

# Design Techniques for Energy-Efficient and Scalable Machine Learning Accelerators

by

Teyuh Chou


A dissertation submitted in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
(Electrical and Computer Engineering)  
in The University of Michigan  
2022

Doctoral Committee:

Professor Zhengya Zhang, Chair  
Professor Michael Flynn  
Professor Wei Lu  
Professor Trevor Mudge

Teyuh Chou

teyuh@umich.edu

 ORCID iD 0000-0001-7033-336X

© Teyuh Chou 2022

All Rights Reserved

To my family, friends, and coworkers

# TABLE OF CONTENTS

<b>DEDICATION</b> . . . . .	ii
<b>LIST OF FIGURES</b> . . . . .	v
<b>LIST OF TABLES</b> . . . . .	viii
<b>ABSTRACT</b> . . . . .	ix
<b>CHAPTER</b>	
<b>I. Introduction</b> . . . . .	1
1.1 Deep Neural Networks . . . . .	1
1.2 Perception Processing . . . . .	3
1.3 Chiplet Technology . . . . .	4
1.4 Processing In Memory . . . . .	5
1.5 Thesis Outline . . . . .	6
<b>II. NetFlex: Flexible Chiplet-Based System for Perception Nets</b> . . . . .	7
2.1 Introduction . . . . .	7
2.2 Background . . . . .	9
2.2.1 DNN-Based SfM Learner . . . . .	9
2.2.2 Adaptive Layer Skipping for Easy Scenes . . . . .	12
2.2.3 Hardware Implementation Challenges . . . . .	12
2.3 NetFlex Architecture . . . . .	13
2.3.1 Accelerator Architecture . . . . .	13
2.3.2 System Architecture . . . . .	21
2.4 Chip Measurement . . . . .	23
2.5 System Evaluation of 4-Chiplet Package . . . . .	26
2.6 Summary . . . . .	28
<b>III. AR-PIM: PIM with Adaptive Range</b> . . . . .	30



3.1	Introduction . . . . .	30
3.2	SRAM-Based PIM . . . . .	32
3.3	7nm SRAM PIM Design Considerations . . . . .	35
3.3.1	Input Encoding . . . . .	35
3.3.2	BL Resolution . . . . .	36
3.3.3	ADC Resolution and ADC Sharing . . . . .	37
3.3.4	Necessity to Control Resolution . . . . .	39
3.4	The AR-PIM Architecture . . . . .	39
3.4.1	Runtime Density Detection . . . . .	40
3.4.2	Energy Minimization . . . . .	42
3.5	Evaluation . . . . .	44
3.5.1	Energy Consumption and Performance . . . . .	44
3.5.2	Discussion . . . . .	46
3.6	Summary . . . . .	47
 <b>IV. CASCADE: Chaining PIM in Analog Dataflow . . . . .</b>		<b>50</b>
4.1	Introduction . . . . .	50
4.2	RRAM-Based PIM . . . . .	54
4.2.1	Workloads and Mapping to RRAM . . . . .	55
4.2.2	In-RRAM Computation . . . . .	59
4.2.3	A/D Conversion for In-RRAM Computation . . . . .	60
4.3	The CASCADE Architecture . . . . .	61
4.3.1	Input Streaming and Weight Mapping in MAC RRAM . . . . .	63
4.3.2	Buffering of Partial Sums in Buffer RRAM . . . . .	63
4.3.3	TIA Interface between MAC RRAM and Buffer RRAM . . . . .	68
4.3.4	Final Accumulation and A/D Conversion . . . . .	69
4.3.5	Noise Tolerance . . . . .	71
4.4	Evaluation . . . . .	71
4.4.1	Methodology . . . . .	73
4.4.2	CASCADE Design Space Exploration . . . . .	75
4.4.3	Performance and Energy Consumption . . . . .	76
4.4.4	Extension to Spiking Neural Networks . . . . .	78
4.5	Summary . . . . .	78
 <b>V. Conclusion . . . . .</b>		<b>79</b>
 <b>BIBLIOGRAPHY . . . . .</b>		<b>81</b>

## LIST OF FIGURES

### Figure

1.1	Complexity represented by circles and operations of DNN models [1].	2
1.2	The performance gap between processor and memory [2]. . . . .	5
2.1	The NetFlex block diagram. . . . .	13
2.2	The convolver design. The zoom in view on the left shows the adder tree design. The zoom in view on the right shows the convolution operation over cycles. . . . .	14
2.3	A convolution operation example of the convolver over cycles for the $3 \times 3$ kernel with the stride 1. . . . .	15
2.4	A convolution operation example of the convolver over cycles for the $3 \times 3$ kernel with the stride 2. . . . .	15
2.5	A deconvolution operation example of the convolver over cycles for the $3 \times 3$ kernel. . . . .	16
2.6	A re-organized deconvolution operation example of the convolver over cycles for the $3 \times 3$ kernel. . . . .	17
2.7	The star topology of the core accelerator. . . . .	17
2.8	IA memory mapping. . . . .	19
2.9	Weight memory mapping. . . . .	19
2.10	The OA order from the convolver output. . . . .	20
2.11	The 4 chiplet streaming topology in the normal mode. . . . .	22
2.12	The 4 chiplet streaming topology in the bypass mode. . . . .	22
2.13	Die photo. . . . .	23
2.14	Frequency and power consumption with the voltage scaling. . . . .	24
2.15	Package photo of the 4-chiplet system. . . . .	25
2.16	A face to back connection of 4 chiplets. . . . .	26
2.17	Energy consumption of SfM depth net. . . . .	28
3.1	The PIM mapping of the convolution operation where $R \times S$ is the kernel size, $C$ is the input channel and $K$ is the output channel. The 2b weight in the example is stored in two columns. The zoom-in view shows the 6T SRAM and current $I_{DS}$ discharged by each bitcell. Both baseline PIM and AR-PIM adopt this mapping for convolution operation. . . . .	33

3.2	The multi-bit (2b in the example) input representation of (a) WL pulse level and (b) WL pulse train. . . . .	34
3.3	ADC energy consumption with effective number of bits (ENOB) from [3]. . . . .	35
3.4	Energy efficiency of 1b, 2b, 4b, 8b input activation. The different input encoding for each configuration is implemented by DAC bits and DAC cycles with the corresponding ADC resolution. . . . .	36
3.5	SRAM BL current levels from SPICE simulation for WL voltage of (a) 0.8V and (b) 0.6V. Each Gaussian distribution represents one output level. The less overlap between two distributions means the better sensing margin for distinguishing two output levels. . . . .	37
3.6	The energy efficiency comparison for synthesized digital design and PIM design with different BL resolutions and 3 Input Activation (IA) and Weight (W) bitwidth combinations. . . . .	38
3.7	Area comparison between synthesized digital design and PIM design for (a) 1 ADC per BL and (b) 1 ADC shared by 2 BLs. . . . .	38
3.8	AR-PIM architecture with BL detection circuitry. . . . .	40
3.9	Example of runtime density detection and adaptation. . . . .	41
3.10	Energy of AR-PIM for IA/W density from 5% to 95% with conditions of ADC ENOB r-1, r, r+1 (top) and ADC ENOB r for IA/W density from 5% to 95%. . . . .	42
3.11	Energy consumption and latency of AR-PIM compared and normalized to the baseline PIM with 7b ADC resolution (the rightmost bar in each figure) for each layer running (a)(b) LeNeT using MNIST dataset, (c)(d) AlexNet using ImageNet dataset, and (e)(f) VGG11 using ImageNet. . . . .	48
3.12	Latency overhead. . . . .	49
4.1	(a) A dot product implemented in an RRAM crossbar. (b) Parallel dot products implemented in an RRAM crossbar array. The input vector is converted to voltage pulses by DACs and the weights are stored in the RRAM crossbar. The BL currents are sampled and held (S&H) and converted to digital values. . . . .	54
4.2	Mapping of convolution operation on an RRAM crossbar array. . . . .	56
4.3	Illustration of the CASCADE architecture. The bold lines indicate the proposed dataflow. The zoom-in view shows an analog processing unit (APU). . . . .	62
4.4	Comparison of (a) in-RRAM MAC and digital accumulation of partial sums, and (b) in-RRAM MAC and in-RRAM buffering and accumulation of partial sums. The dashed lines indicate D/A and A/D boundaries. The inner loop is highlighted in blue. . . . .	65

4.5	Illustration of R-Mapping for 4-bit weights and 4-bit inputs. The inputs are serially streamed in with LSB first. The MAC RRAM stores two sets of 4-bit weights $W_a$ and $W_b$ . The arrows indicate the align and store of partial sums through TIAs and input drivers. The partial sum $P_{i,j}$ is stored in the $i$ th row and $j$ th column in cycle $i$ of the buffer RRAM. . . . .	66
4.6	Normalized energy consumption of digital partial-sum accumulation and analog in-RRAM partial-sum buffering and accumulation. . . .	67
4.7	(a) Schematic of TIA for converting an input current to a proportional output voltage; (b) simulation waveforms of the TIA designed in a 65nm CMOS technology and the charge leakage of the output capacitor over 25ns of the buffer RRAM write period; and (c) zoom-out view of the TIA sensing and transfer phase. . . . .	69
4.8	(a) Illustration of the final accumulation; and (b) schematic of a summing amplifier. . . . .	70
4.9	(a) Two-layer MLP classification accuracy for different configurations noted by $[b_{WL}, b_{cell}, N_{rows}, b_{BL}]$ , and (b) AlexNet top-5 classification accuracy for CASCADE and configurations based on ISAAC and PipeLayer. . . . .	72
4.10	Computation density across the design space. A notation, e.g., H64-T32-A7 R80, represents 80 $64 \times 64$ RRAM arrays with 32 TIAs per array and 7 ADCs shared by the arrays. . . . .	74
4.11	(a) Energy consumption of CASCADE compared to reference architectures running DNN and RNN benchmarks; (b) Throughput of CASCADE compared to reference architectures running DNN and RNN benchmarks. . . . .	76
4.12	Energy breakdown of CASCADE and two reference architectures. . .	77

## LIST OF TABLES

### Table

2.1	The detailed configuration of depth net. . . . .	11
2.2	The detailed configuration of pose net. . . . .	12
2.3	Measurement results for various configurations. . . . .	24
2.4	Comparison with prior works for perception. . . . .	25
2.5	Microarchitecture parameters of NetFlex. The package-level results are from [4] . . . . .	27
2.6	Comparison of chiplet-based prototypes. . . . .	27
3.1	Mean and standard deviation of BL density numbers. (The maximum BL density is 128 for a $128 \times 128$ array.) . . . . .	43
4.1	In-RRAM MAC architecture comparison. . . . .	59
4.2	Configurations of the CASCADE, ADC-based and SA-based reference architectures. . . . .	73
4.3	Benchmarks for evaluation. Convolution layers are denoted as $R \times S, K/D (L)$ , where $R, S$ and $K$ correspond to the notations used in Figure 4.2, and $L$ denotes the number of such layers. . . . .	74

## ABSTRACT

Machine learning is a key application driver of new computing hardware. Designing high-performance machine learning hardware requires a large number of operations and a high memory bandwidth. The energy efficiency of the hardware is often limited by the data movement and the memory access bottleneck. As the machine learning models evolve to become even larger and more complex over time, it is also a constant challenge to meet the computational requirements of these new models. In this work, we investigate processing in memory (PIM) approaches to overcome the memory access bottleneck and a chiplet-based integration approach to efficiently scale up machine learning hardware by reusing chiplets.

DNN model size and complexity growths have already outpaced the DNN chip upgrades. Making monolithic chips to keep up with the model evaluations is challenging. We demonstrate a chiplet-based approach to designing DNN hardware. The proposed chiplet is called NetFlex – a modular design that can be connected together to build larger DNN hardware. NetFlex is designed to support various layers, including convolutional layer, deconvolutional layer, and fully connected layer, and multiple configurations. The deconvolution dataflow is optimized by removing computation of both row-wise and element-wise 0s. In the PE arrays, spatial processing with three dimensional parallelism is implemented for data reuse and temporal processing is chosen to adapt to different kernel sizes. The processing scheduling and memory mapping are designed for streaming activations without extra data rearrangement. The chiplets are connected to form a ring topology and can be gated by the skipping module to reduce the computation and memory accesses of simple scenes for

perception. An Advanced Interface Bus (AIB) and an Advanced eXtensible Interface (AXI)-compatible protocol enable data streaming from one chiplet to another chiplet. The chiplets are integrated on the interposer using a 2.5D fan-out wafer level packaging (FOWLP) technology. NetFlex is compared with recent silicon prototypes for perception application. The multi-chiplet system is evaluated and compared with recent chiplet-based designs. The results shows NetFlex achieves a higher throughput with competitive energy and area efficiency and can potentially expand to larger-scale systems.

PIM approach has gained significant attention due to its potential of high energy efficiency for DNN workloads. However, key challenges remain: the overhead of high-resolution ADCs and degraded sensing margin when a large number of bitcells are activated together. We propose adaptive-range PIM (AR-PIM) to take advantage of sparsity to relax the need for high-resolution ADCs and improve the sensing margin. The evaluations, performed using a commercial 7nm FinFET PDK, show that AR-PIM increases the energy efficiency by  $1.7\times$  and reduces the area by  $4.3\times$  over a baseline PIM architecture for DNN workloads while maintaining the inference accuracy.

PIM is a concept to enable massively parallel dot products while keeping one set of operands in memory. PIM is ideal for computationally demanding deep neural networks (DNNs) and recurrent neural networks (RNNs). Processing in resistive RAM (RRAM) is particularly appealing due to RRAM’s high density and low energy. A key limitation of PIM is the cost of multi-bit analog-to-digital (A/D) conversions that can defeat the efficiency and performance benefits of PIM. We demonstrate the CASCADE architecture that connects multiply-accumulate (MAC) RRAM arrays with buffer RRAM arrays to extend the processing in analog and in memory: dot products are followed by partial-sum buffering and accumulation to implement a complete DNN or RNN layer. Design choices are made and the interface is designed to

enable a variation-tolerant, robust analog dataflow. A new memory mapping scheme named R-Mapping is devised to enable the in-RRAM accumulation of partial sums; and an analog summation scheme is used to reduce the number of A/D conversions required to obtain the final sum. CASCADE is compared with recent in-RRAM computation architectures using state-of-the-art DNN and RNN benchmarks. The results demonstrate that CASCADE improves the energy efficiency by  $3.5\times$  while maintaining a competitive throughput.

These three parts, NetFlex, AR-PIM and CASCADE demonstrate more energy-efficient and scalable solutions for future machine learning hardware designs.



# CHAPTER I

## Introduction

Machine learning algorithms demonstrate remarkable successes in recent years and open a new era of artificial intelligence (AI). The supervised learning is conducted with the labeled inputs and the unsupervised learning for non-labeled inputs. The advances in hardware such as GPUs to deal with the large amount of computations boost the growth of the field. The other hardware options including CPUs, FPGAs, and ASICs are also proposed for different use cases. However, key bottlenecks and challenges still persist. One is the memory wall bottleneck that limits the performance and processing efficiency improvement. The other is the scalability of the current hardware in handling fast evolving model of larger sizes as shown in Figure 1.1.

### 1.1 Deep Neural Networks

Neural networks are one subset of machine learning algorithms, and they have evolved to be one primary workload for new computing hardware. A neural network layer consists of the input layer, the hidden layers, and the output layer. In a fully connected layer, the neurons of one layer are fully connected to all neurons of previous layer. The weights of each layer are trained using gradient descent. The error of the last layer from forward propagation and the label is required for backpropagation during training. The weights can be used to predict the classification class of the

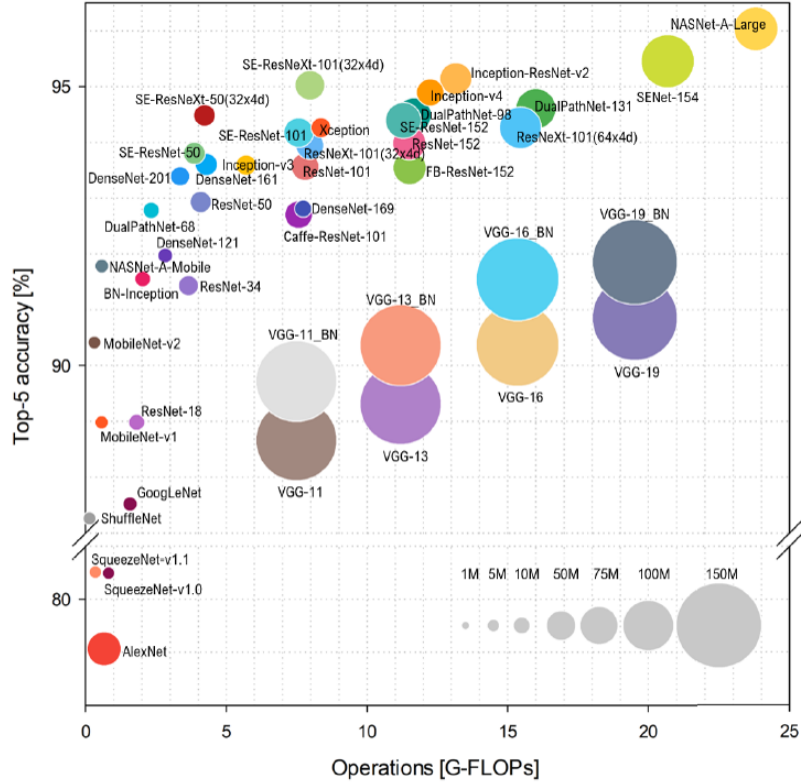


Figure 1.1: Complexity represented by circles and operations of DNN models [1].

unseen data during inference.

A convolutional layer can be represented in 3D volumes of neurons in 3 dimensions, width, height, and depth. Only a small number of neurons in one layer are connected to the layer before it. Convolutional neural networks (CNNs), and generally deep neural networks (DNNs), typically have convolution layers, ReLU layers which apply the non-linearity, pooling layers, and fully-connected layer. DNN computation consists of a large number of matrix multiplication operations with plenty of opportunities to compute in parallel.

Data compression, data sparsity, and quantization are well-known approaches to reduce the computation cost and alleviate the memory bottleneck in the DNN hardware. Processing in memory (PIM) is another approach that promises to remove the memory bottleneck entirely, but PIM has been restricted to small research prototypes and practical PIM systems are often limited by the interface circuitry that performs

the analog to digital conversion.

To address the scalability challenge, larger chips are being built every year in newer process nodes to handle new generations of neural network models, but the design effort and cost are also increasing rapidly. GPU, CPU and FPGA are more flexible platforms, but their energy efficiency and normalized cost are out-of-reach for many practical applications. Recently, chiplet-based systems [5, 6] have started to emerge. The chiplet technology offers a path to more easily scale up hardware to support newer and larger models, but the a modular chiplet and a standard interface are required.

## 1.2 Perception Processing

Autonomous systems have gained lots of attention in recent years including unmanned aerial vehicles (UAV), autonomous driving cars, and intelligent robots etc. An autonomous system consists of sensing, navigation, planning, and control [7]. The input data from the camera, inertial sensors, LiDAR, RADAR, or other sensors are sent to the navigation for localization and object detection and tracking for scene understanding. The planning is the decision making for motion planning, obstacle avoidance, etc. The control is responsible for giving commands to the actuator for acceleration, braking, or steering.

Ego-motion estimation is an important task for a vehicle to know its 3D motion and location within the environment and used for augmented reality (AR) and virtual reality (VR) [8]. The simultaneous localization and mapping (SLAM) algorithms [9, 10] is used to build the map and estimate the location for vehicles/robots.

More DNN based perception [11, 12, 13] and hybrid DNN and SLAM perception [14] algorithms have come up in recently years. The advantage of the DNN-based perception is the use of training to adapt to different environments and sensors, replacing the large number of tuning parameters in the SLAM algorithms.

The perception task involves a large amount of data processing from many sensors and the map is also growing. These can present computation bottlenecks. The other challenge for mobile robots is the limited power budget.

### 1.3 Chiplet Technology

With increasing cost of advanced technologies, modular chips become one of the promising candidates to address the challenge in semiconductor scaling. Compared to a large-scale monolithic system, a 2.5D system can provide the flexibility in package level and a new set of masks is not required for changing a subset of the circuit functions. It can provide integration with different technology nodes for each component such as digital component, memory component, analog component. The place and route time can be manageable and better isolation between components can be obtained.

