace results to estimate power.

We also observed that, pipeline scheme could increase the peak power consumption. As shown in Figure 5.7, the pipeline scheme in ISAAC could achieve lower processing latency (reduction) than non-pipeline scheme. However, this benefit comes from a large peak power. The peak power increases by about  $3\times$  over non-pipeline scheme.

### 5.5.2.2 Data Parallelism

**The benefit of data parallelism is highly related to memory access and interconnect bandwidth.** Figure 5.8 shows the latency results of processing layer 0. Theoretically, performance benefit is proportional to the data parallelism. When we increase the parallelism degree from 1 to 8, the processing latency decreases from 17.57us to 6.69us for bus width equaling to 64. This performance benefit is lower than theoretical ones because the data communication and memory access need to be serial. When we increase the degree of parallelism, we copy the kernels and map them into different IMAs to process feature maps in parallel. In this case, multiple IMAs would issue a memory access command to load data from in-tile buffer, and this access command would be processed in serial. As expected,

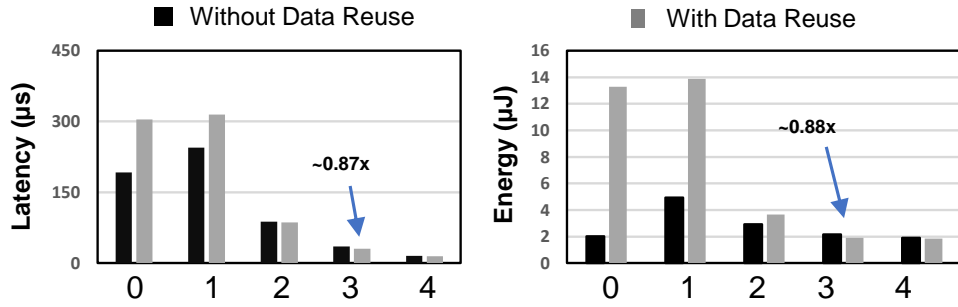


FIGURE 5.9: The energy and latency results to process convolution layers with different configurations of data reuse scheme and baseline scheme.

when we increase the width of the local bus, the benefit of data parallelism would also increase. The performance gain is 2.62x, 3.42x, 3.66x and 3.84x for bus width equaling to 64, 128, 256 and 512, respectively.

In conclusion, the data parallelism for PIM-based architecture should be co-designed with buffer and interconnection. For some embedded PIM-based designs, the memory bandwidth and bus width is relatively small. In these cases, increasing data parallelism cannot actually improve the performance. Even worse, it may increase the peak power as IMAs are running concurrently. Thus, data parallelism is not very applicable in embedded PIM-based designs.

### 5.5.2.3 Data Reuse

**Data reuse scheme is not always useful for providing a low-cost processing.** The benefit highly depends on layer configurations and hardware configurations. Figure 5.9 shows the comparative results of data reuse scheme and baseline for several layer configurations. For layer 0, 1, and 2, the kernel configuration is  $3 \times 3 \times 3 \times 64$ ,  $3 \times 3 \times 128 \times 128$  and  $3 \times 3 \times 128 \times 128$ , respectively. The results are against common senses: the latency and energy for these layers become larger when data reuse scheme is applied. The reason is that the mapping efficiency of data reuse scheme is smaller than the design without data reuse. For layer 3 and 4, the energy and latency results would benefit from data reuse scheme. By reducing memory access from higher level memory, energy and latency could be reduced by

12%, 4% and 16%, 7% for layer 3 and 4, respectively.

The benefit of sub-array level data reuse is quite small for PIM -based design targeting high performance applications. This is because that PIM-based architecture is not communication bounded. For most designs, the on-chip interconnection bandwidth is large, and an in-tile buffer is designed to avoid off-chip memory accesses. As a result, data reuse scheme would contribute inconsiderable improvements. However, for some low-power PIM-based designs, both memory bandwidth and bus width are small. This would increase the overheads of memory access from higher level buffer. In this case, data reuse scheme would benefit the system performance and energy efficiency. We made an additional experiment and the results show that, up to 50% energy and latency could be saved when bus width is 1 byte, which is common for embedded PIM-based design.

In conclusion, the data reuse scheme is only applicable for specific model configurations. For layers with small sizes, the data reuse scheme provides negative effects to latency and energy, as the mapping efficiency of these layers are quite low. The benefits reach the highest level when the number of input channels exactly matches the row size of sub-array. For layers with larger sizes, the benefits of data reuse degenerate since the inter-module communication induces a great amount of costs. Data reuse is unable to mitigate these costs.

#### 5.5.2.4 Pipeline Scheme and Data Parallelism

**PIMulator-NN could present the trade-off between resources requirement and latency under pipeline scheme.** We tried different parallelism of different layers and different number of IMA in each tile. One interesting findings is that, when the pipeline scheme is used, increasing the data parallelism could not greatly reduced the processing latency. As shown in Figure 5.10, the baseline parallelism setting (each layer has parallelism degree of 1), the latency is about 135  $\mu s$ . For this setting, 18 IMAs is needed to support very deep layers in each tile. Free speed up could be achieved by increasing the parallelism degree of shallow layer from 1 to 4. This free speed up comes at the effect that, shallow layers consumes less

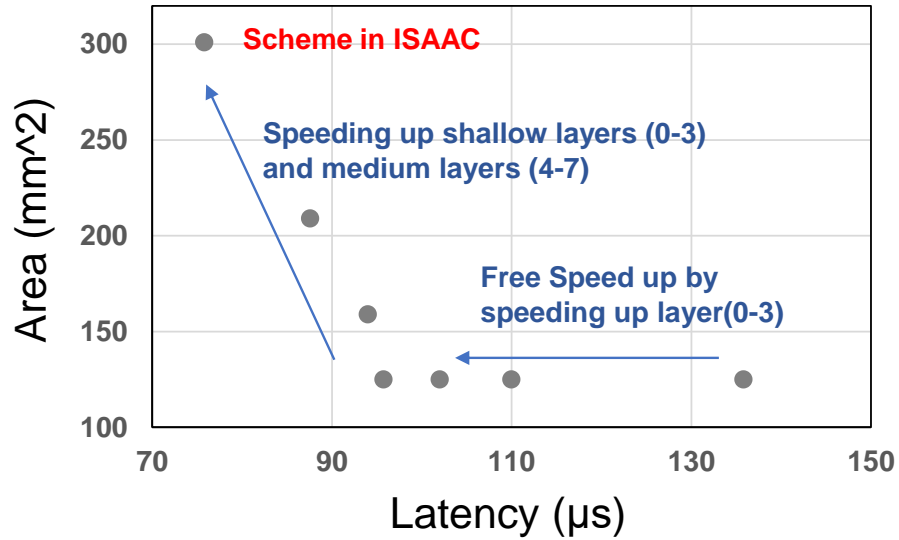


FIGURE 5.10: The trade-off between latency and area under different data parallelism and pipeline scheme.