Several technology leaders have recently demonstrated advanced 2.5D or system-on-chip (SoC) package solutions with a physical layer (PHY) interface to enable heterogeneous integration. Intel demonstrated Embedded Multi-die Interconnect Bridge (EMIB) that utilizes a thin silicon bridge with multi-layer interconnects embedded in a substrate. TSMC demonstrated Chip-on-Wafer-on-Substrate (CoWoS) and the chiplets are integrated using silicon interposer and through silicon vias (TSVs) to connect  $\mu$ bumps and C4 bumps [15]. Chiplets on top of the silicon interposer are connected using  $\mu$ bumps and the package below silicon interposer is connected using C4 bumps. The Low-voltage-In-Package-INterCONnect (LINPINCON) was introduced as the PHY interface. NVIDIA implemented an organic substrate with ground-referenced signaling (GRS) links for chip-to-chip communication [5]. AMD's Zeppelin implemented an organic substrate with serializer-deserializer (SerDes) links between chips. The infinity fabric (IF) is used as the PHY interface [16].

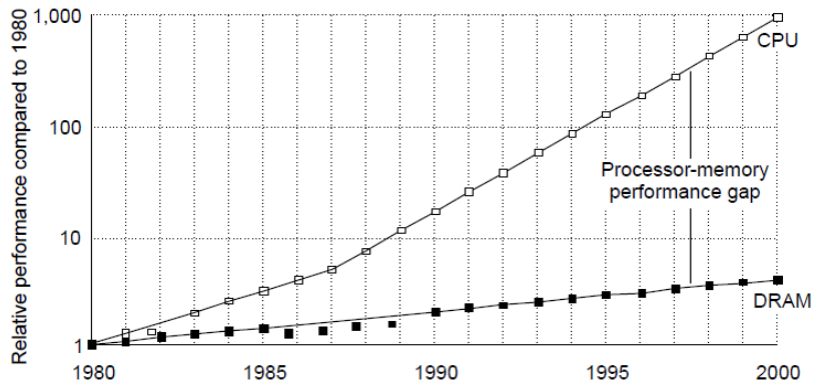


Figure 1.2: The performance gap between processor and memory [2].

## 1.4 Processing In Memory

With the semiconductor scaling, processor speed improves with Moore’s Law. However, memory speed improves around 7% per year which is less than processor speed of around 55% per year as shown in Figure 1.2. This creates the processor memory performance gap and the gap grows around 50% per year. The memory latency would become a huge bottleneck in computing performance.

Conventional von Neumann architectures start to gain less efficiency as the processor frequency increases because the processing unit and memory unit share the same common bus of limited bandwidth. Data access is a bottleneck for data-centric applications, and data movements from off-chip to on-chip consume considerable energy.

PIM was proposed to address the limited communication bandwidth between processing unit and memory unit [17, 18, 19]. The idea is to perform computation in or near the data. PIM enables the memory unit to compute to avoid the large amount of data movements and improve the energy efficiency.

PIM utilizes the analog property to perform matrix-vector or matrix-matrix operation that is massively parallel. Processing near memory uses 3D-stacked memory technology to increase the memory bandwidth and decrease the memory latency to the logic.

The choices of memory include non-volatile memory such as resistive random access memory (RRAM), magnetic random access memory (MRAM), phase-change memory (PCRAM), ferroelectric random access memory (FeRAM), flash and volatile memory such as static random access memory (SRAM) and dynamic random access memory (DRAM). Non-volatile memory provides data storage even when the power supply is turned off, a higher memory capacity, a lower cost per bit but slower speed compared to volatile memory in the memory hierarchy.

## 1.5 Thesis Outline

The thesis contributes to the development of energy-efficient PIM architectures for machine learning applications and scalable ASICs using 2.5D technology for autonomous navigation. The thesis presents three works as follows.

In Chapter II, the NetFlex chip is presented for perception processing in autonomous navigation applications. With 2.5D technology and Advanced Interface Bus (AIB), the chiplets can be used to form a streaming system for large DNN workloads to improve the throughput and support layer skipping to achieve a better energy efficiency.

In Chapter III, an SRAM-based PIM architecture is presented to utilize the runtime bit-level sparsity of data. The comparison between digital and analog NN hardware showcases the PIM design space and trade-offs. The adaptive resolution architecture with runtime density detection and threshold selection provide a higher energy efficiency.

In Chapter IV, an energy-efficient RRAM based PIM architecture is presented to reduce a large portion of the energy consumption of analog-to-digital conversion. The design employs a novel mapping method and summation scheme to support practical design choices for the DNN computation.

## CHAPTER II

# NetFlex: Flexible Chiplet-Based System for Perception Nets

### 2.1 Introduction

Autonomous navigation has risen in its importance because of the needs for robots, unmanned aerial vehicles (UAVs), and self-driving cars. Navigation involves the tasks of perception, motion planning, and control. The perception step is for robots or vehicles to localize themselves and understand the scene. The motion planning step is to find a path to the destination. The control step involves steering and accelerating. The computation bottleneck is perception [20] among these steps of autonomous navigation. In the past few years, most of the perception computations are executed on GPUs, CPUs, or FPGAs. Some recent work including [20, 21] implemented the perception processing on ASICs to further save the computation power.

There are two major categories of algorithms for perception tasks. One approach is based on Simultaneous Localization and Mapping (SLAM) [9, 10] and the other is based on Convolutional Neural Network (CNN) [11, 12, 13]. The combined SLAM and CNN approach is also proposed [14] to fuse the depth measurements from direct monocular SLAM and CNN-predicted dense depth maps to improve the accuracy in image locations. A SLAM-based approach can involve hundreds of configurable

parameters for adapting to various environments and sensors. The kernel operations include matrix operations, Cholesky factorization, etc. The intermediate operations also require high precision floating point arithmetic and complex finite state machine (FSM) [20]. With an end-to-end CNN-based approach, the need for manual synchronization and manual calibration between the camera and the inertial measurement unit (IMU) is eliminated. The kernel operations in a CNN-based approach include matrix operations, activation functions, pooling, etc. The number of parameters is reduced and the dataflow and operations are more regular and unified, making it hardware-friendly.

However, a CNN-based approach is compute-intensive. Mapping all convolutional layers of a practical network requires a large amount of memory and meeting the real-time performance requires a large number of processing arrays. These lead to building large chips that are costly in dollars and engineering effort. A chiplet-based approach [5, 6] was proposed to use multiple instances of a modular chip, called chiplets, to construct a large-scale system in a package to support applications needing large workloads.

A CNN-based approach for perception processing utilizes an encoder-decoder architecture with additional links between network layers. The encoder part extracts features and the decoder part reestablishes the original resolution by deconvolutional layers. The deconvolutional layers upsample to the input dimensions for depth estimation. The deconvolutional layers involve processing many inserted 0s, making existing CNN accelerators inefficient.

We propose NetFlex, a scalable design that utilizes chiplets to construct large-scale systems for perception processing, the computation-intensive part of a navigation system. We exploit the dataflow in CNN and connect the chiplets in a ring for streaming processing among layers to ensure a low latency and a high energy efficiency. Compared to a common mesh topology, a ring topology eliminates complex routers



and their associated overhead. Furthermore, chiplets can be gated by the skipping module to support layer skipping. To support the streaming processing, a process scheduling for activations is designed to remove extra data reorganization for the streaming input buffer. The streaming dataflow is designed to overlap input loading and computation for a lower latency and reduced data buffering.

In Section 2.2, we provide a background of Structure from Motion (SfM) Learner algorithm. In Section 2.3, we illustrate the hardware architecture design and the dataflow. In Section 2.4, we present the chip performance measurement and compare it with other related perception hardware designs. In Section 2.5, we present an evaluation of the performance of a 4-chiplet package and compare it with other chiplet-based systems. In Section 2.6, we conclude this work.

## 2.2 Background

### 2.2.1 DNN-Based SfM Learner

The SfM Learner framework [13] consists of 2 networks, depth net and pose net. The depth net estimates a pixel-wise depth map  $\hat{D}_t$  from the current image ( $I_t$ ). The pose net takes the current image (referred to as the target view) plus the neighboring images (referred to as nearby/source views) in the video sequence. That is, the input comprises the current image ( $I_t$ ) plus the previous ( $I_{t-1}$ ) and next ( $I_{t+1}$ ) image. The pose net infers the camera’s ego-motion from the video sequences, denoted as  $\hat{T}_{t \rightarrow t-1}$  and  $\hat{T}_{t \rightarrow t+1}$  respectively. The estimated pose along with the estimated depth map is then used to reconstruct a view of current time  $\hat{I}_t$  by warping nearby views. The depth net and the pose net are coupled by photometric reconstruction loss to jointly train the depth net and pose net. This view synthesis serves as supervision to learn the ego-motion and depth map without ground truth from the labeled image sequences. Although the networks are jointly trained, the networks can be executed

independently during inference.

**Depth Net.** The depth net is structured as an encoder-decoder with skip connections and multi-scale side predictions, which is adopted from the DispNet architecture [22]. The depth net only takes a single view at a time as input and generates an estimated depth map for each pixel as the output. A convolution layer with the stride of 2 in the encoder part is followed by a pairing convolution layer with the stride of 1, resulting in the same output size. The intermediate results are concatenated with deconvolutional layers in the decoder part to estimate the per-pixel depth maps in multiple scales. All convolutional layers except for the prediction layers are followed by ReLU activation. For the prediction layer, the activation of  $1/(10 \times \text{sigmoid}(x) + 0.1)$  is used to constrain the predicted depth to be always positive within a reasonable range. The detailed network configuration is summarized in Table 2.1.

**Pose Net.** The input of the pose net comprises multiple images in the video sequence. The current image and neighboring images are concatenated in the color (RGB) channels direction, which results in an input depth of  $3\times$  of the original depth. The outputs of the pose net are the estimates of relative poses of the current image and neighboring image, i.e.  $\hat{T}_{t \rightarrow t-2}$ ,  $\hat{T}_{t \rightarrow t-1}$ ,  $\hat{T}_{t \rightarrow t+1}$  and  $\hat{T}_{t \rightarrow t+2}$ . Here each relative pose of the source view consists of 3 Euler angles and 3-D translation, which results in an output depth of  $6 \times 4 = 24$ .

The pose net consists of 7 convolutional layers (cnv1 to cnv7) with the stride of 2, followed by a  $1 \times 1$  convolution (pose\_pred). Finally, predictions at all spatial locations are aggregated by applying global average pooling. All convolutional layers except for the last layer are followed by ReLU. For the last layer, no nonlinear activation is applied. The detailed network configuration is summarized in Table 2.2.

Layer	Kernal			Input			Output		
	Name	Size	Number	Stride	Size	Size	Channel	Size	Size
cnv1	7	32	2	416	128	3	208	64	32
cnv1b	7	32	1	208	64	32	208	64	32
cnv2	5	64	2	208	64	32	104	32	64
cnv2b	5	64	1	104	32	64	104	32	64
cnv3	3	128	2	104	32	64	52	16	128
cnv3b	3	128	1	52	16	128	52	16	128
cnv4	3	256	2	52	16	128	26	8	256
cnv4b	3	256	1	26	8	256	26	8	256
cnv5	3	512	2	26	8	256	13	4	512
cnv5b	3	512	1	13	4	512	13	4	512
cnv6	3	512	2	13	4	512	7	2	512
cnv6b	3	512	1	7	2	512	7	2	512
cnv7	3	512	2	7	2	512	4	1	512
cnv7b	3	512	1	4	1	512	4	1	512
upcnv7	3	512	2	4	1	512	8	2	512
icnv7	3	512	1	7	2	1024	7	2	512
upcnv6	3	512	2	7	2	512	14	4	512
icnv6	3	512	1	13	4	1024	13	4	512
upcnv5	3	256	2	13	4	512	26	8	256
icnv5	3	256	1	26	8	512	26	8	256
upcnv4	3	128	2	26	8	256	52	16	128
icnv4	3	128	1	52	16	256	52	16	128
disp4	3	1	1	52	16	128	52	16	1
upcnv3	3	64	2	52	16	128	104	32	64
icnv3	3	64	1	104	32	129	104	32	64
disp3	3	1	1	104	32	64	104	32	1
upcnv2	3	32	2	104	32	64	208	64	32
icnv2	3	32	1	208	64	65	208	64	32
disp2	3	1	1	208	64	32	208	64	1
upcnv1	3	16	2	208	64	32	416	128	16
icnv1	3	16	1	416	128	17	416	128	16
disp1	3	1	1	416	128	16	416	128	1

Table 2.1: The detailed configuration of depth net.

Layer	Kernel			Input			Output		
	Name	Size	Number	Stride	Size	Size	Channel	Size	Size
cnv1	7	16	2	416	128	15	208	64	16
cnv2	5	32	2	208	64	16	104	32	32
cnv3	3	64	2	104	32	32	52	16	64
cnv4	3	128	2	52	16	64	26	8	128
cnv5	3	256	2	26	8	128	13	4	256
cnv6	3	256	2	13	4	256	7	2	256
cnv7	3	256	2	7	2	256	4	1	256
pose_pred	1	24	1	4	1	256	4	1	24

Table 2.2: The detailed configuration of pose net.

### 2.2.2 Adaptive Layer Skipping for Easy Scenes

The middle layers of the encoder-decoder structure of the depth net are costly with a large number of weights. Several heavy layers in the middle of the network can be selectively skipped for easy scenes with fewer objects to save 12% of computation and 75% of memory access with negligible accuracy loss. The chiplet-based system can support the layer skipping of heavy layers in the middle of the network by a simple chiplet controller and the skipped chiplet can be power-gated to save energy in the proposed hardware.

### 2.2.3 Hardware Implementation Challenges

The perception task for mobile robots including UAVs and autonomous cars is battery powered. Hence, an energy-efficient hardware system is required. In addition, a flexible hardware is needed to process the pose and depth estimation, and a low computation latency is critical for fast reaction to changing environments.

Processing the NN layers of SfM Learner requires large amounts of computation and memory storage. The pose net and depth net contribute a total of 5G MAC operations. A hardware architecture needs to exploit multiple dimensions of parallelism, data reuse, and dataflow optimization to obtain low latency, high throughput, and energy efficiency. The pose net requires 3.05 MB and 2.32 MB of memory to

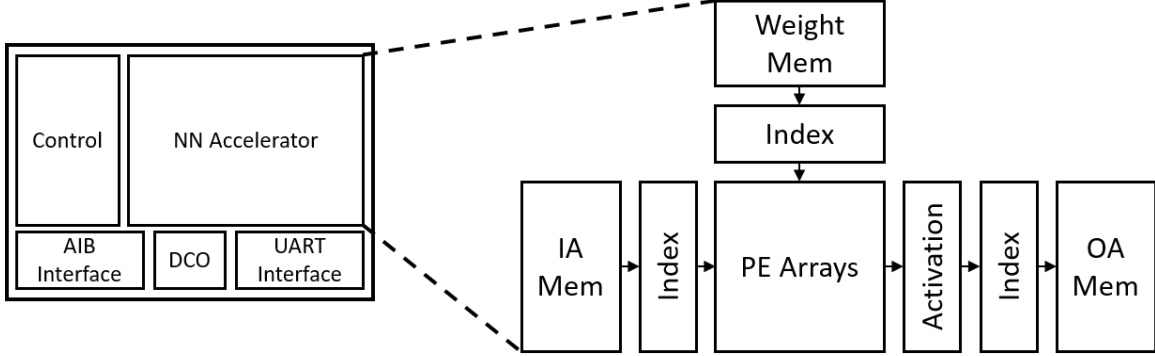


Figure 2.1: The NetFlex block diagram.

store the weights and inputs. The depth net require 60.25 MB and 13.06 MB of memory to store the weights and inputs. The large memory size makes it difficult to store all the weights using on-chip SRAMs. Even if a large chip size is allowed, the clock distribution and power delivery are still challenging across a large chip and can potentially introduce significant overhead and design time.

## 2.3 NetFlex Architecture

### 2.3.1 Accelerator Architecture

Figure 2.1 shows the NetFlex block diagram including NN accelerator, control for different dataflows of PEs according to different layer shapes, digitally controlled oscillator (DCO) for clock generation, UART interface, and AIB interface.<sup>1</sup>

**NN Accelerator.** Figure 2.2 shows the convolver design inside the NN accelerator. The convolver supports three dimensions of parallelism including convolution  $XY$  parallelism of 8, input channel  $C$  parallelism of 16, and output channel  $K$  parallelism of 8 for a total of 1024x parallelism. Each of the 1024 PEs is responsible for a MAC operation of a 16b input and a 16b weight. The input activations are shared among  $K$  PE arrays and each PE array stores its own weights and its corresponding output channel. Within a PE array, the weights are buffered and broadcast along

<sup>1</sup>The modules for data transfer including UART and AIB interface were implemented by Wei Tang and Chester Liu.

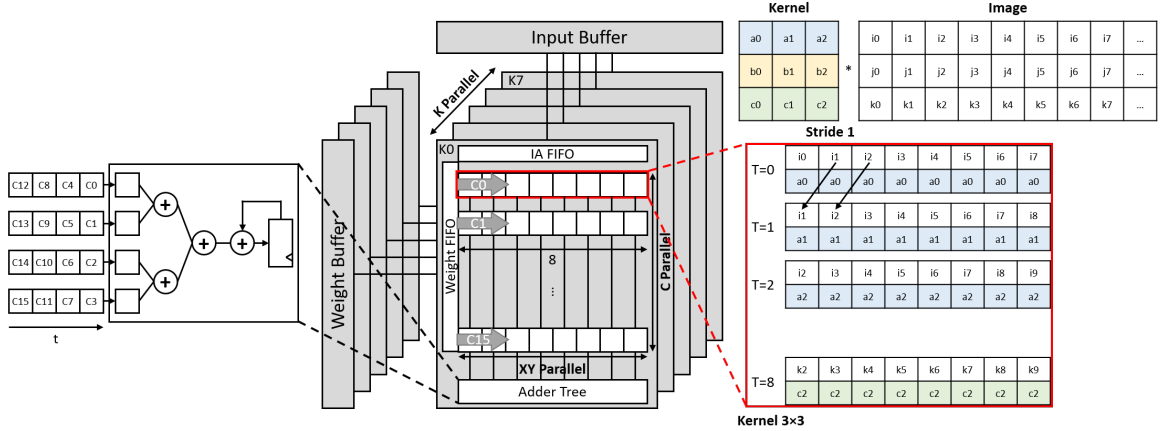


Figure 2.2: The convolver design. The zoom in view on the left shows the adder tree design. The zoom in view on the right shows the convolution operation over cycles.

rows, one per input channel. The convolution is processed temporally to adapt to kernel sizes of 3, 5, and 7 to maintain high utilization among different layers. Each convolution operation takes 9, 25, 49 cycles for kernel sizes 3, 5, and 7 respectively. The convolver can process 8 inputs in each row at the same time and the 8 inputs are left-shifted by shift registers for the next cycle. This forms the 1D convolution processing. The convolver then takes the 8 inputs of the subsequent row, followed by left-shifting, and so on. The horizontal and vertical shifting produce 2D convolution results. The shift registers enable the input activation reuse to save energy of memory accesses. Finally, all the dot product results of input activations and weights are collected and added up along a column of PEs for the input channel reduction. The channel reduction along a column of the PE array is done by a pipelined adder tree for each PE array. The pipelined adder tree sums 4 dot product results at a time and it takes 4 cycles to sum  $C = 16$  values.

**Convolution Layer.** Figure 2.3 shows the detailed convolution operation of the convolver over cycles for the  $3 \times 3$  kernel with the stride of 1. The  $CONV = 8$  convolution operations are processed at the same time. Figure 2.4 shows the case of the  $3 \times 3$  kernel with the stride of 2. In this case, the input left-shifted by shift registers as the case of the stride of 1 but only PEs in odd columns accumulate the

**Convolution: Stride 1, Kernel 3x3**

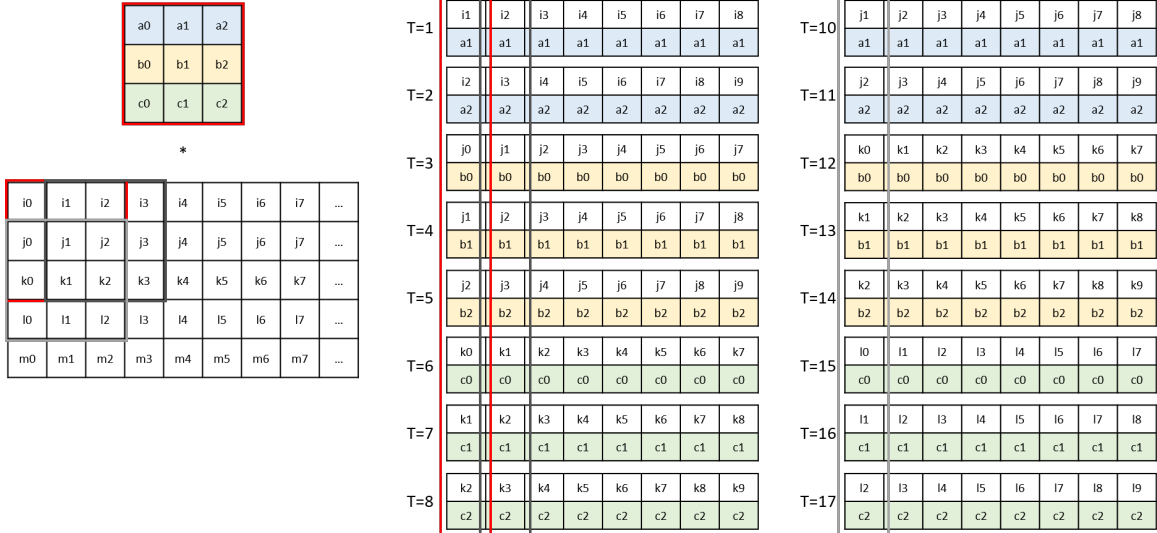


Figure 2.3: A convolution operation example of the convolver over cycles for the 3x3 kernel with the stride 1.

**Convolution: Stride 2, Kernel 3x3**

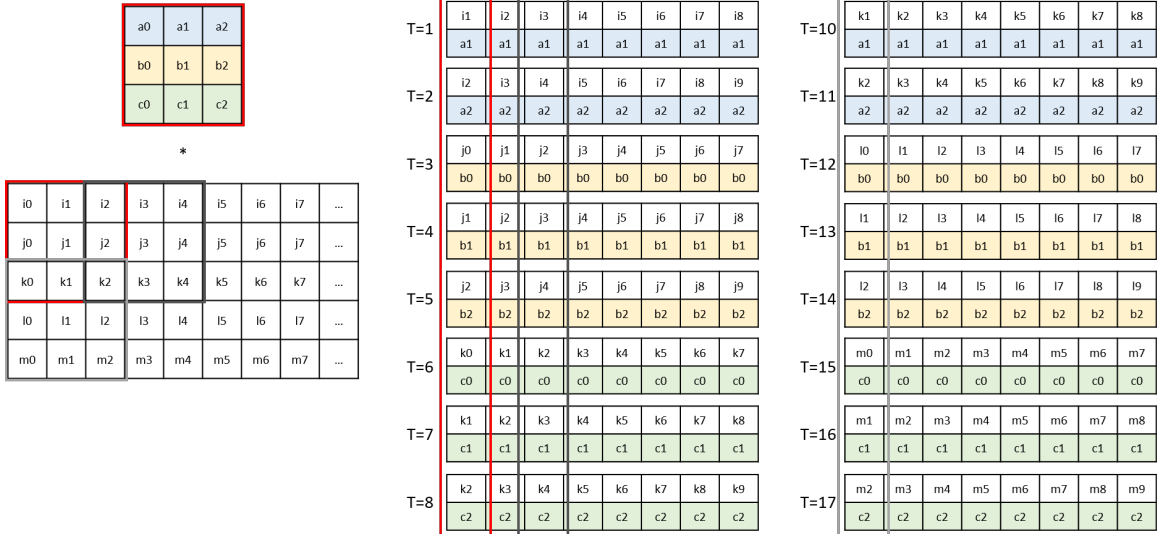


Figure 2.4: A convolution operation example of the convolver over cycles for the 3x3 kernel with the stride 2.

### Deconvolution: Stride 2, Kernel 3×3

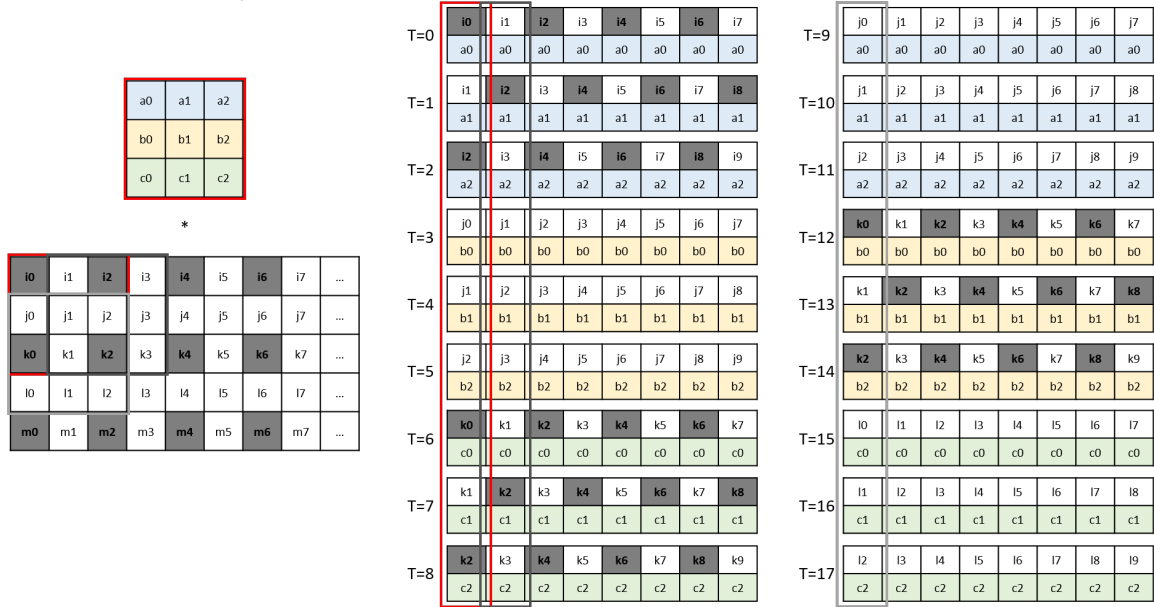


Figure 2.5: A deconvolution operation example of the convolver over cycles for the 3×3 kernel.

dot product results.

**Deconvolutional Layer.** The deconvolution operation is used extensively for semantic segmentation in DNNs. The well-known examples that use deconvolutional layers are fully-convolutional network (FCN) [23] and generative adversarial network (GAN) [24]. The deconvolution operation is also known as the transposed convolution. In the case of the deconvolution operation with the stride of 2, the input is inserted with a zero between each activation both horizontally and vertically. Processing involving 0s needs to be removed for energy saving and latency improvement. [25] exploit the property of deconvolution operation by reorganizing the flow of data to remove computation of input rows consisting of all 0s. However, there are still 0s in input rows that are not all 0s. NetFlex not only removes the row-wise 0s but also element-wise 0s for non-all-zero rows to accelerate the deconvolution operation. Figure 2.5 shows the detailed deconvolution operation of the convolver over cycles for the 3×3 kernel with the stride of 2. We reorganize the processing dataflow to squeeze out the 0 operands as shown in Figure 2.6.



**Deconvolution: Stride 2, Kernel 3x3**

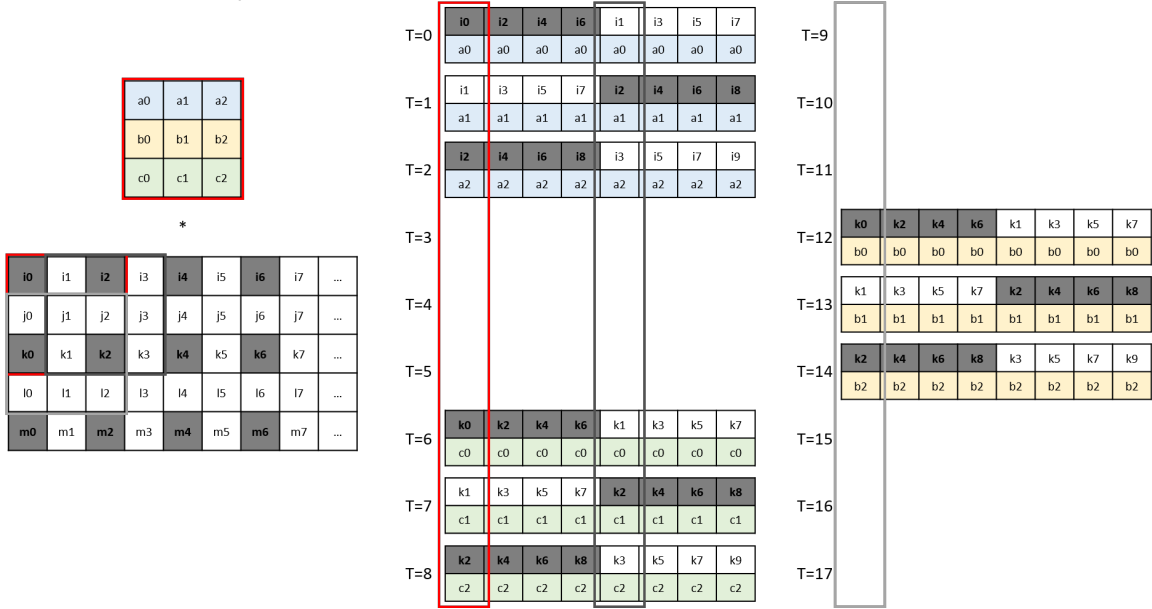


Figure 2.6: A re-organized deconvolution operation example of the convolver over cycles for the 3x3 kernel.

**Fully Connected Layer.** Fully connected layers require the accumulation of more dot-product results. The pipelined adder tree in each PE array for 16 elements addition can be reconfigured to connect to each other to form another adder tree hierarchy for 8-element addition. The design allows 8x16 additions than only 16 additions.

**Star Topology.** The NetFlex Chip uses a star topology with 8 leaf nodes for output channel parallelism as shown in Figure 2.7. The input memory module is

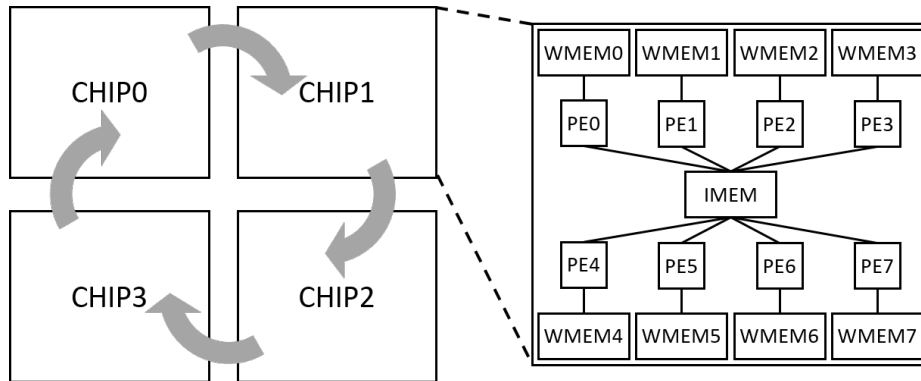


Figure 2.7: The star topology of the core accelerator.

located in the center since the input activations are shared among  $K = 8$  output channels. The input memory module is responsible for receiving the data from the AIB interface and then distributing the data to PE arrays. The weight memory module is located on each leaf of the star topology and stores the weights of each output channel  $K$ . The PEs are in between the input memory module and the weight memory module for receiving both input activations and weights. The star topology reduces the wire routing congestion. The natural sharing property of neural network layers and the star topology remove the need for arbiters for the shared memory modules.

**Memory Modules.** A two-port SRAM is used for the input memory module to support streaming mode. The input memory module receives activations from AIB and sends the activations to PEs. The input memory module is implemented as a first-in-first-out (FIFO) and it handles the data transfer with two pointers, one for read operation and the other for write operation. The read pointer and write pointer are used to check the FIFO condition of full and empty to prevent data from being overwritten before read. The weight memory module is composed of single-port SRAMs for reading weights. The output memory module is composed of single-port SRAMs for writing the output activations.

**Process Scheduling.** In order to support direct streaming, the input memory module acts as the line buffer and the depth-first processing is chosen for finishing an activation as fast as possible. Figure 2.8 illustrates the IA fetching schedule and IA memory mapping. If an NN layer has more than 16 input channels, the convolver follows the process scheduling of depth-first, then width, and finally the height of the input activations. Each row of the IMEM is 2048b and composed of the 8 width $\times$ 16 channels $\times$ 16b. Figure 2.9 illustrates the weight fetching schedule and weight memory mapping. The weights are accessed in the channel direction first, then width, then height, and finally the kernel number. Each row of the WMEM is 2048b and composed

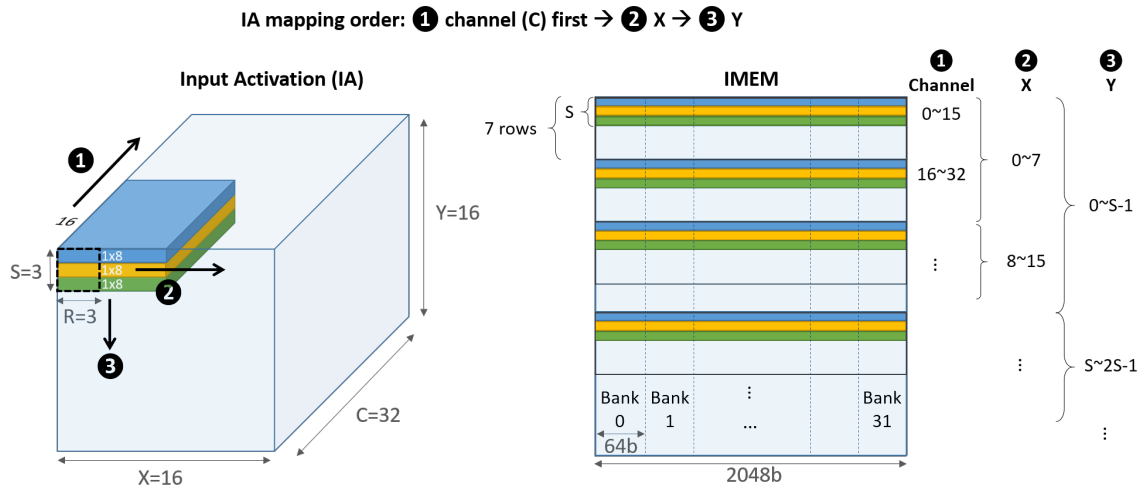


Figure 2.8: IA memory mapping.

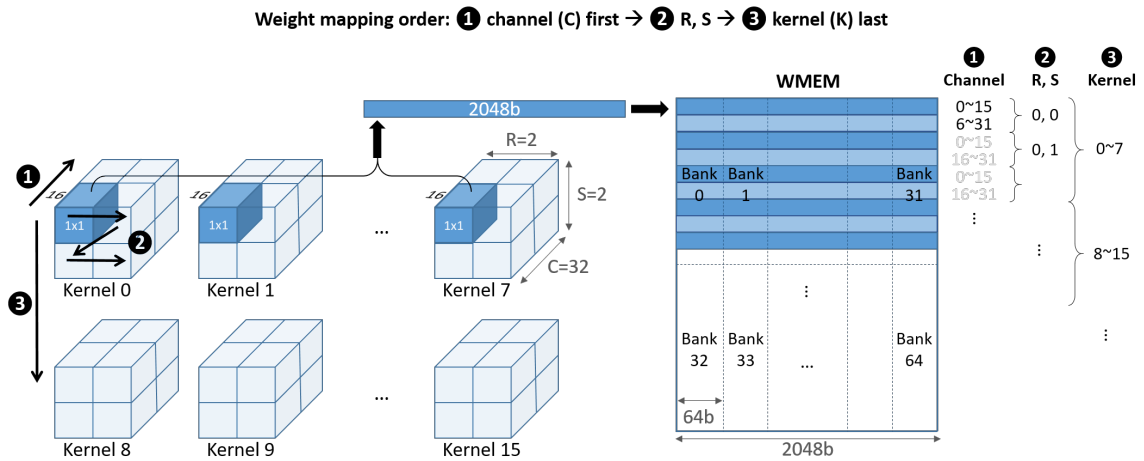
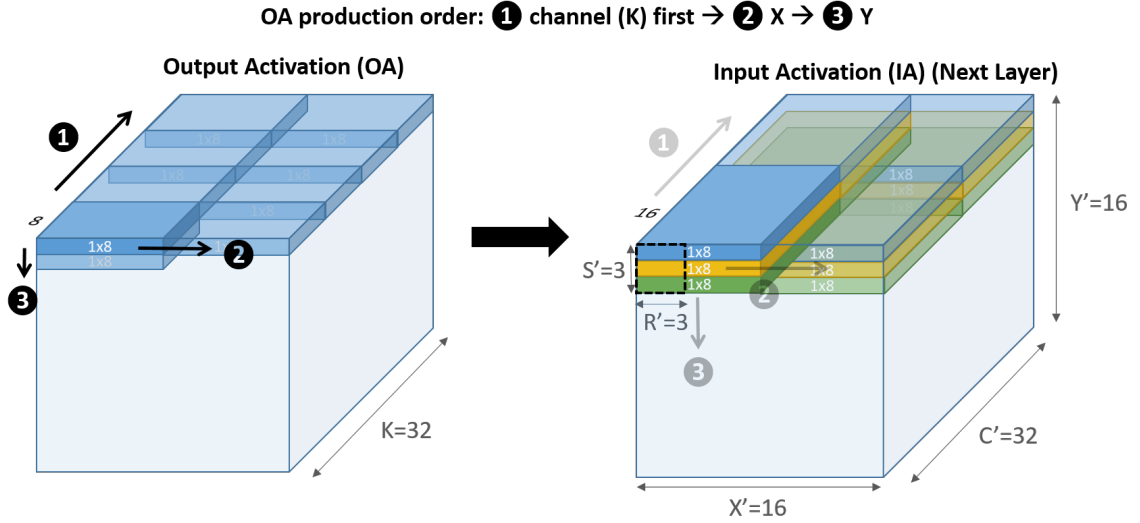


Figure 2.9: Weight memory mapping.



of the 8 kernel numbers $\times$ 16 channels $\times$ 16b. The corresponding output activations order of the convolver output is shown in Figure 2.10. The process scheduling is designed to align the OA order with the IA order. In this scheduling, the activations are computed with no extra indexing rearrangement for the streaming processing to the next layer.

**UART Interface.** The universal asynchronous receiver transmitter (UART) interface of the NetFlex chip is used for chip testing. An OpalKelly FPGA board is configured in the master mode and the chip is configured in the slave mode during chip testing. The UART interface includes two modules. One module of the UART with a set of TX and RX is responsible for configuring the registers of the NN layer parameters. The other module of the UART with 4 sets of TX and RX for faster data uploading and downloading is responsible for data read from or written to the input memory module, weight memory module, or output memory module.

**AIB Interface.** The AIB interface of the NetFlex chiplet has 8 AIB channels and is responsible for communication to FPGA chiplet or other NetFlex chiplets. Each AIB channel includes the parallel AIB I/Os and the AIB adaptor and consists of 96 signal and 42 power and ground pins. [4]. Each AIB signal has a potential

clock rate of 1GHz in double data rate (DDR) for up to 2Gb/s. The high bandwidth interface provides the possibility of scaling up a system to support larger workloads. The AIB interface is used for streaming input activations between layers to save the on-chip storage. The bus protocol can support two modes including master mode and slave mode. In the slave mode, the external signals are received by multiple AIB channels. The bus controller is responsible for aggregating outputs from multiple AIB channels which will be organized into a bus format of address and data. In the master mode, the signals from the core are sent to multiple AIB channels. The bus controller is responsible for distributing addresses and data to multiple AIB channels. The address is defined to be 32 bits and the data is defined to be 512 bits. There are 8 AIB channels in the AIB interface of the NetFlex chiplet and can be individually configured in master mode or slave mode according to the I/O bandwidth requirement for different applications.

### 2.3.2 System Architecture

**Ring Topology.** Multiple NetFlex chiplets can form a ring topology with the AIB interface for data streaming. Compared to the common mesh topology, the ring topology is simpler since data only travel in two directions and it costs less for implementation because of the elimination of complex routers. Without complicated NoC arbitration, the performance is improved. Although the ring topology has the drawback of a larger hop count with more nodes in the network, the mapping of a layer per chiplet for DNN inference naturally enforces the communication is between the neighboring chiplets, and it avoids multi-hop latency overhead.