crossbar than deep layers but they consumes significant latency. Increasing the parallelism degree of these layers does not contribute area overhead of the whole system as 18 IMAs could support maximum parallelism of these layers to be 4. The lowest processing latency comes at ISAAC's scheme (The parallelism degree of the final layer is 1, and increases by 2 when feature map spatial size increases by 2). However, this scheme significantly increases the area since it need more crossbar in each Tile. Increasing the parallelism degree of medium and very deep layer (layer 4-9) could significant increase the area overhead. By using PIMulator-NN, the user could also find some sub-optimal configuration with different degree of parallelism and number of IMAs in each tile.

## 5.6 Conclusion

In this paper, we present PIMulator-NN, an event-driven, cross-level simulation framework for PIM-based NN accelerators. The features, usage and limitations of current version are detailed described. The major feature of PIMulator-NN is that it could model very detailed architecture information, including interconnection scheme, pipeline scheme and data reuse scheme. With well-designed event-driven simulation core and well-defined architecture

templates, users could easily implement their own PIM-based architecture in PIMulator-NN. Furthermore, we demonstrate the usage of PIMulator-NN in small-scale MLP models and large-scale CNN models. By applying several architectural optimization techniques, the impacts of memory access and interconnect could be accurately modeled. These simulation results could enable user to explore their design space efficiently. In the future, we would like to contribute more architecture templates and integrate PIMulator-NN with other simulators for system-level simulations.

## 6. Conclusion

### 6.1 Summary of Contributions

This thesis explores the development of IMC/PIM based computing systems for ML applications, focusing on solving the micro-architecture and circuit level challenges. More specifically, we observed that, conventional IMC/PIM based computing systems are optimized for conventional ML workloads, which is converted to matrix-multiplication primitives. In addition, we observed that, conventional IMC/PIM design methodology, based on fully-analog implementation, suffers from low computing precision issue. The computation primitives challenge and the precision challenge is a big bottleneck to enable utilizing IMC/PIM techniques to build an efficient IMC/PIM based computing system for ML applications.

To address these challenges, we present a unified solution, a hybrid digital/analog design approach to provide both computation pattern flexibility and computation precision flexibility. Firstly, we introduces a novel concept for enhancing the efficiency of mobile applications through hybrid digital/analog in-memory computing, where the digital mode can deliver a vector-wise multiplication and accumulation primitives for the depth-wise convolution with low data reuse opportunities. Then, the work extends this concept to improve the processing of sparse attention operators, thereby enhancing mapping efficiency, where the digital mode is used to provide a vector-vector inner-product and scalar-vector multiplication primitives for fine-grained structural sparsity. Additionally, we also presents a circuit-level implementation to use the proposed primitives and a data-aware datapath selection computing method to increase the power efficiency of IMC/PIM macro for loss-less computation. The proposed data-aware data path selection computing method is also based on the concept of hybrid digital/analog computing, where a data-aware mechanism is established to choose the high throughput analog mode or high precision digital mode. The proposed method could reduce the interface complexity of the analog mode with an enhanced energy efficiency while maintaining the computation throughput and accuracy.

Finally, a cross-level simulation tool for versatile architecture exploration in IMC/PIM based systems is introduced, to justify the idea of both circuit-level innovations and micro-architecture level innovations. This research contributes to advancing in-memory computing technologies, offering solutions to meet the computational challenges of next-generation ML applications.

## **6.2 Future Work**

At a broader perspective of my research, I have concentrated on enhancing the micro-architecture and circuit-level optimizations to boost both efficiency and flexibility for machine learning acceleration. As the landscape of ML models continues to evolve, a myriad of research opportunities emerges, particularly at the micro-architecture and circuit levels.

### **6.2.1 Micro architecture level**

Within the realm of micro-architecture, a significant challenge lies in addressing the interconnect issues present in current IMC/PIM computing systems. Traditional memory organization does not necessitate highly optimized internal bandwidth, as there lacks a need for bank-to-bank communication to facilitate memory operations. However, integrating computational functionalities within memory demands meticulous optimization of on-chip interconnects to ensure seamless data communication between different banks, thereby preventing computational bottlenecks in IMC/PIM banks. The extent to which such modifications adhere to traditional memory chip design principles, without compromising memory capacity, remains an area underexplored by comprehensive studies.

### **6.2.2 Circuit level**

On the circuit level, the incorporation of floating-point computation capabilities is crucial for contemporary ML training systems. For example, the design methodology of the hybrid digital/analog IMC can be further extended to support floating-point computing with the modification of the digital components in the IMC macro. Moreover, the adaptation to new, complex computation patterns introduced by versatile operators in emerging

ML workloads necessitates further support from IMC/PIM systems.

Reliability issue is another consideration when implements the IMC/PIM based macro. By incorporating the design methodology of hybrid digital/analog, we are able to find an optimal way to enhance the efficiency of the computing circuits without reducing the computing robustness with a multi-supply voltage for digital and analog domain, and some timing borrowing scheme, or digital assisted error correction scheme for the analog data path.

### **6.2.3 Architecture level**

Despite these advancements, focusing solely on circuit and micro-architecture levels is insufficient for realizing a state-of-the-art computing system capable of deploying real-world ML models. Two overarching issues at the system architecture level have yet to be thoroughly addressed by the research community. Firstly, the scalability of IMC/PIM computing systems poses a challenge as ML models expand, with existing memory technologies struggling to meet the demands for model and intermediate result storage. This scalability concern raises questions about the continued viability of overcoming the "memory wall" when inter-chip data communication becomes a necessity.

Secondly, system programmability presents a significant hurdle. The stringent requirements for data layout and alignment in current IMC/PIM systems vastly differ from those of conventional computing systems, complicating compiler design. Integrating IMC/PIM technology into real-world systems would require compilers to manage not only compile-time data layout but also run-time data reorganization. This complexity could potentially negate the advantages offered by IMC/PIM systems, emphasizing the need for innovative solutions to these systemic challenges.

## ***6.3 From Academia To Industry***

In this section, we would like to discuss several possible applications to deploy my research outcome, especially the design for IMC/PIM technology into real worlds. The most promising application emerges in delivering edge intelligence to biomedical devices and In-

ternet of Things (IoT) devices, as evidenced by recent studies such as [2]. The primary rationale for this is the critical nature of power in biomedical and IoT devices, coupled with their workloads that are characterized by simplicity, compact model sizes, and intermediate results. For common applications like keyword spotting and human activity detection, models typically require less than 1MB. This scenario makes it cost-effective to construct medium-sized IMC/PIM circuits capable of accommodating the entire model. Essentially, IMC/PIM memory could serve as the last level memory in the system architecture, highlighting its significance in this application domain. From a hardware perspective, the advantage of IMC/PIM is profound as it significantly reduces on-chip data movement, especially between the weight buffer and the processing elements. On the software side, the algorithms intended for deployment on such devices are generally stable, negating the need for a complex software stack. Moreover, these algorithms can be intricately tailored to the hardware, allowing for aggressive optimization of quantization or network architecture.

In conclusion, for IoT devices, the IMC/PIM approach addresses the critical challenge of data movement between on-chip memory and processing elements effectively. Although the IMC/PIM scheme has its limitations, including lower precision and a fixed computation pattern, these drawbacks are mitigated by the potential for co-designing the algorithm with the hardware, underscoring its viability and effectiveness.

## Bibliography

- [1] Mohamed R Abdelhamid et al. "Self-reconfigurable micro-implants for cross-tissue wireless and batteryless connectivity". In: *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. IEEE. 2020, pp. 1–14.
- [2] Shaizeen Aga et al. "Compute caches". In: *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2017, pp. 481–492.
- [3] Akshay Agrawal et al. "A Rewriting System for Convex Optimization Problems". In: *Journal of Control and Decision* 5.1 (2018), pp. 42–60.
- [4] Fabien Alibart et al. "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm". In: *Nanotechnology* 23.7 (2012), p. 075201.
- [5] Stefano Ambrogio et al. "Impact of low-frequency noise on read distributions of resistive switching memory (RRAM)". In: *2014 IEEE International Electron Devices Meeting*. IEEE. 2014, pp. 14–4.
- [6] Salvatore M Amoroso et al. "Investigation of the RTN distribution of nanoscale MOS devices from subthreshold to on-state". In: *IEEE Electron Device Letters* 34.5 (2013), pp. 683–685.
- [7] Shaahin Angizi et al. "Cmp-pim: an energy-efficient comparator-based processing-in-memory neural network accelerator". In: *Proceedings of the 55th Annual Design Automation Conference*. 2018, pp. 1–6.
- [8] Aayush Ankit et al. "Resparc: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks". In: *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM. 2017, p. 27.
- [9] Simone Balatti et al. "True random number generation by variability of resistive switching in oxide-based devices". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 5.2 (2015), pp. 214–221.
- [10] Subho S Banerjee, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. "AcMC 2: Accelerating Markov Chain Monte Carlo Algorithms for Probabilistic Models". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM. 2019, pp. 515–528.
- [11] Iz Beltagy, Matthew E Peters, and Arman Cohan. "Longformer: The long-document transformer". In: *arXiv preprint arXiv:2004.05150* (2020).
- [12] Nathan Binkert et al. "The gem5 simulator". In: *ACM SIGARCH computer architecture news* 39.2 (2011), pp. 1–7.

- [13] Avishek Biswas et al. "CONV-SRAM: An energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks". In: *JSSC* (2018).
- [14] Ralf Brederlow et al. "A low-power true random number generator using random telegraph noise of single oxide-traps". In: *2006 IEEE International Solid State Circuits Conference-Digest of Technical Papers*. IEEE. 2006, pp. 1666–1675.
- [15] Yoeri van de Burgt et al. "A non-volatile organic electrochemical device as a low-voltage artificial synapse for neuromorphic computing". In: *Nature materials* 16.4 (2017), pp. 414–418.
- [16] Dae Seok Byeon et al. "Disturbance-suppressed ReRAM write algorithm for high-capacity and high-performance memory". In: *2014 14th Annual Non-Volatile Memory Technology Symposium (NVMTS)*. IEEE. 2014, pp. 1–4.
- [17] Fuxi Cai et al. "A fully integrated reprogrammable memristor-CMOS system for efficient multiply-accumulate operations". In: *Nature Electronics* 2.7 (2019), pp. 290–299.
- [18] Ruizhe Cai et al. "A stochastic-computing based deep learning framework using adiabatic quantum-flux-parametron superconducting technology". In: *Proceedings of the 46th International Symposium on Computer Architecture*. ACM. 2019, pp. 567–578.
- [19] Muya Chang et al. "A 40nm 60.64 TOPS/W ECC-Capable Compute-in-Memory/Digital 2.25 MB/768KB RRAM/SRAM System with Embedded Cortex M3 Microprocessor for Edge Recommendation Systems". In: *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 65. IEEE. 2022, pp. 1–3.
- [20] Ching-Yi Chen et al. "RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme". In: *IEEE Transactions on Computers* 64.1 (2014), pp. 180–190.
- [21] Fan Chen et al. "Zara: A novel zero-free dataflow accelerator for generative adversarial networks in 3d reram". In: *Proceedings of the 56th Annual Design Automation Conference 2019*. 2019, pp. 1–6.
- [22] Jou-Fan Chen et al. "Financial Time-Series Data Analysis Using Deep Convolutional Neural Networks". In: *7th International Conference on Cloud Computing and Big Data, CCBD 2016, Macau, China, November 16-18, 2016*. 2016, pp. 87–92. DOI: 10.1109/CCBD.2016.027. URL: <https://doi.org/10.1109/CCBD.2016.027>.
- [23] Lerong Chen et al. "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar". In: *Proceedings of the Conference on Design*,

*Automation & Test in Europe*. European Design and Automation Association. 2017, pp. 19–24.