**Direct Streaming and Layer-by-Layer Processing.** The NetFlex chiplet supports direct streaming as well as layer-by-layer processing. For direct streaming, the line-buffered depth-first processing is implemented [26]. The first 3, 5, or 7 rows depending on the kernel sizes will be buffered in a chiplet of the next layer for

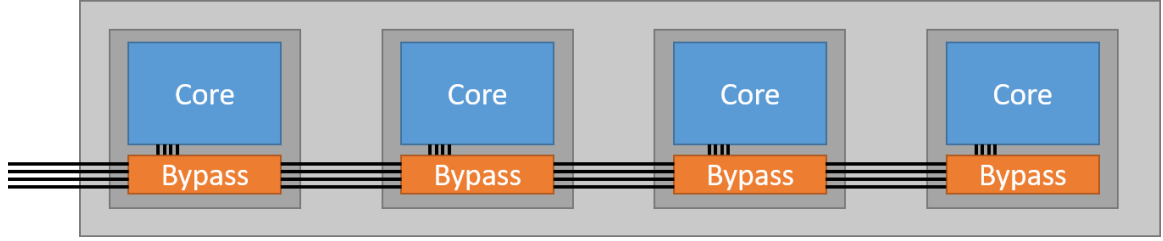
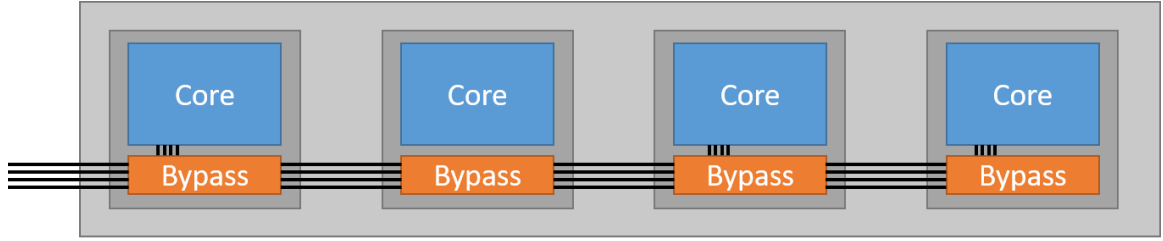
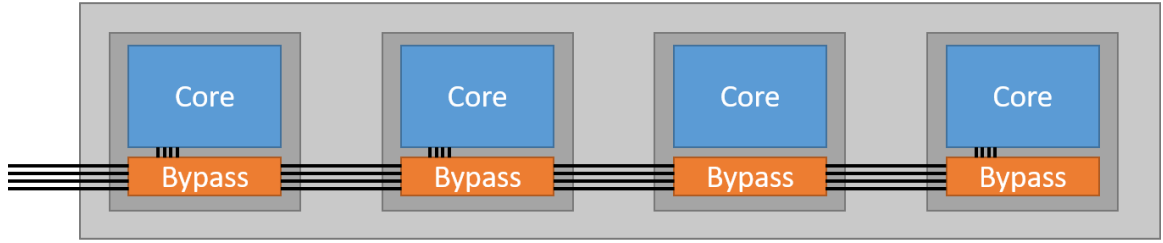


Figure 2.11: The 4 chiplet streaming topology in the normal mode.



(a)



(b)

Figure 2.12: The 4 chiplet streaming topology in the bypass mode.

depth-first processing. The next layer will start once the line buffer collects enough activations. By doing so, every layer can process a convolution operation once the previous layer generates one output activation. When each layer processes the activations at the same speed, depending on the depth of the layers, the direct streaming can better scale with more layers. In the case of mismatching speed for processing the activations between the layers, a chiplet can detect the line buffer of the current layer is full and temporarily stop the processing of the previous layer. However, this will result in pipeline stalls because of the back pressure from the slow-speed chiplet. For layer-by-layer processing, it requires an extra output memory module compared to direct streaming.

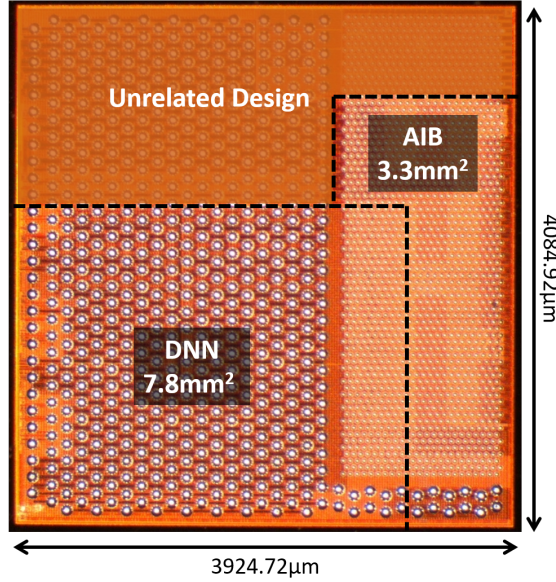


Figure 2.13: Die photo.

**Layer Skipping.** NetFlex supports layer skipping as discussed previously. Figure 2.11 shows the 4 chiplet streaming topology in the normal mode. Each chiplet includes the core processing unit and a bypass block to bypass the core processing unit. Figure 2.12 shows the bypass mode for the second and third chiplet for example.

## 2.4 Chip Measurement

The NetFlex chip was fabricated using Intel 22nm FinFET Low Power (22FFL) technology. The chip is integrated with Intel Stratix 10 in a 2.5D multi-chip package (MCP) through the embedded multi-die interconnect bridge (EMIB). The core accelerator occupies 7.8mm<sup>2</sup> as shown in Figure 2.13. An OpalKelly XEM-6310 FPGA board is used for chip testing. The chip achieves 2.65TOPS/W and 0.141TOPS/mm<sup>2</sup> at the supply voltage of 0.89V, running at the clock frequency of 506.9MHz at room temperature.

Figure 2.14 shows the voltage scaling performance of the chip from 0.45V to 1.05V with the clock frequency from 33.57MHz to 616.87MHz and power consumption from

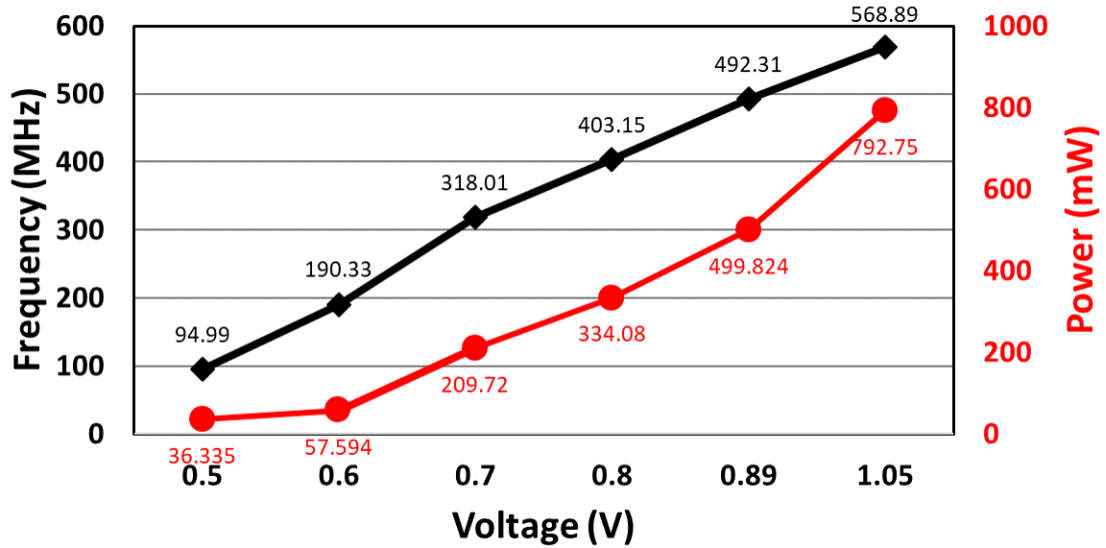


Figure 2.14: Frequency and power consumption with the voltage scaling.

Operation	Convolution						Deconvolution
Configuration	ksize 3, stride 1	ksize 3, stride 2	ksize 5, stride 1	ksize 5, stride 2	ksize 7, stride 1	ksize 7, stride 2	ksize 3, stride 2
Voltage (V)	0.89	0.89	0.89	0.89	0.89	0.89	0.89
Frequency (MHz)	492.31	492.31	492.31	492.31	492.31	492.31	492.31
Power (mW)	499.824	541.12	304.024	532.309	304.914	533.644	445.623

Table 2.3: Measurement results for various configurations.

10.14mW to 701.40mW.

Table 2.3 shows the measured results of convolution operation and deconvolution operation with different kernel size and stride configurations.

Table 2.4 presents the performance of the NetFlex chip compared with prior designs for perception processing. The NetFlex chip achieves competitive energy efficiency and higher area efficiency for a single chiplet. NetFlex is a chiplet-based system that can further scale to a larger system to obtain a higher frame rate and support various DNNs workloads flexibly.



	VLSI 2016 [20]	ISSCC 2019 [21]	This Work
Type	Harris feature + visual SLAM	CNN feature + visual SLAM	End-to-end CNN
Technology	65nm	28nm	22nm
Area	20mm <sup>2</sup>	10.92mm <sup>2</sup>	7.8mm <sup>2</sup>
Memory Size	854kB	1126kB	2492kB
Voltage	1.2V	0.9V	0.89V
Frequency	83.3MHz	215MHz	492.3MHz
Power	24mW + IMU power	243.6mW	499.8mW
TOPS	0.059	0.879	1.07
TOPS/W	2.46	3.61	1.36
TOPS/mm <sup>2</sup>	0.003	0.080	0.096
Dataset	EuRoC	KITTI	KITTI
Image Size	752×480	640×480	416×128
Throughput	90fps	80fps	108fps
Scalability	No	No	Yes
	stand alone accelerator	stand alone accelerator	2.5D integration system

Table 2.4: Comparison with prior works for perception.

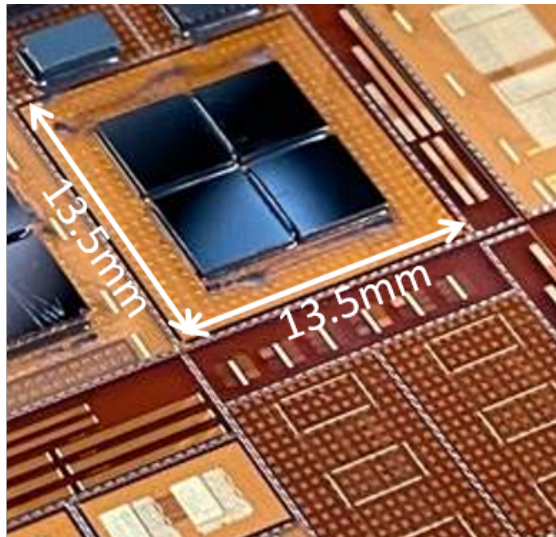


Figure 2.15: Package photo of the 4-chiplet system.

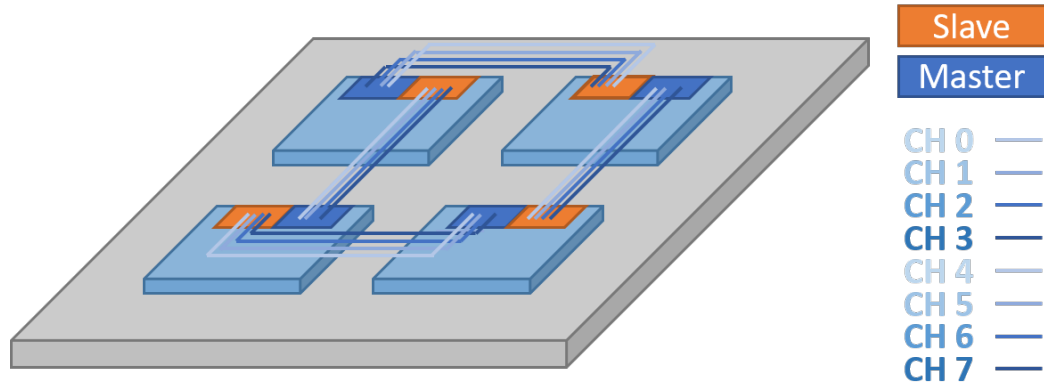


Figure 2.16: A face to back connection of 4 chiplets.

## 2.5 System Evaluation of 4-Chiplet Package

Figure 2.15 shows the 4-chiplet package prototype. The 2.5D integration of NetFlex chiplets is fabricated using redistribution layers (RDLs) using a fan-out wafer level packaging (FOWLP) technology. The bottom layer is connected to PCB with bottom vias and solder balls. The next 5 layers from the bottom are reserved for routing AIB signals and for distributing power and ground network. The metal lines for routing the AIB signals have a minimum spacing of  $2\mu\text{m}$  and a cross section of  $2\mu\text{m}\times 2\mu\text{m}$ . The top layer is connected to the under bump metallization (UBM) layer with top vias. The chiplets are flipped and attached to the UBM layer [27]. The package size is  $13.5\text{mm}\times 13.5\text{mm}$ .

The modular chiplet design provides flexibility for various connections between multiple chiplets. An example of a 4-chiplet system is shown in Figure 2.16. Here we define the side with AIB signal as the face side and the opposite side to be the back side. The AIB channels are assigned to 2 groups with channel 0 to channel 3 as a group and channel 4 to channel 7 as the other group. The AIB channels can be configured to master mode or slave mode that provides more flexibility for the place and route of AIB signals.

Table 2.5 summarizes the microarchitecture parameters of NetFlex. The area is

Package	Number of Chiplets	4
	Size	13.5mm × 13.5mm
	Core Voltage	0.89V
	Interconnect Clock Frequency	1GHz
	Inter-chiplet Interconnect	AIB
	Interconnect Bandwidth	10GB/s/Channel
	Interconnect Latency	<5ns
	Interconnect Energy	0.83pJ/bit
Chiplet	Number of PE arrays (K)	8
	Area	11.1mm <sup>2</sup>
	Technology	22nm FinFET
	Voltage	0.89V
	PE Clock Frequency	0.5GHz
	Bus Bandwidth	32GB/s
	Weight Buffer Size	2360kB
	Line Buffer Size	132kB
PE array	Depth Parallelism (C)	16
	Convolution Parallelism (CONV)	8
	Dataflow	Weight Stationary
	Input/Weight Precision	16b
	Partial-Sum Precision	20b

Table 2.5: Microarchitecture parameters of NetFlex. The package-level results are from [4]

	VLSI 2019 [5]	VLSI 2021 [6]	This Work
Technology	16nm	40nm	22nm
Area	6mm <sup>2</sup>	29.2mm <sup>2</sup>	7.8mm <sup>2</sup> + 3.3mm <sup>2</sup>
Precision	INT8	INT8, FP16	INT 16
Memory Size	752KB	0.5MB + 2MB RRAM	2492kB
Voltage	0.42-1.2V	1.1V	0.89V
Frequency	161-2000MHz	200MHz	492.3MHz
Power	30-4160mW	126mW	499.8mW + 287.7mW
TOPS	0.32-4	0.92	1.07
TOPS/W	0.2-9.1	2.2	1.36
TOPS/mm <sup>2</sup>	0.053-0.666	0.031	0.096
Package Substrate	Organic substrate	PCB	Silicon interposer
I/O Design	High speed serial	Slow speed	Moderate speed parallel

Table 2.6: Comparison of chiplet-based prototypes.

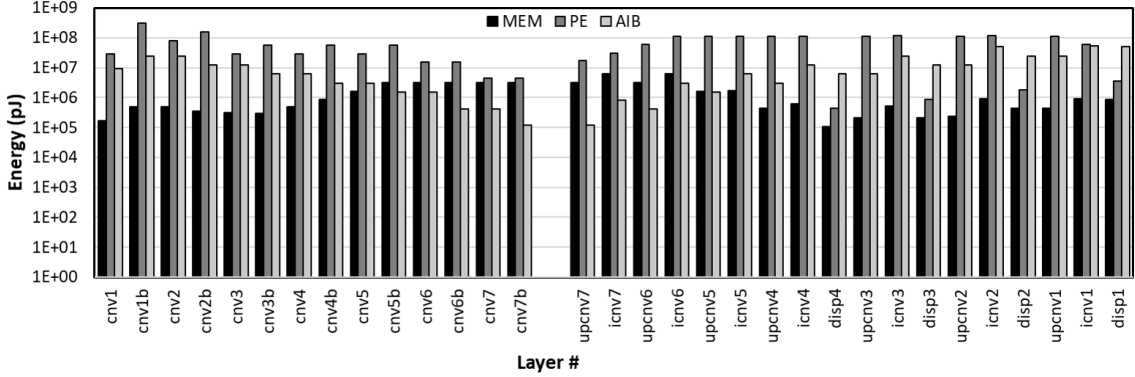


Figure 2.17: Energy consumption of SfM depth net.

made of the  $7.8\text{mm}^2$  core accelerator and the  $3.3\text{mm}^2$  AIB I/Os and the AIB adaptor. Table 2.6 shows the performance comparison of chiplet-based systems. Compared to state-of-the-art works, NetFlex can achieve higher energy and area efficiency from the co-optimization of the system and the accelerator architecture as well as the seamless pipelining between the core processing unit and the communication unit.

Figure 2.17 shows the processing, communication, and memory energy consumption of SfM depth net as a large workload example. Most of the energy is consumed by PE processing rather than AIB communication. The shape of the NN layer makes the first few and the last few layers computation-dominant with more activations and more MAC operations and the middle layers parameter-dominant with more weights. NetFlex can support both computation-dominant layers and parameter-dominant layers by configuring the chiplet to the direct streaming mode or the layer-by-layer mode depending on the layer property and the skip connections in the network.

## 2.6 Summary

As more DNN-based perception approaches and algorithms are proposed for navigation, more flexible hardware becomes more crucial. This work presents a scalable NetFlex accelerator for reconfigurable NN layers. The spatial processing and the temporal processing in the PE arrays are implemented to facilitate data reuse and

provide adaptation to different kernel sizes. The optimized dataflow for deconvolution operations removes the computation of 0s that exist in the decoder part. Process scheduling and memory mapping are designed for data streaming without extra indexing overhead. The modular chiplets are connected to form a ring topology to support large DNN workloads and can be gated by the skipping module to reduce the energy of computation and memory accesses. A NetFlex chip is fabricated and it achieves a higher throughput with competitive energy efficiency and higher area efficiency compared to the other silicon prototypes for perception application.

NetFlex can potentially support larger DNN-based perception applications by 2.5D integration. The parallel AIB and the ring topology contribute to a low latency overhead for streaming processing because of the simplicity of the parallel data interface and the ring topology. A 4-chiplet package demonstrates an example of the chiplet-based system. Compared to prior chiplet-based systems, NetFlex can achieve higher energy and area efficiency because of its co-optimization of the system and the accelerator architecture.

## CHAPTER III

# AR-PIM: PIM with Adaptive Range

### 3.1 Introduction

Deep neural network (DNN) has achieved exceptional success in many application domains. DNN has also become one primary workload for modern computing hardware. GPU, CPU, FPGA, and digital ASIC hardware [28, 29, 30, 31, 32, 33, 34] have demonstrated substantial accelerations for DNN workloads. In mobile and IoT devices, the energy budget is severely limited, requiring a higher energy efficiency to enable DNN processing. DNN model compression has emerged to be an effective technique. Sparse DNN hardware architectures have been created to support compressed models by storing compressed weights and eliminating redundant operations [35, 36]. However, unstructured pruning [35, 37] introduces irregularity, and extra control overhead is needed to handle the irregular data flow. Structured pruning [38] reduces the overhead, but it may sacrifice accuracy.

From the circuit level, processing in memory (PIM) is the ideal candidate for DNN inference computation for improving performance and energy efficiency. The DNN computation is easily parallelizable and can be directly mapped to PIM. More importantly, PIM removes the memory wall by eliminating data movement between memory units and processing units. However, PIM peripheral circuits need to be considered as an integral part of the evaluations. These include digital-to-analog

converters (DACs) for converting the digital inputs and analog-to-digital converters (ADCs) for digitizing the outputs. The peripheral circuits are known to overtake the memory device as the dominant part of PIM’s energy consumption, especially when the required resolution is high [17]. Reducing the resolution relaxes the converter requirements, but will sacrifice the accuracy. Maintaining the same resolution with the reduced number of rows will sacrifice the compute density.

Choosing the right PIM memory device is not obvious. Nonvolatile memory (NVM) devices such as resistive RAM (RRAM), magnetoresistive RAM (MRAM), phase-change RAM (PCRAM), ferroelectric RAM (FeRAM), are suitable for PIM thanks to their nonvolatility, low standby power, and high density. However, the commercially available NVM devices are still at 22nm [39, 40, 41, 42], significantly lagging the scaling of logic devices. The process, voltage, temperature (PVT) variations of the NVM devices also require a diligent control. Even though both SRAM and NVM suffer from variability, SRAM has no extra process cost, no drift issues, infinite endurance, and always leads technology scaling. 7nm and 5nm SRAM are already commercially available [43]. SRAMs in an even more advanced process is upcoming. Comparing a 7nm SRAM to a 22nm RRAM or MRAM, a 7nm SRAM offers a higher density, lower access energy, and higher access speed. A 7nm SRAM is an even better candidate for PIM for its higher compute density and energy efficiency if nonvolatility is not of primary consideration.

In this work, we investigate PIM based on a 7nm SRAM and explore practical analog PIM design choices. The investigation points to a promising direction of utilizing data sparsity in an adaptive design to relax the high-resolution ADC requirements. After introducing the challenges behind SRAM PIM architectures in Section 3.2, we present the contributions of this work, which are briefly summarized below:

1. An analysis of the practical design choices for 7nm SRAM PIM, accounting for the noise and non-idealities derived from the intrinsic nature of the SRAM cell

and analog accumulation in the bitline (Section 3.3).

2. Runtime range detection and adaptive-range PIM (AR-PIM) to relax the high-resolution ADC requirements (Section 3.4).

AR-PIM is benchmarked using DNN workloads with MNIST and ImageNet datasets to demonstrate energy efficiency and area improvements.

In Section 3.2, we provide a background of PIM using SRAM. In Section 3.3, we evaluate the PIM design choices using 7nm SPICE simulation. In Section 3.4, we propose the AR-PIM architecture and present a new runtime density detection mechanism and energy minimization strategy. In Section 3.5, we evaluate the AR-PIM architecture and compare with the reference architecture for DNN benchmarks. In Section 3.6, we conclude this work.

## 3.2 SRAM-Based PIM

PIM designs usually adopt a weight-stationary approach, where the weights are stored in a memory array and the input activations are passed to the array to perform computation. By keeping the weights in memory, the data movement is avoided. The weights are stored bit-parallel across columns as in [44, 45, 46]. In computation, a vector of inputs is driven, one per wordline (WL). The cells along a column are turned on, and the currents are summed on the bitline (BL), accomplishing the dot product between the input vector and the vector stored on the column of the memory array. Across the columns, dot products are conducted in parallel, essentially realizing a vector-matrix multiplication (VMM).

An SRAM bitcell stores a value and its complement. When its WL turns on, the stored value drives BL and the complement drives BLB. In SRAM-based PIM, either BL or BLB can be taken as the output as in Figure 3.1. In this work, BL is taken as the output. The key design parameters are considered below.



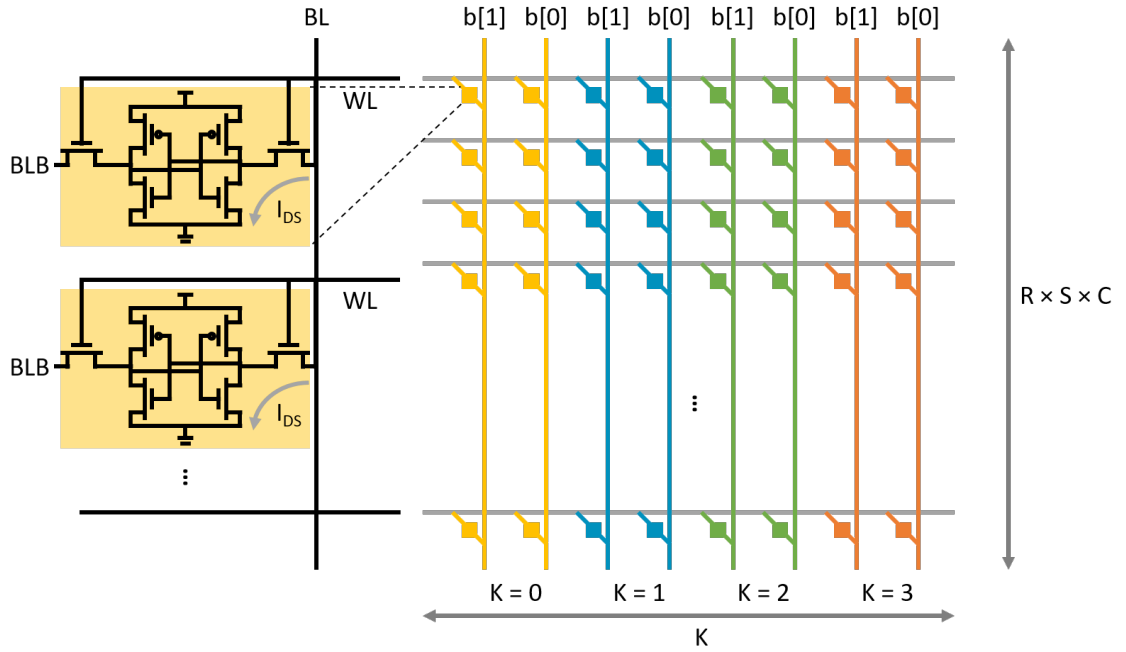


Figure 3.1: The PIM mapping of the convolution operation where  $R \times S$  is the kernel size,  $C$  is the input channel and  $K$  is the output channel. The 2b weight in the example is stored in two columns. The zoom-in view shows the 6T SRAM and current  $I_{DS}$  discharged by each bitcell. Both baseline PIM and AR-PIM adopt this mapping for convolution operation.

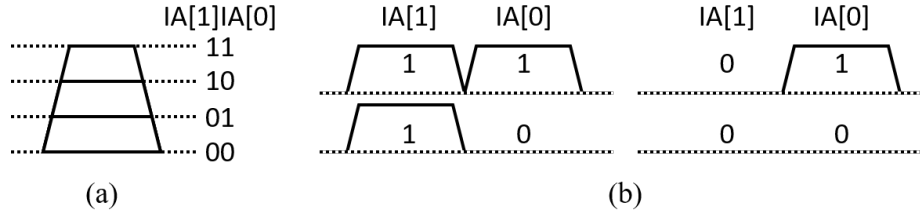


Figure 3.2: The multi-bit (2b in the example) input representation of (a) WL pulse level and (b) WL pulse train.

**Array Size.** Using a larger array, more cells can be computed in parallel, providing a higher performance; and the row and column peripheral circuits are amortized more effectively, leading to a higher compute density. A drawback of a larger array is the lower utilization in mapping smaller VMMs, leaving unused cells. A larger array also presents a higher capacitive loading on WL and BL, resulting in a longer delay. Finally, a larger array implies the potential activation of more cells to add their currents to the same BL, and thus the accumulation of more noise that may degrade the signal-to-noise ratio (SNR) making it impossible to discriminate the LSB.

**Input Encoding.** The input activations can be encoded in two forms as shown in Figure 3.2: (a) pulse level or (b) pulse train, i.e., each bit of the multi-bit input is represented by a 1b pulse. The pulse train is more linear compared to pulse level or width, and it can be better controlled [43], but the latency increases with the bit width. Past designs have combined pulse level and pulse train [17, 18, 19].

**BL Resolution.** The BL resolution depends on the WL resolution ( $b_{WL}$ ), the memory cell resolution and the number of activated memory cells ( $N_{cells}$ ) in a column. Since SRAM is a digital (1b) memory, the BL resolution is  $b_{BL} = b_{WL} + \log_2 N_{cells}$ . A higher BL resolution requires a higher-resolution ADC, which in turn significantly impacts power and area [17, 3]. A higher BL resolution also exacerbates PIM’s variation and reduces the capability of error tolerance [47].

**ADC Sharing.** The ADC area can be significantly larger compared to the bitcell pitch. Placing one ADC per BL is difficult due to the physical layout constraint.

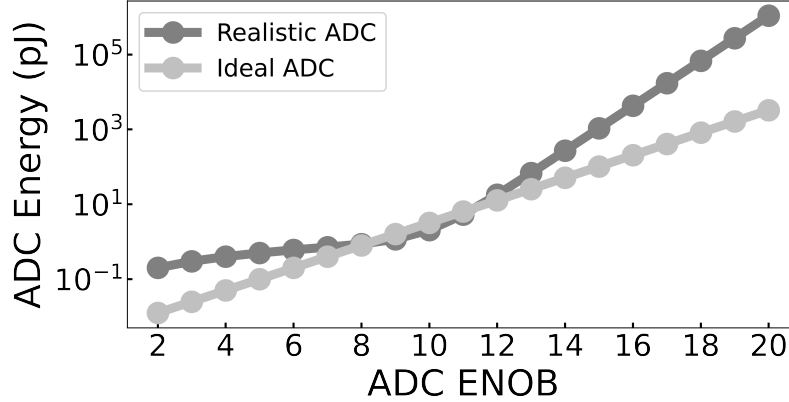


Figure 3.3: ADC energy consumption with effective number of bits (ENOB) from [3].

Since the conversion time of ADC can be much shorter than the time it takes for the SRAM BL current to develop, sharing an ADC between BLs becomes a necessity, e.g., 1 ADC is shared by 4 BLs in [43]. ADC sharing requires extra circuits such as sample and hold [17] or weighted capacitors [43] to store the BL value before the conversion starts.

### 3.3 7nm SRAM PIM Design Considerations

The evaluations are based on SPICE simulations on a  $128 \times 128$  SRAM array in 7nm FinFET CMOS technology. <sup>1</sup> The ADC energy is extracted from [3]. The array size of 128 achieves the highest utilization for DNN workloads as indicated in [48].

#### 3.3.1 Input Encoding

Input encoding choices are investigated. If the input activations are 1b, they can be encoded in 1b WL pulses. Since a BL is connected to 128 bitcells, the BL resolution is  $b_{WL} = 8$ . An 8b ADC consumes 96% of the total energy (Figure 3.4), significantly higher than the energy due to memory access or the 1b DAC. For a 2b input activation, two input encoding options are available: 1b pulses over 2 cycles (with partial sums

<sup>1</sup>This section is collaborated with Arm Research.

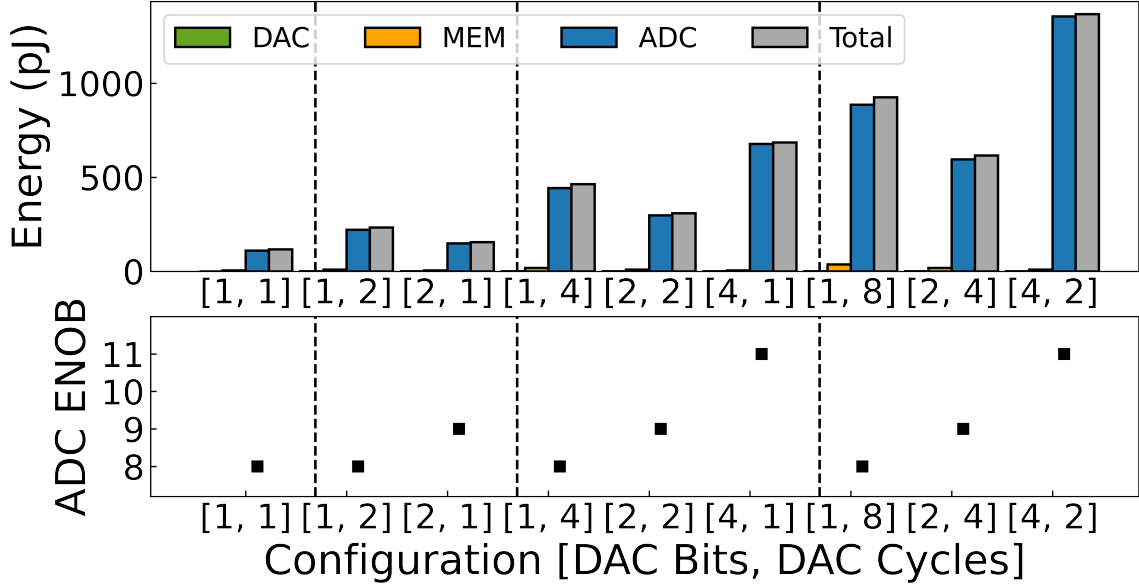


Figure 3.4: Energy efficiency of 1b, 2b, 4b, 8b input activation. The different input encoding for each configuration is implemented by DAC bits and DAC cycles with the corresponding ADC resolution.

scaled and added digitally post-ADC) or a single-cycle 2b pulse, resulting in 8b and 9b BL resolution, respectively. A 9b ADC consumes 35% more energy than an 8b ADC (Figure 3.3). Hence, two 8b analog-to-digital (A/D) conversions with [DAC bits, DAC Cycles] = [1, 2] result in 49% higher energy than one 9b A/D conversion with [DAC bits, DAC Cycles] = [2, 1], so the single-cycle 2b pulse encoding is preferable in terms of energy (Figure 3.4).

Moving to 4b and 8b input activations, more input encoding options are available. The 2b-pulse encoding was found to be the sweet spot in energy consumption. However, regardless of the input encoding choice, the ADC dominates the energy consumption. A WL resolution of higher than 2b is not practical due to the significant escalation of ADC energy.

### 3.3.2 BL Resolution

If a bitcell storing a 1 is activated, the bitcell discharges one unit of current from BL. However, process variations complicate the picture. As more discharging

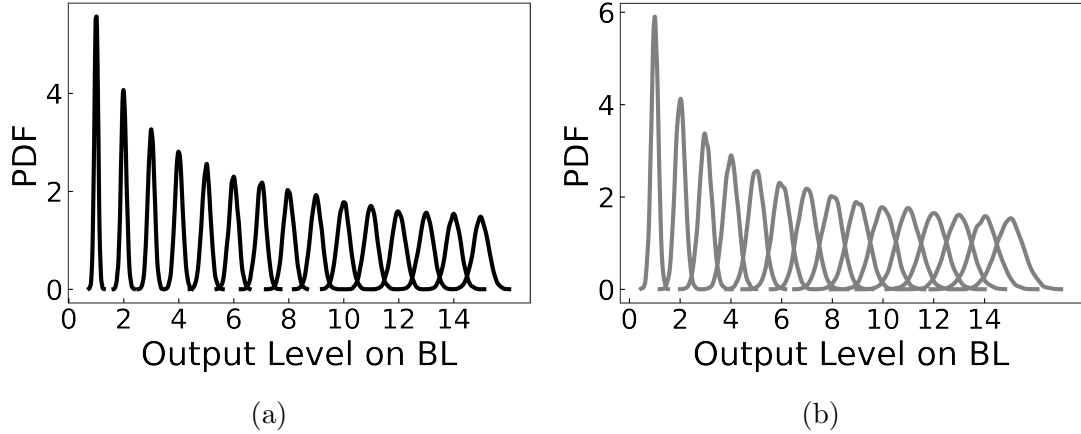


Figure 3.5: SRAM BL current levels from SPICE simulation for WL voltage of (a) 0.8V and (b) 0.6V. Each Gaussian distribution represents one output level. The less overlap between two distributions means the better sensing margin for distinguishing two output levels.

bitcells are activated, the distribution of current gets wider (Figure 3.5a). The wide distribution makes it challenging to decode as few as 16 current levels. This insight suggests that the degradation of SNR due to process variations may make using a very high-resolution ADC inconsequential.

As the WL voltage level is reduced, e.g., in supporting WL pulse-level input encoding, the current level boundaries are further obscured as seen in Figure 3.5b. If the PVT is considered, the effective BL resolution will be further reduced.

### 3.3.3 ADC Resolution and ADC Sharing

For this investigation, a reference SRAM-based PIM design is adopted: a  $128 \times 128$  SRAM array; the input is provided bit serially; WL is encoded in 1b pulses. For example, an 8b input activation is passed to WL by a 1b DAC in 8 cycles. An 8b weight is stored in 8 SRAM bitcells in a row. Therefore, the full resolution required at each BL is 8b. A weight-stationary digital design was synthesized using an array of multiply-accumulates (MACs) with weight storage to mimic PIM. The partial sums are accumulated along a column of MACs. The digital design also follows the same bitwidth as the PIM design for comparison at the MAC level.

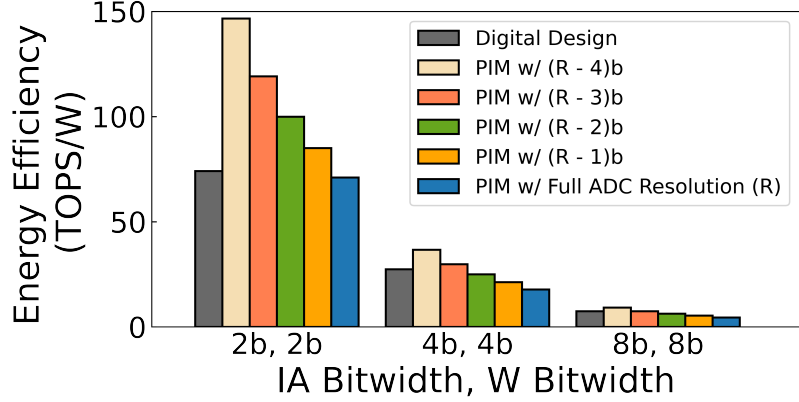


Figure 3.6: The energy efficiency comparison for synthesized digital design and PIM design with different BL resolutions and 3 Input Activation (IA) and Weight (W) bitwidth combinations.

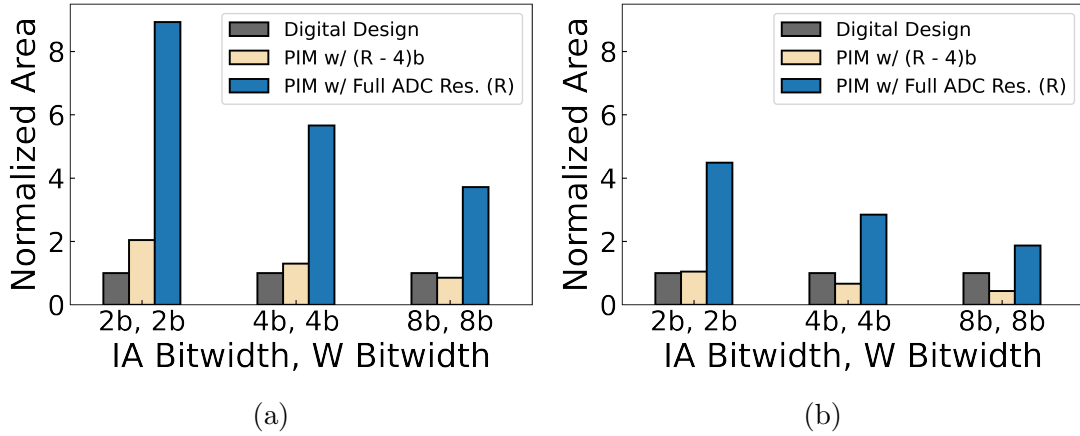


Figure 3.7: Area comparison between synthesized digital design and PIM design for (a) 1 ADC per BL and (b) 1 ADC shared by 2 BLs.

The energy efficiency of the PIM design is compared to the digital design in Figure 3.6, assuming one ADC per BL. PIM achieves the best energy efficiency when the activation and weight bitwidth are low. Also, note that PIM with an ADC that supports the full BL resolution fares worse than the digital design. For PIM’s energy efficiency to be competitive, the ADC resolution needs to be reduced to 3b or 4b below the full BL resolution.

The area of the synthesized digital design is compared to PIM in Figure 3.7 for one ADC per BL. Due to the relatively large area of ADC, especially a high-resolution

ADC, the area of PIM easily exceeds the digital design. When the ADC resolution is reduced to 3b below the full BL resolution, the area becomes comparable. Figure 3.7b shows the configurations of one ADC shared by 2 BLs to reduce the PIM area. The results highlight the importance of reducing the ADC resolution and ADC sharing in keeping PIM more area efficient than a digital design.

### 3.3.4 Necessity to Control Resolution

The above sections highlight the challenges behind a high BL resolution and its feasibility due to the sensing margin and reducing the ADC resolution is a must to make PIM more competitive in energy efficiency and area.

To control the BL resolution, WL resolution can be reduced by employing 1b-pulse encoding over multiple cycles, and activating only a subset of rows at a time such as in [49], which requires more cycles to complete the computation. We propose an additional approach to leverage data sparsity in controlling the BL range in runtime. This approach can prevent sacrificing the classification accuracy incurred by direct truncation on BL values with the reduced ADC resolution.

## 3.4 The AR-PIM Architecture

For a bitcell to contribute to the BL current and increase the BL range, the bitcell needs to store a 1 and it needs to be activated. This implies that both the weight value and the input activation value are 1 (at bit level). If either the weight value or the input activation value is 0, the bitcell does not contribute to the BL current or the BL range. Therefore, the BL resolution quoted in the previous sections is the maximum resolution, while the effective BL range can be lower.