- [24] Pai-Yu Chen, Xiaochen Peng, and Shimeng Yu. “NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.12 (2018), pp. 3067–3080.
- [25] Ruicong Chen. “Analog-to-Digital Converters for Secure and Emerging AIoT Applications”. PhD thesis. Massachusetts Institute of Technology, 2023.
- [26] Ruicong Chen, Anantha Chandrakasan, and Hae-Seung Lee. “Sniff-SAR: A 9.8 fJ/c.s 12b secure ADC with detectiondriven protection against power and EM side-channel attack”. In: *2023 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE. 2023, pp. 1–2.
- [27] Ruicong Chen et al. “A Bit-level Sparsity-aware SAR ADC with Direct Hybrid Encoding for Signed Expressions for AIoT Applications”. In: *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 2022, pp. 1–6.
- [28] Ruicong Chen et al. “RaM-SAR: A Low Energy and Area Overhead, 11.3 fJ/conv.-step 12b 25MS/s Secure Random-Mapping SAR ADC with Power and EM Side-channel Attack Resilience”. In: *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE. 2022, pp. 94–95.
- [29] Wei-Hao Chen et al. “A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors”. In: *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE. 2018, pp. 494–496.
- [30] Xiaoming Chen et al. “Design and optimization of FeFET-based crossbars for binary convolution neural networks”. In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2018, pp. 1205–1210.
- [31] Yang Yin Chen et al. “Improvement of data retention in HfO<sub>2</sub>/Hf 1T1R RRAM cell under low operating current”. In: *2013 IEEE International Electron Devices Meeting*. IEEE. 2013, pp. 10–1.
- [32] Yiran Chen et al. “Access scheme of multi-level cell spin-transfer torque random access memory and its optimization”. In: *2010 53rd IEEE International Midwest Symposium on Circuits and Systems*. IEEE. 2010, pp. 1109–1112.
- [33] Zhiyu Chen et al. “CAP-RAM: A charge-domain in-memory computing 6T-SRAM for accurate and precision-programmable CNN inference”. In: *JSSC* (2021).

- [34] Ming Cheng et al. "Time: A training-in-memory architecture for memristor-based deep neural networks". In: *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM. 2017, p. 26.
- [35] Ping Chi et al. "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory". In: *ACM SIGARCH Computer Architecture News* 44.3 (2016), pp. 27–39.
- [36] Yu-Der Chih et al. "16.4 an 89tops/w and 16.3 tops/mm<sup>2</sup> all-digital sram-based full-precision compute-in memory macro in 22nm for machine-learning edge applications". In: *ISSCC*. 2021.
- [37] Yu-Der Chih et al. "An 89TOPS/W and 16.3 TOPS/mm<sup>2</sup> 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. IEEE. 2021, pp. 252–254.
- [38] Rewon Child et al. "Generating long sequences with sparse transformers". In: *arXiv preprint arXiv:1904.10509* (2019).
- [39] Yen-Cheng Chiu et al. "A 4-Kb 1-to-8-bit configurable 6T SRAM-based computation-in-memory unit-macro for CNN-based AI edge processors". In: *JSSC* (2020).
- [40] Edward Choi et al. "A 133.6 TOPS/W compute-in-memory SRAM macro with fully parallel one-step multi-bit computation". In: *CICC*. 2022.
- [41] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.
- [42] Teyuh Chou et al. "CASCADE: Connecting RRAMs to Extend Analog Dataflow In An End-To-End In-Memory Processing Paradigm". In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM. 2019, pp. 114–125.
- [43] Gonçalo M Correia, Vlad Niculae, and André FT Martins. "Adaptively sparse transformers". In: *arXiv preprint arXiv:1909.00015* (2019).
- [44] Baiyun Cui et al. "Fine-tune BERT with sparse self-attention mechanism". In: *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*. 2019, pp. 3548–3553.

- [45] G. Dai et al. "GraphH: A Processing-in-Memory Architecture for Large-Scale Graph Processing". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2019).
- [46] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.
- [47] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).
- [48] Ashutosh Dhar et al. "FReaC cache: folded-logic reconfigurable computing in the last level cache". In: *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. 2020, pp. 102–117.
- [49] Steven Diamond and Stephen Boyd. "CVXPY: A Python-Embedded Modeling Language for Convex Optimization". In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.
- [50] Xiangyu Dong et al. "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31.7 (2012), pp. 994–1007.
- [51] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).
- [52] Zidong Du et al. "ShiDianNao: Shifting vision processing closer to the sensor". In: *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 2015, pp. 92–104.
- [53] Charles Eckert et al. "Neural cache: Bit-serial in-cache acceleration of deep neural networks". In: *2018 ACM/IEEE 45th annual international symposium on computer architecture (ISCA)*. IEEE. 2018, pp. 383–396.
- [54] Charles Eckert et al. "Neural cache: Bit-serial in-cache acceleration of deep neural networks". In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2018, pp. 383–396.
- [55] Ligang Gao, Pai-Yu Chen, and Shimeng Yu. "Programming protocol optimization for analog weight tuning in resistive memories". In: *IEEE Electron Device Letters* 36.11 (2015), pp. 1157–1159.
- [56] Benjamin Graham et al. "LeViT: a Vision Transformer in ConvNet's Clothing for Faster Inference". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 12259–12269.

- [57] Saransh Gupta, Mohsen Imani, and Tajana Rosing. “Felix: Fast and energy-efficient logic in memory”. In: *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2018, pp. 1–7.
- [58] Tae Jun Ham et al. “A<sup>3</sup>: Accelerating attention mechanisms in neural networks with approximation”. In: *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2020, pp. 328–341.
- [59] Tae Jun Ham et al. “ELSA: Hardware-Software co-design for efficient, lightweight self-attention mechanism in neural networks”. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2021, pp. 692–705.
- [60] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [61] Zhezhi He et al. “Noise Injection Adaption: End-to-End ReRAM Crossbar Non-ideal Effect Adaption for Neural Network Mapping”. In: *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM. 2019, p. 57.
- [62] Pouya Houshmand et al. “Diana: An end-to-end hybrid digital and analog neural network soc for the edge”. In: *JSSC* (2022).
- [63] Andrew G Howard et al. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).
- [64] M. Hu et al. “Memristor crossbar based hardware realization of BSB recall function”. In: *The 2012 International Joint Conference on Neural Networks (IJCNN)*. 2012.
- [65] Miao Hu et al. “Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication”. In: *Proceedings of the 53rd annual design automation conference*. ACM. 2016, p. 19.
- [66] Miao Hu et al. “Memristor crossbar based hardware realization of BSB recall function”. In: *The 2012 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2012, pp. 1–7.
- [67] Miao Hu et al. “Memristor-based analog computation and neural network classification with a dot product engine”. In: *Advanced Materials* 30.9 (2018), p. 1705914.
- [68] Qilin Hua et al. “A threshold switching selector based on highly ordered Ag nanodots for X-point memory applications”. In: *Advanced Science* 6.10 (2019), p. 1900024.
- [69] Weizhe Hua et al. “Boosting the performance of cnn accelerators with dynamic fine-grained channel gating”. In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 2019, pp. 139–150.

- [70] Chien-Yuan Huang et al. "A contact-resistive random-access-memory-based true random number generator". In: *IEEE Electron Device Letters* 33.8 (2012), pp. 1108–1110.
- [71] Je-Min Hung et al. "An 8-Mb DC-Current-Free Binary-to-8b Precision ReRAM Nonvolatile Computing-in-Memory Macro using Time-Space-Readout with 1286.4-21.6 TOPS/W for Edge-AI Devices". In: *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 65. IEEE. 2022, pp. 1–3.
- [72] Daniele Ielmini, Federico Nardi, and Carlo Cagli. "Resistance-dependent amplitude of random telegraph-signal noise in resistive switching memories". In: *Applied Physics Letters* 96.5 (2010), p. 053503.
- [73] Ahmet Inci, Mehmet Meric Isgenc, and Diana Marculescu. "DeepNVM++: Cross-layer modeling and optimization framework of nonvolatile memories for deep learning". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.10 (2021), pp. 3426–3437.
- [74] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. "Spatial transformer networks". In: *Advances in neural information processing systems* 28 (2015).
- [75] Shubham Jain et al. "RxNN: A framework for evaluating deep neural networks on resistive crossbars". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.2 (2020), pp. 326–338.
- [76] Houxiang Ji et al. "ReCom: An efficient resistive accelerator for compressed deep neural networks". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2018, pp. 237–240.
- [77] Hao Jiang et al. "Pulse-width modulation based dot-product engine for neuromorphic computing system using memristor crossbar array". In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2018, pp. 1–4.
- [78] Norman P Jouppi et al. "In-datacenter performance analysis of a tensor processing unit". In: *Proceedings of the 44th annual international symposium on computer architecture*. 2017, pp. 1–12.
- [79] Liu Ke et al. "Recnmp: Accelerating personalized recommendation with near-memory processing". In: *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2020, pp. 790–803.
- [80] Win-San Khwa et al. "A 65nm 4Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3 ns and 55.8 TOPS/W fully parallel product-sum operation for binary DNN edge processors". In: *ISSCC*. 2018.

- [81] Hyunjoon Kim et al. "A 1-16b precision reconfigurable digital in-memory computing macro featuring column-mac architecture and bit-serial computation". In: *ESSCIRC 2019-IEEE 45th European Solid State Circuits Conference (ESSCIRC)*. IEEE. 2019, pp. 345–348.
- [82] Kyeonghan Kim et al. "An energy-efficient processing-in-memory architecture for long short term memory in spin orbit torque mram". In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2019, pp. 1–8.
- [83] Yulhwa Kim et al. "Input-Splitting of Large Neural Networks for Power-Efficient Accelerator with Resistive Crossbar Memory Array". In: *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM. 2018, p. 41.
- [84] LB Kish and CG Granqvist. "Noise in nanotechnology". In: *Microelectronics Reliability* 40.11 (2000), pp. 1833–1837.
- [85] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. "Reformer: The efficient transformer". In: *arXiv preprint arXiv:2001.04451* (2020).
- [86] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of Neural Information Processing Systems". In: *Proceedings of Neural Information Processing Systems (NIPS)*. 2012.
- [87] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Commun. ACM* 60.6 (2017), pp. 84–90. DOI: 10.1145/3065386. URL: <http://doi.acm.org/10.1145/3065386>.
- [88] Shahar Kvatinsky et al. "MAGIC—Memristor-aided logic". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 61.11 (2014), pp. 895–899.
- [89] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. "Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning". In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 2019, pp. 740–753.
- [90] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [91] Chia-Fu Lee et al. "A 12nm 121-TOPS/W 41.6-TOPS/mm<sup>2</sup> All Digital Full Precision SRAM-based Compute-in-Memory with Configurable Bit-width For AI Edge Applications". In: *VLSI*. 2022.