The presence of 0s in weights and input activations are known as weight and input sparsity. Sparsity exists even in unpruned models, especially with the popular rectified linear unit (ReLU) as the activation function that produces 0s in activations. The

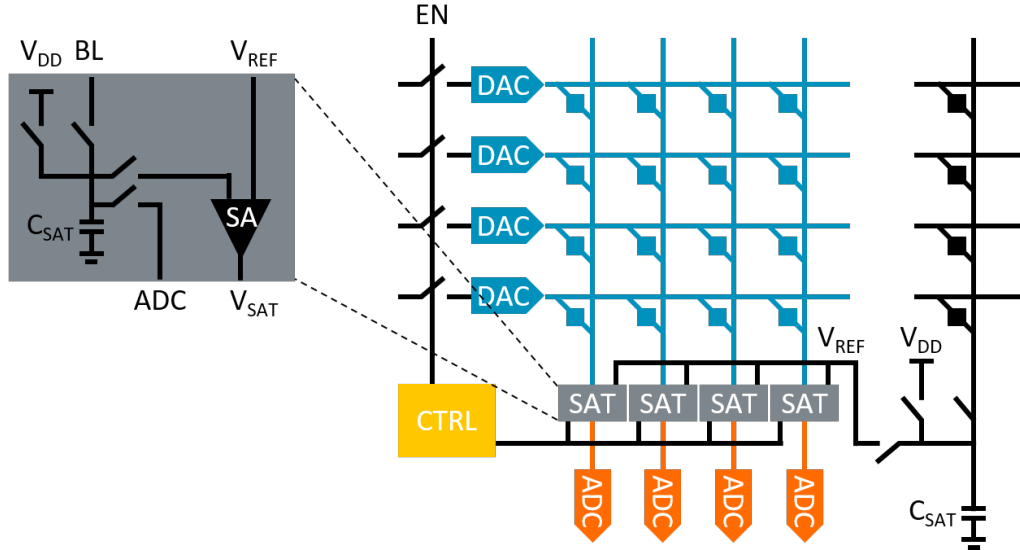


Figure 3.8: AR-PIM architecture with BL detection circuitry.

sparsity can be further increased in pruned models, where some weights are removed by pruning algorithms. In addition to word-level sparsity, plenty of bit-level sparsity exists in both weights and input activations as identified by [50].

In DNN inference, a model is given and the weight sparsity is static. However, the activation sparsity is dynamic, namely input dependent and determined at runtime. As a result, when the computation of the DNN inference is mapped to PIM, the BL range can vary due to the dynamic activation sparsity. We propose a technique to detect the runtime sparsity (or density). If the density is low, an energy-inexpensive low-resolution ADC can be used; and if the density is high, the BL range can be adjusted by activating only a portion of the bitcells.

### 3.4.1 Runtime Density Detection

The runtime density detection can be implemented by reusing the SRAM sense amplifier (SA) in the readout circuitry and a reference column as shown in Figure 3.8. The reference column stores a preset number of 1s to correspond to a given density level, e.g., 25% of the reference column storing 1 to represent a density of 25%. Prior



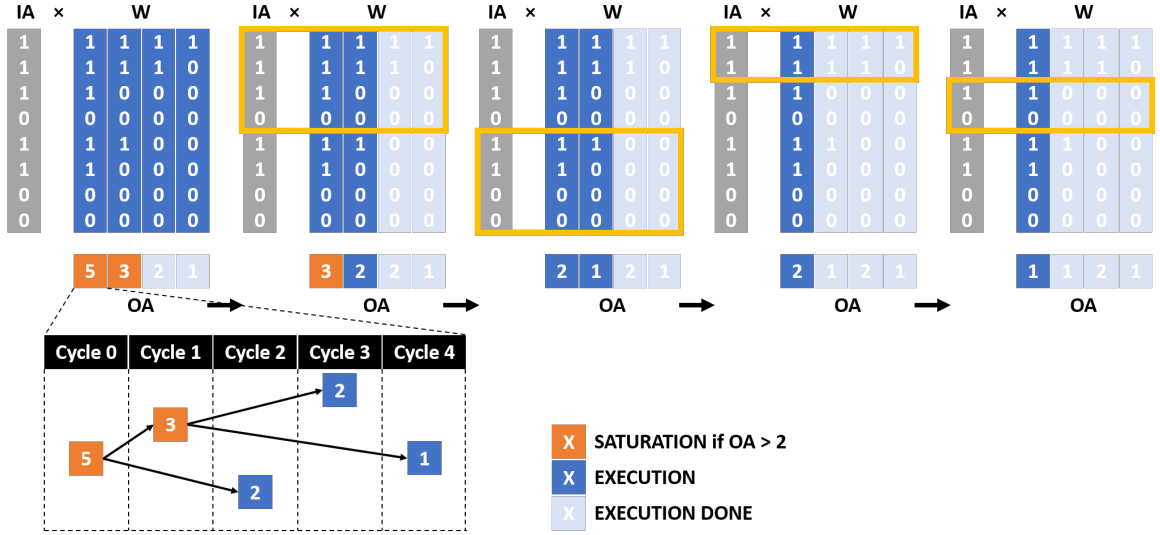


Figure 3.9: Example of runtime density detection and adaptation.

to detection, a readout from the reference column is performed by applying unit pulses on all WLs. The reference column's BL current is integrated on a sampling capacitor as the threshold voltage to represent a 25% density.

The density detection is done by an SRAM readout. The BL current is integrated on a sampling capacitor to be the BL voltage. The SA compares the BL voltage to the threshold voltage generated by the reference column. If the BL voltage is below the threshold voltage, the density of the column is higher than the reference density, and the SA signals  $EN = 1$  to the controller.

The controller checks all BLs' SA outputs. If at least one SA signals  $EN = 1$ , the controller activates only 50% of the WLs and another round of SRAM readout follows. In the next round, if at least one  $EN = 1$ , the controller activates just 25% of the WLs in subsequent SRAM readouts. The process continues until the density is reduced to the reference level or below, thereby controlling the BL range.

Figure 3.9 illustrates BL range control. Assume that prior to the detection, the threshold voltage is set to represent a 25% density. In the example, in cycle 0, the first and the second BL density exceed the 25% threshold, and the controller only activates 50% of the rows in the subsequent cycle 1 and cycle 2. In cycle 1, the first

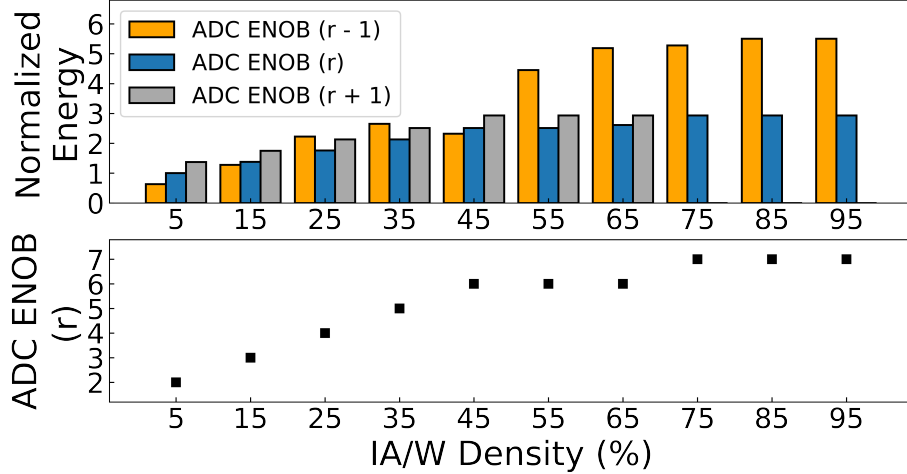


Figure 3.10: Energy of AR-PIM for IA/W density from 5% to 95% with conditions of ADC ENOB  $r-1$ ,  $r$ ,  $r+1$  (top) and ADC ENOB  $r$  for IA/W density from 5% to 95%.

BL density still exceeds the threshold, and the controller activates only 25% of the rows in the subsequent cycle 3 and cycle 4. By actively limiting the density below 25%, the BL resolution is reduced by 2b. The proposed range control adapts to the effective BL resolution by activating more or fewer bitcells, i.e., tuning the range of BL currents. Thus we name it AR-PIM.

### 3.4.2 Energy Minimization

To reduce the ADC resolution and improve the sensing margin, low-resolution ADCs are necessary. When adopting low-resolution ADCs, only a portion of WLs at a time can be activated and the BL values are read out sequentially. Therefore, it may result in more processing cycles, which in turn costs more energy and a longer latency.

To amortize this overhead, AR-PIM exploits the lower effectual BL range in runtime originating from data density levels of input activations and weights.

The lowest energy is investigated by sweeping the activation and weight density from 5% to 95%. If the ADC resolution is set based on the effective BL range (using the IA/W density levels as the proxy indicator), the number of processing cycles and

Model	LeNet		AlexNet				
Dataset	MNIST		ImageNet				
Layer	CONV1	CONV2	CONV1	CONV2	CONV3	CONV4	CONV5
BL Value Mean	0.383	4.630	27.957	14.478	6.932	2.945	2.764
BL Value Std	1.038	4.277	17.429	12.804	6.123	2.977	2.999

Model	VGG11							
Dataset	ImageNet							
Layer	CONV1	CONV2	CONV3	CONV4	CONV5	CONV6	CONV7	CONV8
BL Value Mean	6.420	9.030	9.906	5.907	6.798	3.905	3.463	2.925
BL Value Std	4.250	9.604	8.650	5.468	5.710	3.750	3.420	3.110

Table 3.1: Mean and standard deviation of BL density numbers. (The maximum BL density is 128 for a  $128 \times 128$  array.)

energy consumption can be minimized. Figure 3.10 shows the result of choosing the appropriate ADC resolution represented as ENOB for optimal energy consumption.

We develop a strategy to minimize the energy cost with respect to ADC resolution and data density. The product of an input bit and the bitcell value follows a Bernoulli distribution with  $p$  being the probability of 1 and  $q = 1 - p$  being the probability of 0. Here  $p$  reflects the activation and weight density. Adding  $n$  products on a BL, where  $n$  is the array size, produces the sum that follows a Binomial distribution of mean  $np$  and variance  $npq$  (effectual BL range). Let  $k$  be the density number, i.e., the number of 1s being summed up on the BL. The CDF of the binomial distribution can be written as:

$$F(k, n, p) = \sum_{j=0}^k \frac{n!}{j!(n-j)!} p^j (1-p)^{n-j}.$$

Let ADC resolution  $t$  be an integer value ranging from 0 to  $\log_2 n$  and the threshold is set as  $k_t = 2^t$ . The total energy  $E_{AR-PIM}$  can be written as the sum of energy of each PIM step  $E_{PIM}$ , taking into account the need for more steps when the density exceeds the threshold as shown in Figure 3.9.  $E_{PIM}(t, n)$  represents the sum of ADC, DAC, other peripheral circuits, and memory energy for a MAC operation.

$$\begin{aligned}
E_{AR-PIM}(k_t, n, p) &= E_{PIM}(t, n) \cdot F(k_t, n, p) \\
&+ E_{PIM}(t, n) \cdot 2 \cdot [ F(2k_t, n, p) - F(k_t, n, p) ] + \dots \\
&+ E_{PIM}(t, n) \cdot \frac{n}{2^t} \cdot [ F(n, n, p) - F(\frac{n}{2}, n, p) ] \\
&= E_{PIM}(t, n) \cdot [ \frac{n}{2^t} \cdot F(n, n, p) + \sum_{i=0}^{\log_2 n - t - 1} (2^i - 2^{i+1}) \cdot F(2^i k_t, n, p) ].
\end{aligned}$$

Substituting  $F(n, n, p) = 1$ , we get

$$\begin{aligned}
&E_{AR-PIM}(k_t, n, p) \\
&= E_{PIM}(t, n) \cdot [ \frac{n}{2^t} - \sum_{i=0}^{\log_2 n - t - 1} 2^i \cdot F(2^i k_t, n, p) ].
\end{aligned}$$

AR-PIM minimizes energy consumption by jointly considering the data density  $p$  and appropriate ADC resolution  $t$  utilizing the runtime BL density detection technique.

## 3.5 Evaluation

### 3.5.1 Energy Consumption and Performance

The energy consumption of the AR-PIM architecture is evaluated using three DNN workloads based on the bit-level sparsity of activations and weights. The DNN workloads include LeNet using MNIST dataset, AlexNet, and VGG11 using ImageNet dataset. The energy of AR-PIM is highly dependent on two factors including the runtime data sparsity and the ADC resolution.

Using LeNet with MNIST dataset, both activations and weights are quantized to 8b in the evaluations. Table 3.1 shows the average density numbers of BLs when running inference in the first and the second convolution layers. Each layer is mapped

to a  $128 \times 128$  PIM module [17]. The first layer consists of six  $3 \times 3$  kernels. Each kernel is mapped to 8 columns (one bit per column) for a total of 48 columns. Note that the utilization of each column of cells is relatively low, at  $9/128$  or 7%. The second layer consists of sixteen  $3 \times 3 \times 6$  kernels. Each kernel is mapped to 8 columns (one bit per column) for a total of 128 columns. The utilization of each column of cells is at  $54/128$  or 42%.

In both layers, the average BL densities stay below 10%, making AR-PIM suitable as the effective BL range is low and consistent between columns.

Figure 3.11a shows the normalized energy with different ADC resolution settings for the first convolution layer and the second convolution layer of LeNet. In the first convolution layer, a lower ADC resolution reduces the energy consumption. The low utilization of cells, or  $9/128$ , along each column leads to a consistent low effective BL range. The low average BL density and the narrow distribution (Table 3.1) allow the setting of a low ADC resolution to aggressively reduce the BL resolution to save the most energy.

In the second convolution layer, a different trend is observed: when the ADC resolution is too low, the energy consumption increases. Different from the first layer, the utilization in the second layer of mapping is higher, at  $54/128$ . The higher utilization results in a broader BL density distribution (Table 3.1). The lowest ADC resolution could result in a large number of extra cycles and more energy. The energy-optimal ADC resolution can be set to capture most of the BLs, leaving only a small number of extra cycles to capture the remaining BLs.

Figure 3.11b shows the latency implications of different ADC resolution settings. Generally speaking, the lower the ADC resolution, the higher the latency. The energy-optimal points tend to be low-resolution points where the latency does not increase excessively.

Figure 3.11c and Figure 3.11d shows the normalized energy and latency of AR-PIM

with different ADC resolution settings for the first, middle, and the last convolutional layers in AlexNet using the ImageNet dataset. The activations and weights are quantized to 16b and 12b in the evaluations while maintaining the inference accuracy. The mean of BL density numbers decreases and the distributions get narrow in deep layers as shown in Table 3.1.

Similar behavior can be observed in VGG11 as shown in Figure 3.11e and Figure 3.11f. Across different DNN workloads, AR-PIM can minimize the energy consumption while maintaining the inference accuracy over the baseline with full 7b ADC resolution. As a result, the AR-PIM architecture improves the energy gain up to  $3.2\times$  over the baseline PIM in the deep layers as the BL density decreases. In latency constrained applications, AR-PIM can still provide up to  $1.7\times$  higher energy efficiency.

### 3.5.2 Discussion

**Latency Overhead.** AR-PIM exploits the runtime data sparsity of multiple BLs in an array. Each BL is detected and checked whether the BL value overflows or not. Since the BLs within an array share the same input, the latency overhead will emerge for waiting for other BLs to resolve the dot product result. The proposed AR-PIM benefits more with consistent BL values. If the BL values within an array are consistently high in the same cycle, then every BL of the array takes longer cycles together. If the BL values within an array are consistently low, then the array can be early terminated together. So the span of BL values within an array matters more. Figure 3.12a shows the BL distribution of 128 columns for the first layer of AlexNet workload. It is observed from the box plot that for the same time step the difference between BL values is not exceeding 30% of 128. Figure 3.12b shows the BL distribution of 128 columns for the last layer of AlexNet workload. The BL value difference for the same time step becomes even smaller. The BL distributions of the

first and the last layer show that the BL difference is not large so the latency overhead is within the acceptable range.

### 3.6 Summary

This work explores the design boundary for analog PIM using SRAM in a 7nm process. The input encoding, BL resolution, ADC resolution, and ADC sharing are studied to analyze their impacts on the energy efficiency and the area cost of analog PIM. From the analyses, we conclude that low-bitwidth NNs are more suitable to be deployed on the analog PIM to save energy on power-constrained mobile devices. Addressing the challenges behind the deployment of multi-bit matrices in analog accelerators, the AR-PIM architecture is presented with a runtime BL density detection mechanism to adapt to a lower effective BL range.

The AR-PIM architecture eliminates the need for high-resolution ADCs and reduces the energy consumption of the ADC proportion. By reducing the BL range, AR-PIM also enhances the variation tolerance capability and the sensing margin. By considering the energy gain and latency overhead together, our evaluations show that AR-PIM provides  $1.7\times$  higher energy efficiency over the baseline PIM with  $4.3\times$  area reduction while maintaining the inference accuracy.

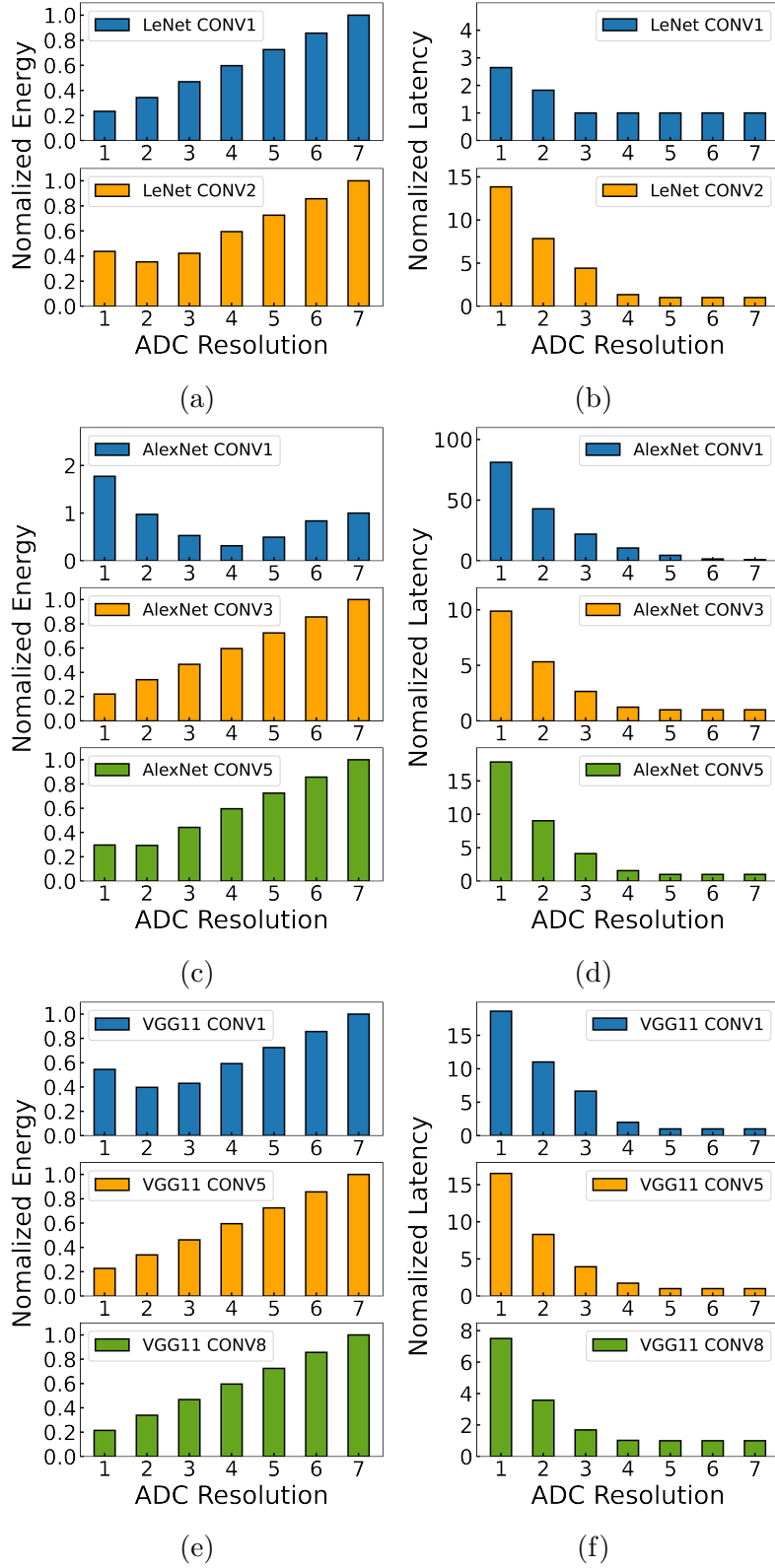
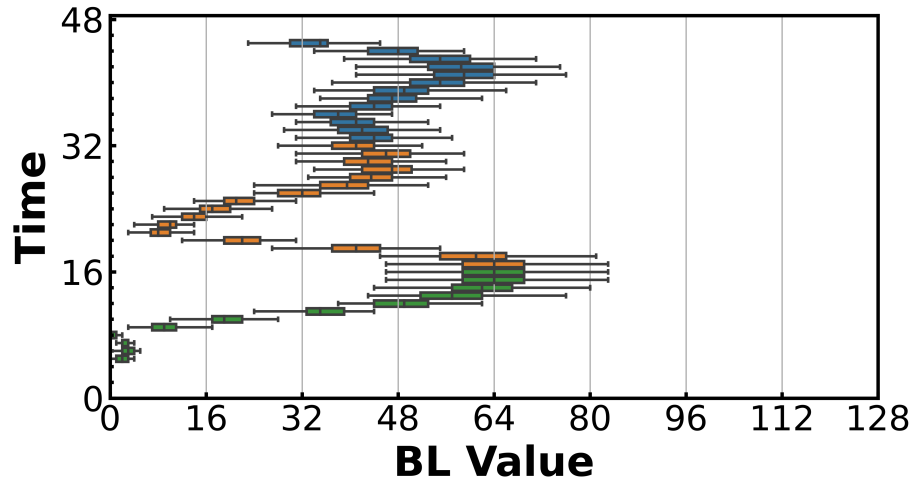
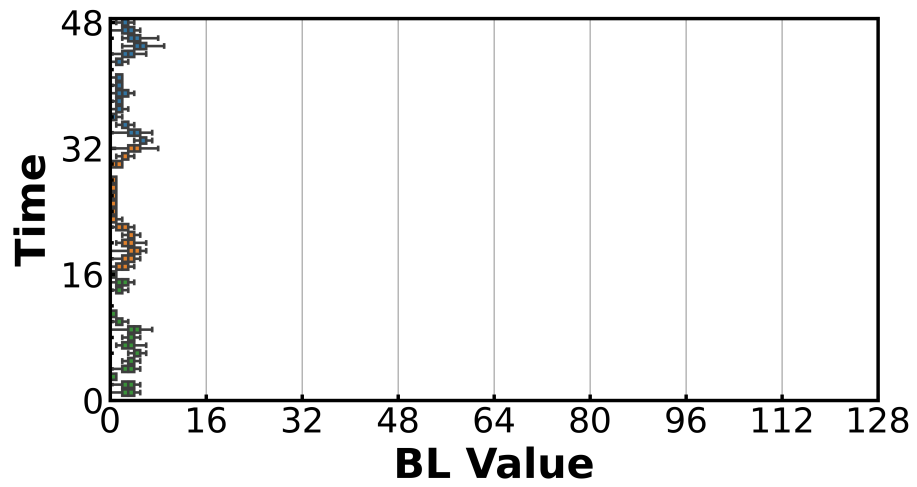


Figure 3.11: Energy consumption and latency of AR-PIM compared and normalized to the baseline PIM with 7b ADC resolution (the rightmost bar in each figure) for each layer running (a)(b) LeNet using MNIST dataset, (c)(d) AlexNet using ImageNet dataset, and (e)(f) VGG11 using ImageNet.