- [92] Jinseok Lee et al. "Fully Row /Column-Parallel In-memory Computing SRAM Macro employing Capacitor-based Mixed-signal Computation with 5-b Inputs". In: *VLSI*. 2021.
- [93] Sukhan Lee et al. "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product". In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2021, pp. 43–56.
- [94] Boxun Li et al. "Memristor-based approximated computation". In: *International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE. 2013, pp. 242–247.
- [95] Boxun Li et al. "Training itself: Mixed-signal training acceleration for memristor-based neural network". In: *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2014, pp. 361–366.
- [96] Shiyu Li et al. "Penni: Pruned kernel sharing for efficient CNN inference". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5863–5873.
- [97] Zirui Li et al. "ASTERS: adaptable threshold spike-timing neuromorphic design with twin-column ReRAM synapses". In: *DAC*. 2022.
- [98] Zirui Li, Bonan Yan, and Hai Li. "Resipe: Reram-based single-spiking processing-in-memory engine". In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2020, pp. 1–6.
- [99] Yu-Hsuan Lin et al. "Performance Impacts of Analog ReRAM Non-ideality on Neuromorphic Computing". In: *IEEE Transactions on Electron Devices* 66.3 (2019), pp. 1289–1295.
- [100] Yudeng Lin et al. "Bayesian Neural Network Realization by Exploiting Inherent Stochastic Characteristics of Analog RRAM". In: *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE. 2019, pp. 14–6.
- [101] Beiye Liu et al. "Digital-assisted noise-eliminating training for memristor crossbar-based analog neuromorphic computing engine". In: *Proceedings of the 50th Annual Design Automation Conference*. 2013, pp. 1–6.
- [102] Beiye Liu et al. "Reduction and IR-drop compensations techniques for reliable neuromorphic computing systems". In: *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press. 2014, pp. 63–70.
- [103] Beiye Liu et al. "Vortex: variation-aware training for memristor x-bar". In: *Proceedings of the 52nd Annual Design Automation Conference*. ACM. 2015, p. 15.

- [104] Chenchen Liu et al. "A memristor crossbar based computing engine optimized for high speed and accuracy". In: *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE. 2016, pp. 110–115.
- [105] Chenchen Liu et al. "A spiking neuromorphic design with resistive crossbar". In: *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE. 2015, pp. 1–6.
- [106] Xiaoxiao Liu et al. "Harmonica: A framework of heterogeneous computing systems with memristor-based neuromorphic computing accelerators". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 63.5 (2016), pp. 617–628.
- [107] Xiaoxiao Liu et al. "RENO: A high-efficient reconfigurable neuromorphic computing accelerator design". In: *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE. 2015, pp. 1–6.
- [108] Liqiang Lu et al. "Sanger: A Co-Design Framework for Enabling Sparse Attention using Reconfigurable Architecture". In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 2021, pp. 977–991.
- [109] Wolfgang Maass. "Networks of spiking neurons: the third generation of neural network models". In: *Neural networks* 10.9 (1997), pp. 1659–1671.
- [110] Jiachen Mao et al. "Modnn: Local distributed mobile computing system for deep neural network". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE. 2017, pp. 1396–1401.
- [111] *MAX5152, Low-Power, Dual, 13-Bit Voltage-Output DACs with Configurable Outputs*. <https://www.maximintegrated.com/en/products/analog/data-converters/digital-to-analog-converters/MAX5152.html>. Accessed: 2019-11-10.
- [112] Emmanuelle J Merced-Grafals et al. "Repeatable, accurate, and high speed multi-level programming of memristor 1T1R arrays for power efficient analog computing applications". In: *Nanotechnology* 27.36 (2016), p. 365202.
- [113] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0". In: *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society. 2007, pp. 3–14.
- [114] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. "CACTI 6.0: A tool to model large caches". In: *HP laboratories* 27 (2009), p. 28.
- [115] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. "CACTI 6.0: A Tool to Model Large Caches". In: 2009.

- [116] B Murmann. *ADC Performance Survey 1997-2019 (ISSCC & VLSI Symposium)*. <https://web.stanford.edu/~murmam/adcsurvey.html>. Accessed: 2019-11-18.
- [117] Leibin Ni et al. "An energy-efficient and high-throughput bitwise CNN on sneak-path-free digital ReRAM crossbar". In: *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE. 2017, pp. 1–6.
- [118] Shunsuke Okumura et al. "A Ternary Based Bit Scalable, 8.80 TOPS/W CNN accelerator with Many-core Processing-in-memory Architecture with 896K synapses/mm<sup>2</sup>." In: *2019 Symposium on VLSI Circuits*. IEEE. 2019, pp. C248–C249.
- [119] Ioannis A Papistas et al. "A 22 nm, 1540 TOP/s/W, 12.1 TOP/s/mm<sup>2</sup> in-memory analog matrix-vector-multiplier for DNN acceleration". In: *2021 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE. 2021, pp. 1–2.
- [120] Marcel JM Pelgrom, Aad CJ Duinmaijer, and Anton PG Welbers. "Matching properties of MOS transistors". In: *IEEE Journal of solid-state circuits* 24.5 (1989), pp. 1433–1439.
- [121] Xiaochen Peng et al. "DNN+ NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies". In: *2019 IEEE international electron devices meeting (IEDM)*. IEEE. 2019, pp. 32–5.
- [122] Matt Poremba and Yuan Xie. "Nvmain: An architectural-level main memory simulator for emerging non-volatile memories". In: *2012 IEEE Computer Society Annual Symposium on VLSI*. IEEE. 2012, pp. 392–397.
- [123] Francesco Maria Puglisi et al. "A complete statistical investigation of RTN in HfO<sub>2</sub>-based RRAM in high resistive state". In: *IEEE Transactions on Electron Devices* 62.8 (2015), pp. 2606–2613.
- [124] Zheng Qu et al. "DOTA: detect and omit weak attentions for scalable transformer acceleration". In: *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2022, pp. 14–26.
- [125] Md Tauhidur Rahman et al. "TI-TRNG: Technology independent true random number generator". In: *Proceedings of the 51st Annual Design Automation Conference*. ACM. 2014, pp. 1–6.
- [126] Pranav Rajpurkar et al. "Squad: 100,000+ questions for machine comprehension of text". In: *arXiv preprint arXiv:1606.05250* (2016).
- [127] S Ramey et al. "Transistor reliability variation correlation to threshold voltage". In: *2015 IEEE International Reliability Physics Symposium*. IEEE. 2015, 3B–2.