(a)



(b)

Figure 3.12: Latency overhead.

## CHAPTER IV

# CASCADE: Chaining PIM in Analog Dataflow

### 4.1 Introduction

In recent years, machine learning is becoming the dominant workload for the next-generation computation systems. As one of the most important machine learning kernels, deep neural networks (DNNs) and recurrent neural networks (RNNs) are now being widely deployed in tasks from image analysis to speech recognition.

DNN workloads are typically highly vectorized with well-defined dataflow patterns. A large number of digital DNN accelerators [33, 32, 34, 35, 36] have been designed to achieve high performance and high efficiency. With the continued increase in DNN complexity and the growing demand for faster and lower power machine learning use cases, we see the need of power efficient chips that can be employed in a wide range of applications including autonomous driving and mobile devices. In such edge applications, an advanced nonvolatile memory, such as resistive RAM (RRAM), can play an important role thanks to its high storage density, low leakage power and fast wake-up from sleep [51].

Beyond high-density storage, recent work started looking into in-RRAM computation, a form of processing in memory (PIM), to enable massively parallel dot products in an RRAM crossbar array without moving the stored operands. The core computation of a DNN layer can be easily mapped to an RRAM crossbar array: the

input activations of a DNN layer are applied to the wordlines (WL) of an RRAM crossbar as voltage pulses, and the weights are stored as conductances of the RRAM crossbar. A dot product is obtained from the bitline (BL) of each column of the RRAM crossbar by the physics of Ohm’s law for multiplication and Kirchhoff’s law for accumulation. The simple and elegant in-RRAM dot product has been projected to achieve impressive performance and efficiency [52, 53, 54, 55, 56, 57].

In-RRAM dot product is a form of analog computation. When used as a part of a digital system, the digital inputs to the RRAM crossbar need to be converted to voltage pulses using digital-to-analog converters (DACs), and the outputs of the RRAM crossbar in the form of analog currents need to be integrated and digitized using analog-to-digital converters (ADCs). In-RRAM computation pushes the resolution requirement of analog computation to accommodate tens or hundreds of products of multi-bit WL pulses with multi-bit RRAM conductances that are summed together. High-resolution ADCs are required, adding a significant overhead. As RRAM crossbar size and device resolution continue to increase, the required ADC resolution also increases. It would not be surprising to see that analog-to-digital (A/D) conversion will eventually dominate the area and energy consumption of in-RRAM computation to an extent that renders in-RRAM computation impractical.

A survey of recent work on in-RRAM computation highlights the overhead of A/D conversion as a severe limitation. As a full-fledged DNN accelerator in RRAM, ISAAC [17] employs 8-bit ADCs that are estimated to cost 58% of the power and 31% of the silicon area. PRIME [18] uses sense amplifiers (SAs) instead of conventional ADCs to reduce area. However, an SA is only capable of resolving one bit at a time. To obtain a 6-bit digital output, the SA uses up to  $2^6$  cycles in decision time, resulting in a long latency that is exponentially dependent on the resolution. The SA interface limits the throughput for demanding applications.

The second limitation of in-RRAM computation is that even a single layer in a

state-of-the-art DNN or RNN can be too large to fit on a practical RRAM crossbar. The reason is twofold. First, despite the rapid progress in RRAM technology development, most RRAM crossbars demonstrated are of sizes from  $64 \times 64$  to  $256 \times 256$  [58], allowing up to 64 to 256 partial sums to be accumulated on each BL. In comparison, a single point in the output feature map (fmap) of a fully-connected (FC) layer in AlexNet [59] requires up to 9,216 partial sum accumulations, and a single point in the output fmap of a convolutional layer in GoogLeNet [60] requires up to 1,728 partial sum accumulations, both easily exceeding the number of analog accumulations that can be done in a practical RRAM crossbar. Therefore, one kernel computation needs to be separated and mapped to multiple RRAM crossbars. The resulting partial sums from multiple crossbars need to be digitized and accumulated in the digital domain. Second, it is impractical to assume that a 16-bit or even a 8-bit weight value can be reliably stored in one RRAM cell. Multi-level cell (MLC) requires the use of more complex DACs and ADCs, and can be more easily affected by noise and process variation. It is more practical to map a multi-bit weight value to multiple RRAM cells. Similarly, it is more practical to separate an input to units of 1 or 2 bits and apply them serially to simplify the circuitry and reduce the noise and variation uncertainty. It is also more practical to activate a subset of WLs, instead of all WLs, in a large RRAM crossbar. All of these practical approaches lead to more partial sums that need to be digitized and digitally accumulated.

In essence, in-RRAM computation consists of at least three parts: in-RRAM dot products, A/D conversion, and digital accumulation of partial sums. Currently, only the first part is done in RRAM, while the second and the third part are done by conventional CMOS circuits. Besides the aforementioned overhead of high-resolution A/D conversion, we estimate that the energy and area of digital partial-sum accumulation can surpass in-RRAM dot products to yield the core in-RRAM computation insignificant.

In this work, we present CASCADE, an in-RRAM computation architecture for DNNs and RNNs, to specifically address the problems of high-cost A/D conversion and digital partial sum accumulation associated with the current in-RRAM computation approach. The contributions of this work are as follows:

1. We choose more practical and robust in-memory dot products by reducing effective BL resolution to ensure noise and variation tolerance. Only low-resolution analog outputs can be reliably cascaded. We analyze the trade-off between resolution and inference accuracy, and show that only by lowering the BL resolution, a good accuracy can be reliably achieved.
2. We propose R-Mapping scheme to use a buffer RRAM to perform in-RRAM partial sum accumulation, replacing digital partial sum accumulation. The analog summation bypasses the A/D conversions of low-order sums, reducing the number of A/D conversions.
3. We connect MAC RRAMs to buffer RRAMs by using the transimpedance amplifiers (TIAs) as the interface to convert MAC RRAMs’ BL outputs from analog current to analog voltage that can directly feed to buffer RRAMs as inputs. Cascading MAC RRAMs with buffer RRAMs not only enables the “analog” dataflow to meet the computation requirement of a DNN or RNN layer, but also keeps all intermediate values in analog and in memory to obtain the highest possible energy efficiency and performance.

In Section 4.2, we provide a background of PIM using RRAM. In Section 4.3, we illustrate the CASCADE architecture and present a new memory mapping scheme that is R-Mapping and an analog summation scheme. In Section 4.4, we evaluate the CASCADE architecture and compare with ADC-based and SA-based reference architectures for the 10 DNN and 1 RNN benchmarks. In Section 4.5, we conclude this work.

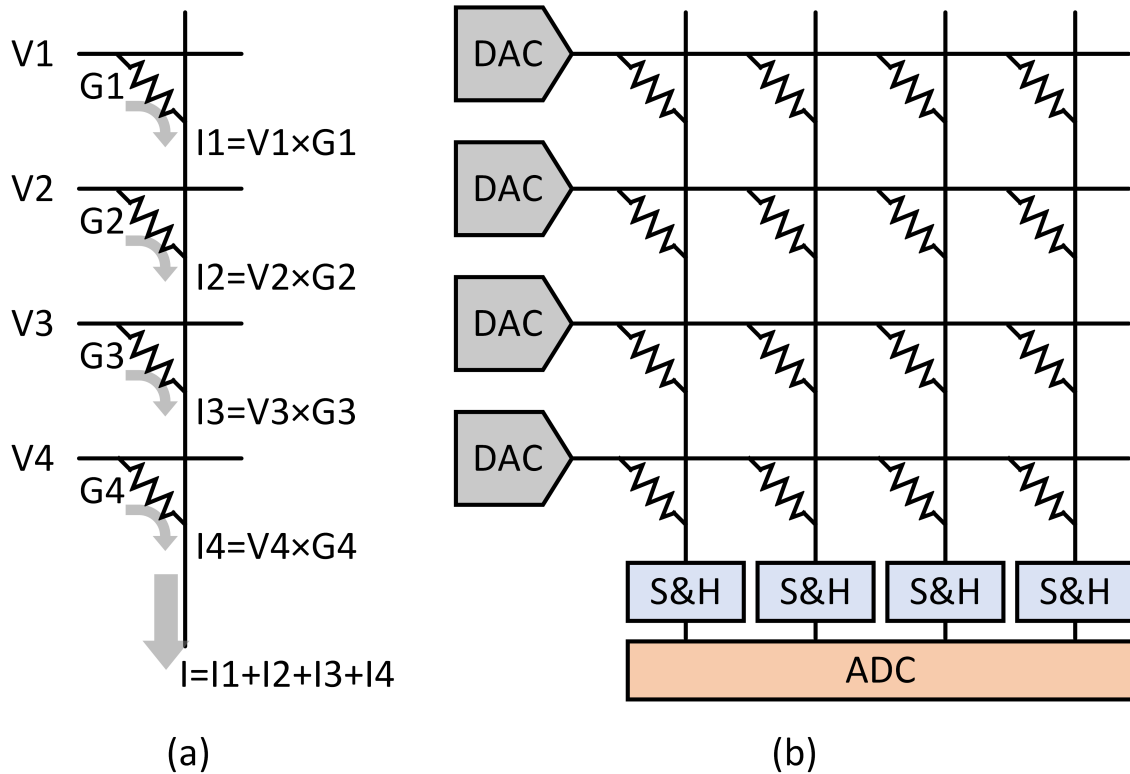


Figure 4.1: (a) A dot product implemented in an RRAM crossbar. (b) Parallel dot products implemented in an RRAM crossbar array. The input vector is converted to voltage pulses by DACs and the weights are stored in the RRAM crossbar. The BL currents are sampled and held (S&H) and converted to digital values.

## 4.2 RRAM-Based PIM

An RRAM cell is a metal-insulator-metal (MIM) device that stores information via its programmable conductance. Typical RRAM devices are constructed in a crossbar array to provide dense storage to fulfill the growing demand of low-power nonvolatile memory. In addition to storage, an RRAM crossbar array can be used to perform parallel dot products.

Figure 4.1 shows examples of a dot product and parallel dot products using RRAM crossbars. In Figure 4.1(b), a 4x4 weight matrix is stored on the RRAM crossbar as conductances. A 1x4 input vector is sent to four DACs to be converted to read voltage pulses and applied to the WLs of the crossbar. A read voltage pulse over

an RRAM cell’s conductance produces a current that represents the product of the voltage input with the conductance. The currents through RRAM cells along a column are aggregated on the BL to complete a dot product.

As illustrated in the above example, a read voltage pulse is applied to a WL that drives a row of RRAM cells. Activating multiple or all of the WLs in the RRAM crossbar enables the second dimension of parallelism. Throughout the dot product operation, one set of operands, e.g., the 4×4 weight matrix in the above example, is kept in memory, saving significant time and energy in data movement.

In addition to dot products, an RRAM device supports accumulation in a write process. Applying consecutive write pulses, i.e., set or reset pulses, to an RRAM cell increases or decreases its conductance [61, 62]. The benefit of in-RRAM accumulation can be significant, as a typical  $n$ -step digital accumulation requires  $n$  reads from memory to fetch the temporary sum, and  $n$  writes to memory to update the sum, which are all eliminated by in-RRAM accumulation.

#### 4.2.1 Workloads and Mapping to RRAM

DNNs and RNNs have emerged to be one of the most important machine learning workloads. A DNN consists of layers of convolution (CONV), pooling, normalization, and fully-connected (FC) layers. CONV and the FC layers are the most computation-intensive and memory-intensive layers.

A CONV layer is shown in Figure 4.2. An input activation sized  $W \times H$  of  $C$  channels is convolved with  $K$  kernels sized  $R \times S$  of  $C$  channels to produce an output fmap sized  $X \times Y$  of  $K$  channels ( $X = W - R + 1$  and  $Y = H - S + 1$ ). The kernels (weights) are learned through a training algorithm. A point  $(x, y, k)$  of the output fmap,  $f^{out}(x, y, k)$ , is calculated as follows:

$$f^{out}(x, y, k) = \sigma\left(\sum_{c=1}^C \sum_{r=1}^R \sum_{s=1}^S f^{in}(x+r, y+s, c) \times w_k(r, s, c)\right), \quad (4.1)$$

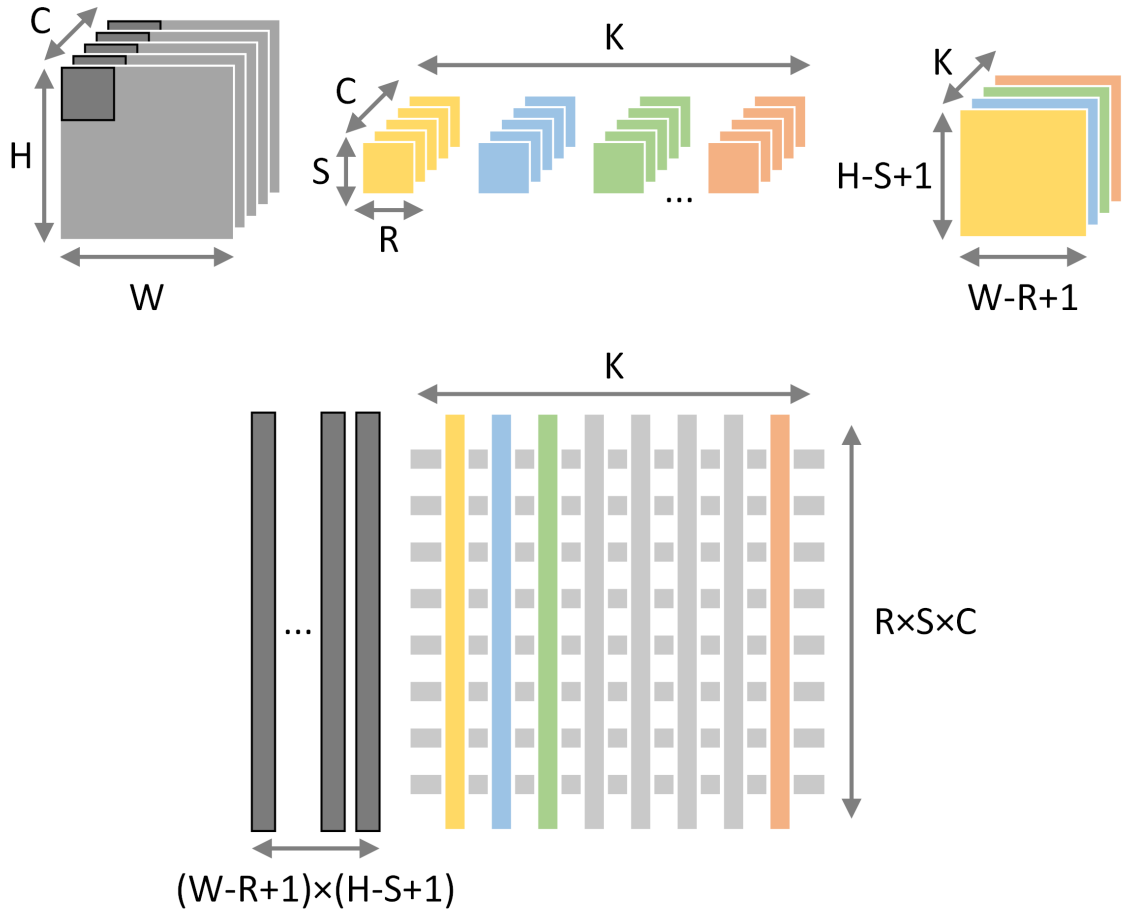


Figure 4.2: Mapping of convolution operation on an RRAM crossbar array.



where  $f^{in}(x, y, c)$  is the input activation at  $(x, y, c)$ ,  $w_k(r, s, c)$  is the weight value of  $k^{\text{th}}$  kernel at  $(r, s, c)$ , and  $\sigma$  is the activation function.

Each elementary step of the computation involves the product of a pair of input activation and weight. To compute each point in the output fmap, (4.1) describes three layers of loops over  $R$ ,  $S$ , and  $C$ , involving accumulation of  $R \times S \times C$  partial sums. To increase throughput, the three layers of loops can be unrolled or partially unrolled. To complete one entire output fmap, there are three additional outer loops over  $X$ ,  $Y$ , and  $K$ .

An FC layer can be viewed as a special case of CONV layer with  $W = R$  and  $H = S$ , i.e., the dimensions of the input activations and kernels are matched. The  $R \times S$  for a FC layer is typically larger than a CONV layer. Since the dimensions of the input activations and weights are matched, an output fmap is sized  $1 \times 1 \times K$ .

To perform inference, the weights  $w$  of a CONV or a FC layer remain static as the input activations  $f^{in}$  are streamed in. Therefore, it is advantageous to store the weights in RRAM, reuse the weights as new input activations are applied. One efficient way of storing weights in an RRAM crossbar array is shown in Figure 4.2, where one  $R \times S \times C$  kernel is stored in  $R \times S \times C$  cells as conductances in one column, and  $K$  kernels are stored across  $K$  columns. The illustration in Figure 4.2 assumes that the RRAM array consists of at least  $R \times S \times C$  rows and  $K$  columns, and each RRAM device provides a sufficient resolution to store a weight value. Following this mapping,  $R \times S \times C$  partial sums are accumulated on one BL to complete the computation of one point in the output fmap.

Practical RRAM arrays may not provide nearly as many rows or columns. The  $R \times S \times C$  of a layer in a state-of-the-art DNN or RNN can easily exceed 10,000. Furthermore, a practical RRAM cell may not provide enough distinguishing levels to store a 16-bit or even an 8-bit weight value. The technology limitation requires a weight value to be stored in multiple RRAM cells, and the  $R \times S \times C$  accumulations to

be separated and performed on multiple BLs of one RRAM array or multiple arrays. Each BL is digitized by an ADC or an SA, and the final accumulation is done in the digital domain.