- [128] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 39.6 (2017), pp. 1137–1149. DOI: 10.1109/TPAMI.2016.2577031. URL: <https://doi.org/10.1109/TPAMI.2016.2577031>.
- [129] G. Ross. "Fast r-cnn". In: *Proceedings of IEEE international conference on computer vision*. 2015.
- [130] Aurko Roy et al. "Efficient content-based sparse attention with routing transformers". In: *Transactions of the Association for Computational Linguistics* 9 (2021), pp. 53–68.
- [131] M. Saberi et al. "Analysis of power consumption and linearity in capacitive digital-to-analog converters used in successive approximation ADCs". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* (2011).
- [132] Mark Sandler et al. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.
- [133] Suproteem K. Sarkar et al. "Robust Classification of Financial Risk". In: *CoRR* abs/1811.11079 (2018). arXiv: 1811.11079. URL: <http://arxiv.org/abs/1811.11079>.
- [134] Ali Shafiee et al. "ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars". In: *Proceedings of the 43rd International Symposium on Computer Architecture*. 2016, pp. 14–26.
- [135] Xin Si et al. "15.5 A 28nm 64Kb 6T SRAM computing-in-memory macro with 8b MAC operation for AI edge chips". In: *ISSCC*. 2020.
- [136] Xin Si et al. "A Local Computing Cell and 6T SRAM-Based Computing-in-Memory Macro With 8-b MAC Operation for Edge AI Chips". In: *JSSC* (2021).
- [137] Xin Si et al. "A twin-8T SRAM computation-in-memory unit-macro for multibit CNN-based AI edge processors". In: *JSSC* (2019).
- [138] Linghao Song et al. "GraphR: Accelerating graph processing using ReRAM". In: *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2018, pp. 531–543.
- [139] Linghao Song et al. "Pipelayer: A pipelined reram-based accelerator for deep learning". In: *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2017, pp. 541–552.

- [140] Linghao Song et al. “ReBNN: in-situ acceleration of binarized neural networks in ReRAM using complementary resistive cell”. In: *CCF Transactions on High Performance Computing* 1.3 (2019), pp. 196–208.
- [141] Baohua Sun et al. “MRAM co-designed processing-in-memory CNN accelerator for mobile and IoT applications”. In: *arXiv preprint arXiv:1811.12179* (2018).
- [142] Xiaoyu Sun et al. “XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks”. In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2018, pp. 1423–1428.
- [143] Shibin Tang et al. “AEPE: An area and power efficient RRAM crossbar-based accelerator for deep CNNs”. In: *2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*. IEEE. 2017, pp. 1–6.
- [144] Tianqi Tang et al. “Binary convolutional neural network on RRAM”. In: *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2017, pp. 782–787.
- [145] Yi Tay et al. “Sparse sinkhorn attention”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9438–9447.
- [146] Masayuki Terai et al. “Memory-State Dependence of Random Telegraph Noise of Ta2O5/TiO2 Stack ReRAM”. In: *IEEE Electron Device Letters* 31.11 (2010), pp. 1302–1304.
- [147] Hugo Touvron et al. “Training data-efficient image transformers & distillation through attention”. In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 10347–10357.
- [148] Florian Tramèr and Dan Boneh. “Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. 2019. URL: <https://openreview.net/forum?id=rJVorjCckQ>.
- [149] M Ueki et al. “Low-power embedded ReRAM technology for IoT applications”. In: *2015 Symposium on VLSI Technology (VLSI Technology)*. IEEE. 2015, T108–T109.
- [150] Kodai Ueyoshi et al. “DIANA: A n End-to-End Energy-Efficient Digital and Analog Hybrid Neural Network SoC”. In: *ISSCC*. 2022.
- [151] Hossein Valavi et al. “A 64-tile 2.4-Mb in-memory-computing CNN accelerator employing charge-domain compute”. In: *IEEE Journal of Solid-State Circuits* 54.6 (2019), pp. 1789–1799.

- [152] Alex Wang et al. "GLUE: A multi-task benchmark and analysis platform for natural language understanding". In: *arXiv preprint arXiv:1804.07461* (2018).
- [153] Hanrui Wang, Zhekai Zhang, and Song Han. "Spatten: Efficient sparse attention architecture with cascade token and head pruning". In: *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 97–110.
- [154] Wenhai Wang et al. "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 568–578.
- [155] Yitu Wang et al. "Reboc: Accelerating block-circulant neural networks in reram". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1472–1477.
- [156] Yitu Wang et al. "Rerec: In-reram acceleration with access-aware mapping for personalized recommendation". In: *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.
- [157] Yu Wang et al. "Energy efficient RRAM spiking neural network for real time classification". In: *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*. ACM, 2015, pp. 189–194.
- [158] Zongwei Wang et al. "Modulation of nonlinear resistive switching behavior of a TaOx-based resistive device through interface engineering". In: *Nanotechnology* (2016).
- [159] Zongwei Wang et al. "Self-activation neural network based on self-selective memory device with rectified multilevel states". In: *TED* (2020).
- [160] Mark Wilkening et al. "RecSSD: near data processing for solid state drive based recommendation inference". In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2021, pp. 717–729.
- [161] Ming-Chi Wu et al. "A study on low-power, nanosecond operation and multilevel bipolar resistance switching in Ti/ZrO<sub>2</sub>/Pt nonvolatile memory with 1T1R architecture". In: *Semiconductor Science and Technology* 27.6 (2012), p. 065010.
- [162] Ping-Chun Wu et al. "A 28nm 1Mb Time-Domain Computing-in-Memory 6T-SRAM Macro with a 6.6 ns Latency, 1241GOPS and 37.01 TOPS/W for 8b-MAC Operations for Edge-AI Devices". In: *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 65. IEEE, 2022, pp. 1–3.

- [163] Lixue Xia et al. "MNSIM: Simulation platform for memristor-based neuromorphic computing system". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.5 (2017), pp. 1009–1022.
- [164] Lixue Xia et al. "Stuck-at fault tolerance in RRAM computing systems". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8.1 (2017), pp. 102–115.
- [165] Lixue Xia et al. "Technological exploration of RRAM crossbar array for matrix-vector multiplication". In: *Journal of Computer Science and Technology* 31.1 (2016), pp. 3–19.
- [166] Peichen Xie, Bingzhe Wu, and Guangyu Sun. "BAYHENN: Combining Bayesian Deep Learning and Homomorphic Encryption for Secure DNN Inference". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. 2019, pp. 4831–4837. DOI: 10.24963/ijcai.2019/671. URL: <https://doi.org/10.24963/ijcai.2019/671>.
- [167] Qizhe Xie et al. "Large-scale cloze test dataset created by teachers". In: *arXiv preprint arXiv:1711.03225* (2017).
- [168] Cong Xu et al. "Overcoming the challenges of crossbar resistive memory architectures". In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2015, pp. 476–488.
- [169] Cong Xu et al. "Understanding the trade-offs in multi-level cell ReRAM memory design". In: *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE. 2013, pp. 1–6.
- [170] Sheng Xu et al. "PIMSim: A flexible and detailed processing-in-memory simulator". In: *IEEE Computer Architecture Letters* 18.1 (2018), pp. 6–9.
- [171] Cheng-Xin Xue et al. "24.1 A 1Mb Multibit ReRAM Computing-In-Memory Macro with 14.6 ns Parallel MAC Computing Time for CNN Based AI Edge Processors". In: *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE. 2019, pp. 388–390.
- [172] Bonan Yan et al. "A 1.041-Mb/mm<sup>2</sup> 27.38-TOPS/W Signed-INT8 Dynamic-Logic-Based ADC-less SRAM Compute-in-Memory Macro in 28nm with Reconfigurable Bitwise Operation for AI and Embedded Applications". In: *ISSCC*. 2022.
- [173] Bonan Yan et al. "A neuromorphic ASIC design using one-selector-one-memristor crossbar". In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2016, pp. 1390–1393.

- [174] Bonan Yan et al. "RRAM-based spiking nonvolatile computing-in-memory processing engine with precision-configurable in situ nonlinear activation". In: *2019 Symposium on VLSI Technology*. IEEE. 2019, T86–T87.
- [175] J Joshua Yang, Dmitri B Strukov, and Duncan R Stewart. "Memristive devices for computing". In: *Nature nanotechnology* 8.1 (2013), pp. 13–24.
- [176] Tzu-Hsien Yang et al. "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks". In: *Proceedings of the 46th International Symposium on Computer Architecture*. 2019, pp. 236–249.
- [177] Xiaoxuan Yang et al. "Improving the Robustness and Efficiency of PIM-based Architecture by SW/HW Co-design". In: *ASP-DAC*. 2023.
- [178] Xiaoxuan Yang et al. "ReTransformer: ReRAM-based processing-in-memory architecture for transformer acceleration". In: *Proceedings of the 39th International Conference on Computer-Aided Design*. 2020, pp. 1–9.
- [179] Jean Yang-Scharlotta et al. "Reliability characterization of a commercial TaO x-based ReRAM". In: *2014 IEEE International Integrated Reliability Workshop Final Report (IIRW)*. IEEE. 2014, pp. 131–134.
- [180] Peng Yao et al. "The effect of variation on neuromorphic network based on 1T1R memristor array". In: *2015 15th Non-Volatile Memory Technology Symposium (NVMTS)*. IEEE. 2015, pp. 1–3.
- [181] Jong-Hyeok Yoon et al. "29.1 A 40nm 64Kb 56.67 TOPS/W read-disturb-tolerant compute-in-memory/digital RRAM macro with active-feedback-based read and in-situ write verification". In: *2021 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 64. IEEE. 2021, pp. 404–406.
- [182] Shimeng Yu, Ximeng Guan, and H-S Philip Wong. "Understanding metal oxide RRAM current overshoot and reliability using kinetic Monte Carlo simulation". In: *2012 International Electron Devices Meeting*. IEEE. 2012, pp. 26–1.
- [183] Manzil Zaheer et al. "Big bird: Transformers for longer sequences". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 17283–17297.
- [184] Chiyuan Zhang et al. "Understanding deep learning requires rethinking generalization". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017. URL: <https://openreview.net/forum?id=Sy8gdB9xx>.
- [185] Grace Li Zhang et al. "Reliable and Robust RRAM-based Neuromorphic Computing". In: *Proceedings of the 2020 on Great Lakes Symposium on VLSI*. 2020, pp. 33–38.

- [186] Shijin Zhang et al. "Cambricon-X: An accelerator for sparse neural networks". In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2016, pp. 1–12. DOI: 10.1109/MICRO.2016.7783723.
- [187] Qilin Zheng et al. "Artificial neural network based on doped HfO<sub>2</sub> ferroelectric capacitors with multilevel characteristics". In: *EDL* (2019).
- [188] Qilin Zheng et al. "Enhance the robustness to time dependent variability of ReRAM-based neuromorphic computing systems with regularization and 2R synapse". In: *ISCAS*. 2019.
- [189] Qilin Zheng et al. "Lattice: an adc/dac-less reram-based processing-in-memory architecture for accelerating deep convolution neural networks". In: *DAC*. 2020.
- [190] Qilin Zheng et al. "PIMulator-NN: An Event-Driven, Cross-Level Simulation Framework for Processing-In-Memory-Based Neural Network Accelerators". In: *TCAD* (2022).
- [191] Qilin Zheng et al. "Accelerating Sparse Attention with a Reconfigurable Non-volatile Processing-In-Memory Architecture". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2023, pp. 1–6.
- [192] Qilin Zheng et al. "MobiLattice: a depth-wise DCNN accelerator with hybrid digital/analog nonvolatile processing-in-memory block". In: *Proceedings of the 39th International Conference on Computer-Aided Design*. 2020, pp. 1–9.
- [193] Qilin Zheng et al. "Mobilattice: A depth-wise dcnn accelerator with hybrid digital/analog nonvolatile processing-in-memory block". In: *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE. 2020, pp. 1–9.
- [194] Jiantao Zhou, Kuk-Hwan Kim, and Wei Lu. "Crossbar RRAM arrays: Selector device requirements during read operation". In: *IEEE Transactions on Electron Devices* 61.5 (2014), pp. 1369–1376.
- [195] Zhenhua Zhu et al. "A Configurable Multi-Precision CNN Computing Framework Based on Single Bit RRAM". In: *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM. 2019, p. 56.
- [196] Mohammed A Zidan et al. "A general memristor-based partial differential equation solver". In: *Nature Electronics* 1.7 (2018), pp. 411–420.

## **Biography**

Qilin Zheng received the B.S. degree from Peking University, Beijing, China, in 2019, and the M.S. degree from KU Leuven, Leuven, Belgium, in 2022. He is currently pursuing the Ph.D. degree in ECE with Duke University, Durham, NC, USA. His current research interests include computer architecture, non-volatile memory, and compute-in-memory design.