The underlying computation in an RNN can be mapped in the same manner. For example, consider the long short-term memory (LSTM) [63, 64], a kind of RNN. Given an input sequence  $X = (x_1, x_2, \dots, x_T)$ , where  $x_t$  is the input at time step  $t$ ,  $t \in \{1, \dots, T\}$ , a typical LSTM layer is defined as follow.

$$\text{Input gate : } i_t = \sigma(W_x^i x_t + U_h^i h_{t-1}) \quad (4.2)$$

$$\text{Forget gate : } f_t = \sigma(W_x^f x_t + U_h^f h_{t-1}) \quad (4.3)$$

$$\text{Output gate : } o_t = \sigma(W_x^o x_t + U_h^o h_{t-1}) \quad (4.4)$$

$$\text{Candidate memory : } \tilde{c}_t = \tanh(W_x^c x_t + U_h^c h_{t-1}) \quad (4.5)$$

$$\text{Memory cell : } c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (4.6)$$

$$\text{Hidden state : } h_t = o_t \odot \tanh(c_t), \quad (4.7)$$

where  $W^j$  and  $U^j, j = \{i, f, o, c\}$ , are parameters learned through a training algorithm,  $\odot$  denotes element-wise multiplication,  $\sigma$  is the sigmoid function, and  $\tanh$  is the hyperbolic tangent function. Both  $\sigma$  and  $\tanh$  are element-wise nonlinear activation functions. The computation intensive part is the matrix-vector product in (4.2) to (4.5), which can be rewritten as:

$$\begin{bmatrix} W_x^i & U_x^i \\ W_x^f & U_x^f \\ W_x^o & U_x^o \\ W_x^c & U_x^c \end{bmatrix} \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} \quad (4.8)$$

Using this formulation, the dot products in an LSTM can be implemented in RRAM following the same approach.

	ISAAC [17]	PRIME [18]	PipeLayer [19]	CASCADE
Input bitwidth	16	6 w/ fraction encoding	16	16
Input bits per cycle ( $b_{WL}$ )	1	3	1	1
Weight bitwidth	16	8 w/ fraction encoding	16	16
Cell resolution ( $b_{cell}$ )	2	4	4	1
Array size ( $N_{rows} \times N_{cols}$ )	128×128	256×256	128×128	64×64
BL resolution ( $b_{BL}$ )	9	15	11	7
Output bitwidth	8 w/ encoding	6 w/ truncation	N/A	6 w/ encoding
Output interface	ADC	SA	Spiking integrate and fire	TIA

Table 4.1: In-RRAM MAC architecture comparison.

#### 4.2.2 In-RRAM Computation

ISAAC [17], PRIME [18] and PipeLayer [19] are three recently published architectures for implementing DNN and RNN through in-RRAM computation. There are also examples of in-SRAM computation [44, 45, 46] and in-DRAM computation [65] that follow the same concept. A comparison of key aspects of ISAAC, PRIME and PipeLayer is shown in Table 4.1.

**Row circuitry.** PRIME applies both width- and level-modulation to each WL. A 6-bit input is converted to one of 8 voltage levels over two pulse periods. Providing 8 precise voltage levels is challenging, complicating the DAC circuits. ISAAC streams inputs in a bit-serial fashion: a binary input is sent to the WL driver one bit at a time, and a voltage pulse is produced and passed onto the WL. A 1-bit driver is simpler to design and the read process is also better controlled. PipeLayer also adopts ISAAC’s bit-serial input streaming.

**BL resolution.** In-RRAM computation can produce a high BL resolution. PRIME uses 4 bits per RRAM cell and a 256-row RRAM array, resulting in a BL resolution of 15 bits. The outputs are truncated to 6 bits to be practical. ISAAC uses bit-serial input streaming, stores 2 bits per cell, and uses a 128-row array to reduce the BL resolution to 9 bits. The bit-serial input streaming is also adopted by PipeLayer. However, PipeLayer stores 4 bits per cell, leading to a 11-bit BL resolution.

**Column circuitry.** PRIME uses an SA for each BL. The SA takes up to  $2^n$  cycles to perform  $2^n$  comparisons to produce a  $n$ -bit output, where  $n$  is the output bitwidth. ISAAC uses an 8-bit ADC, which can be costly in terms of power and area. The ADC is shared among 128 BLs in an array to amortize the cost. PipeLayer uses an integrate-and-fire component for each BL to generate spikes. The serial integration is slow and the latency scales with  $2^n$ .

**Physical implementation challenges.** State-of-the-art PIM chips demonstrate the challenges in designing peripheral circuitry to support a high BL resolution. To be realistic, the resolution of inputs and weights are often lowered, and only a subset of WLs are activated. The latest PIM chips, including the ones based on SRAM [44, 45, 46] and the ones based on RRAM [66], chose to digitize only the most significant bits (MSBs) to reduce the cost of A/D conversion. In particular, 1-bit output was used in [44]. Doing so severely limits the application of PIM. In [49], only 9 WLs are activated per column. Since high-resolution PIM can be affected by process, voltage and temperature (PVT) variations [47], it is critical to take variations and noise into account in PIM designs.

### 4.2.3 A/D Conversion for In-RRAM Computation

A/D conversion is an integral part of in-RRAM computation, and it contributes about 60% of the power consumption based on the latest work [17]. Common A/D conversion choices are ADC or SA. An ADC’s complexity and power consumption depend on its sampling rate and resolution. In-RRAM computation has a relatively relaxed sampling rate, due to the intrinsic RC delay of WL and BL propagation. However, in-RRAM computation can require a high resolution that depends on the the resolution of the analog read pulse  $b_{WL}$ , the resolution of RRAM cell  $b_{cell}$ , and the number of rows that are activated in parallel  $N_{rows}$ . With the growing desire of using multi-level cell (MLC) RRAM and a higher degree of parallelism by activating

more rows in parallel, the A/D resolution is constantly raised. Since the area and energy consumption of A/D conversion scale exponentially with the resolution [67, 17], designing the A/D conversion for in-RRAM computation is a main challenge.

An SA is commonly found in the peripheral circuitry of single-level cell (SLC) memories such as SRAM or DRAM. An SA can be viewed as a 1-bit ADC that compares BL voltage with a reference voltage to produce a 1-bit output. An SA can serve as a multi-bit ADC by sweeping the reference voltages, i.e., a reference voltage ramp, and keeping track of when the SA output flips using a counter [18]. The SA circuitry is simple and compact, but using SA for multi-bit A/D conversion can cost a high latency of up to  $2^n$  cycles, where  $n$  is the resolution.

### 4.3 The CASCADE Architecture

The CASCADE in-RRAM computation architecture targets inference in edge/IoT devices with a stringent energy and area envelope. A CASCADE chip is made of analog processing units (APUs) that each consists of a number of RRAM crossbar arrays, as shown in Figure 4.3. An RRAM array can be tasked with performing in-memory dot products, buffering or in-memory accumulation. CASCADE executes a DNN or RNN model layer by layer as in [34]. The trained weights in a layer are first loaded to the APUs from main memory. The weights and inputs are assumed to be 16 bits, and the dot products are quantized to 16 bits.

Compared to ISAAC, PRIME or PipeLayer, CASCADE uses an efficient analog dataflow with a TIA interface at the output of a multiply-accumulate (MAC) RRAM to convert the analog current to voltage. The voltage is applied to a buffer RRAM directly to accomplish partial-sum accumulation. The results are sent to summing amplifiers and ADCs to convert to digital values, followed by activation, normalization and pooling. The outputs are stored in main memory for the next layer of processing. Cascading MAC RRAMs with buffer RRAMs realizes the core computation of a

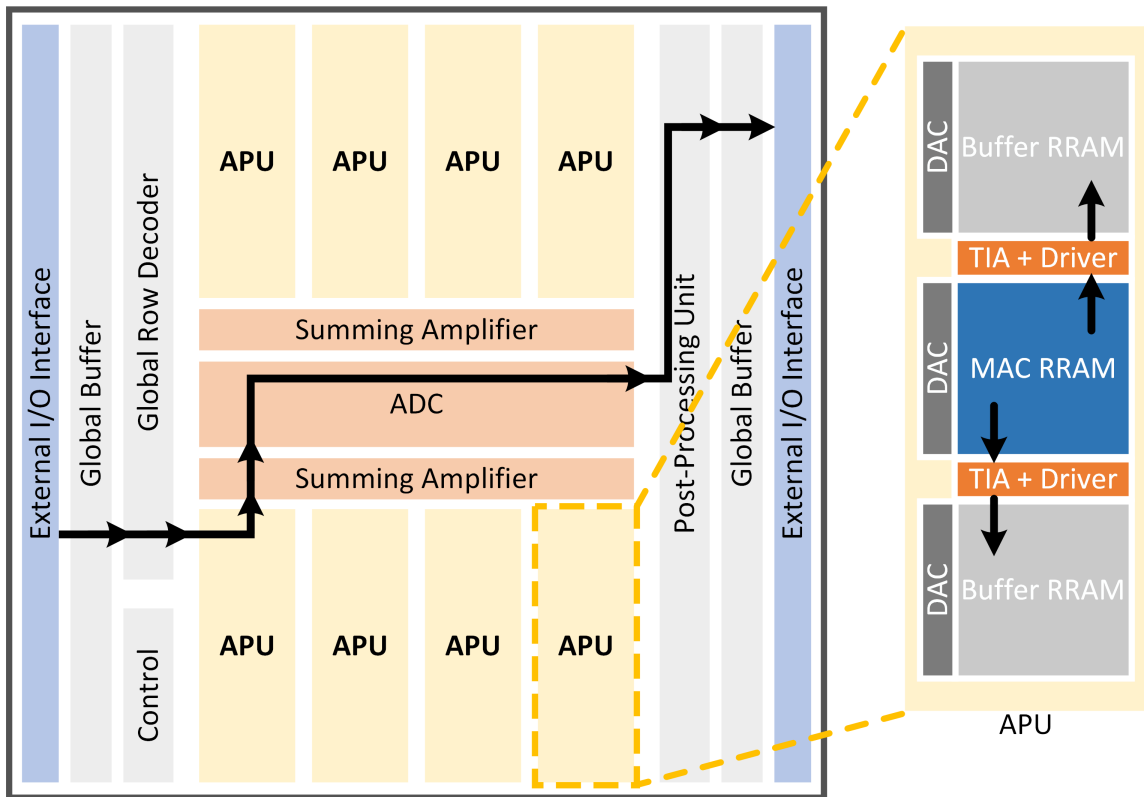


Figure 4.3: Illustration of the CASCADE architecture. The bold lines indicate the proposed dataflow. The zoom-in view shows an analog processing unit (APU).

CONV or FC layer in analog and in RRAM. Since A/D conversion occurs in the very end of the computation, redundant conversions of intermediate values are saved.

### 4.3.1 Input Streaming and Weight Mapping in MAC RRAM

In designing CASCADE, we adopt the bit-serial streaming of input ( $b_{WL} = 1$ ) to a MAC RRAM. Each 16-bit input ( $\eta = 16$ ) is streamed from LSB to MSB as 16 WL pulses. A 1-bit WL driver is simpler to design, more compact and consumes less power than a multi-bit DAC.

The weight values are stored in a MAC RRAM using  $b_{cell}$  bits per cell. A 16-bit weight value ( $\omega = 16$ ) is mapped to  $\omega/b_{cell}$  RRAM cells. To limit the BL resolution and the impact of variation and noise, we use 1-bit weight mapping ( $b_{cell} = 1$ ) and moderate-sized RRAM array of  $64 \times 64$  ( $N_{rows} = 64, N_{cols} = 64$ ). In this way, the BL resolution is kept to 7 bits, lower than all the previous work as shown in Table 4.1. Following the encoding in [17], the BL resolution is further reduced from 7 to 6 bits. With bit-serial input streaming and binary weight mapping in a MAC RRAM, only two voltage references are needed, one for read and one for write, simplifying routing and driver circuitry.

In CASCADE, a 16-bit weight is stored in 16 cells in a row; and a  $64 \times 64$  MAC RRAM stores 4 16-bit weights per row and 256 weights in total. The  $64 \times 64$  MAC RRAM can be effectively divided into 4  $64 \times 16$  subsections, each subsection represents a  $64 \times 1$  16-bit weight vector. The MAC RRAM performs the dot products of a  $1 \times 64$  input bit vector with four  $64 \times 1$  16-bit weight vectors at a time.

### 4.3.2 Buffering of Partial Sums in Buffer RRAM

After in-RRAM dot products, the BLs of the MAC RRAM carry the analog partial sums associated with every bit of the weights. Using bit-serial input streaming, every new input bit vector produces a new set of analog partial sums that need to be aligned

and accumulated.

Partial sum accumulation is also needed due to the mapping of wide dot products on multiple MAC RRAMs. As discussed previously, in the case of large  $R \times S \times C$ , a wide dot product needs to be separated and mapped to multiple MAC RRAMs or one MAC RRAM through time-multiplexing. The partial sums need to be accumulated to obtain the final result.

**Digital Accumulation.** In previous work [17, 18, 19], the accumulation of partial sums is done in the digital domain. The dataflow is illustrated in Figure 4.4, and it follows the steps below for every input bit vector:

1. Convert the BL outputs of a MAC RRAM to digital partial sums using ADCs or SAs;
2. Read out the temporary sums stored in SRAM or registers;
3. Shift and accumulate the partial sums by S+A;
4. Truncate the LSBs of the sum to maintain a given bitwidth;
5. Write back the updated sum to SRAM or registers;

As illustrated in Figure 4.4, for a 16-bit input, the partial-sum accumulation incurs 16 A/D conversions and data movement in and out of SRAM or registers, which significantly worsens the energy efficiency and performance. Some of the A/D conversions are wasteful due to the LSB truncation in Step (4).

**Analog in-RRAM Buffering and Accumulation.** The CASCADE architecture employs analog buffering and in-RRAM accumulation by cascading a MAC RRAM with two buffer RRAMs via TIA interface, as shown in Figure 4.3. The dataflow is illustrated in Figure 4.4, and it follows the steps below for every input bit vector:

1. Convert the BL outputs of MAC RRAM to analog voltages using TIAs;



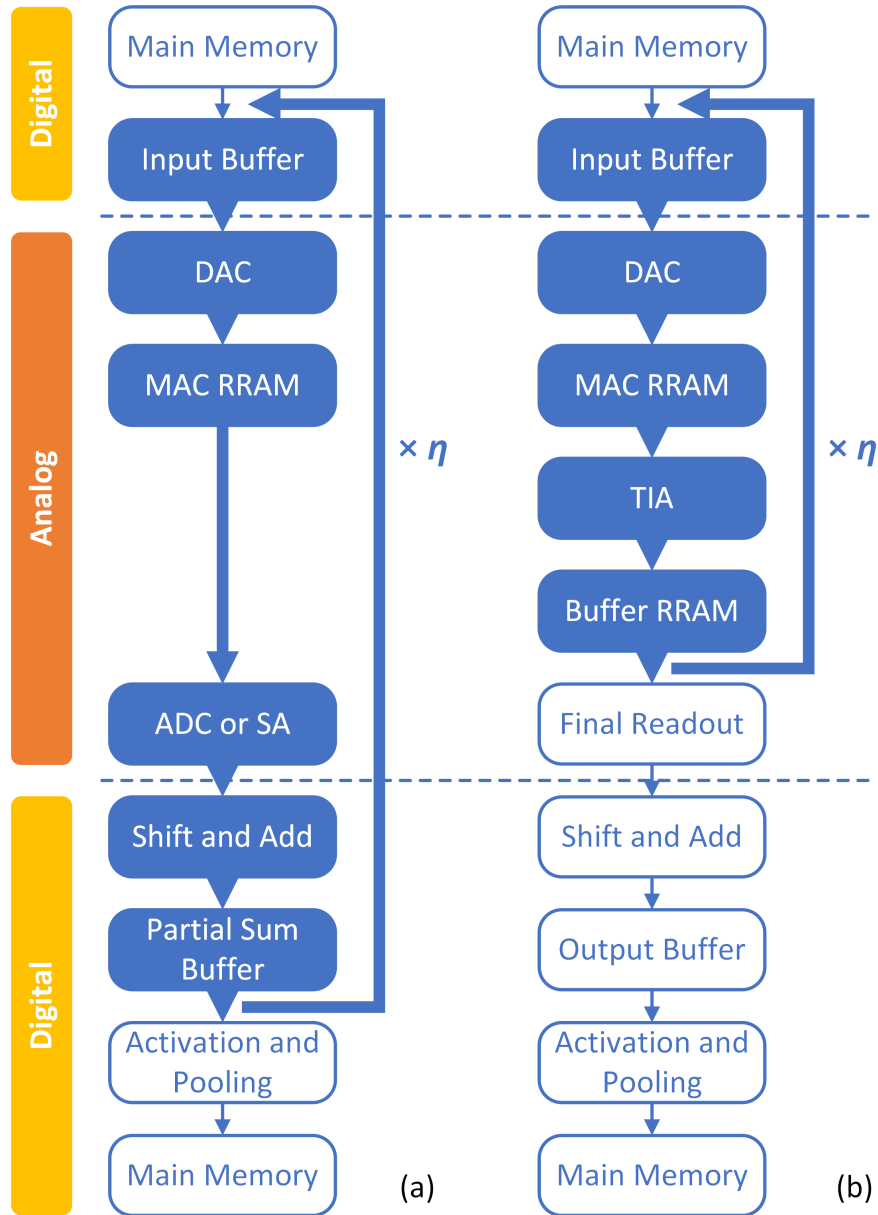


Figure 4.4: Comparison of (a) in-RRAM MAC and digital accumulation of partial sums, and (b) in-RRAM MAC and in-RRAM buffering and accumulation of partial sums. The dashed lines indicate D/A and A/D boundaries. The inner loop is highlighted in blue.

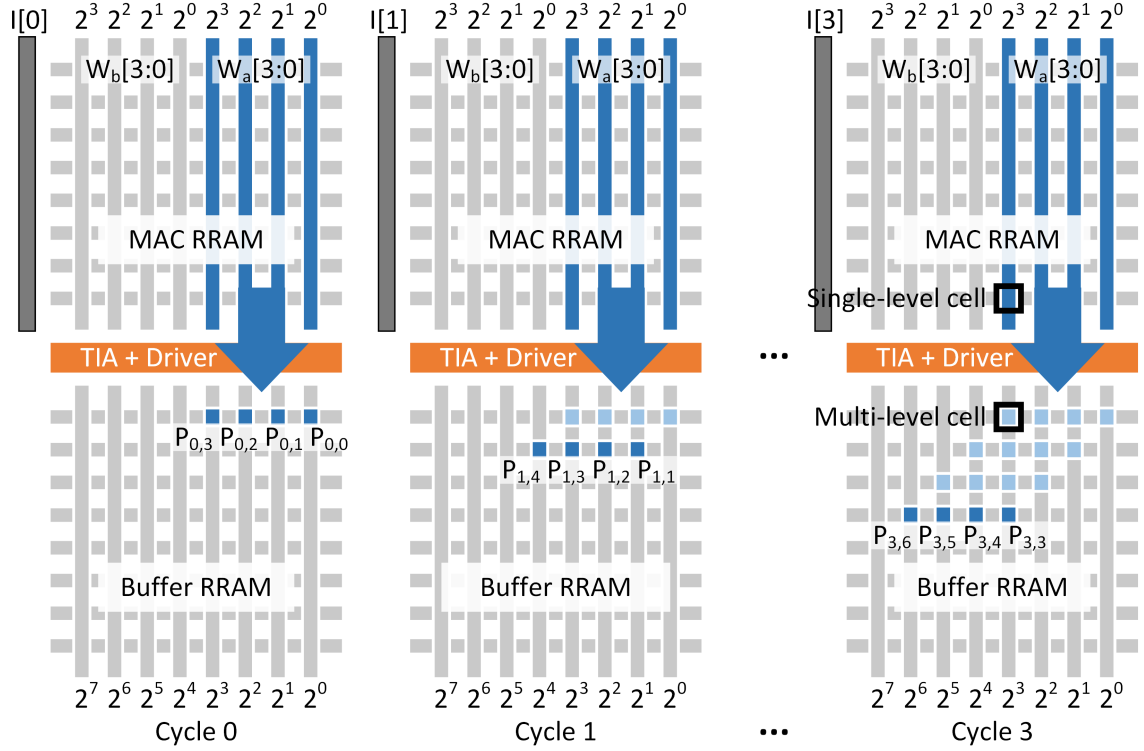


Figure 4.5: Illustration of R-Mapping for 4-bit weights and 4-bit inputs. The inputs are serially streamed in with LSB first. The MAC RRAM stores two sets of 4-bit weights  $W_a$  and  $W_b$ . The arrows indicate the align and store of partial sums through TIAs and input drivers. The partial sum  $P_{i,j}$  is stored in the  $i$ th row and  $j$ th column in cycle  $i$  of the buffer RRAM.

2. Align the voltages as inputs to buffer RRAMs to store the analog partial sums;

Since the MAC RRAM's BL resolution is 6 bits, we propose to use 6-bit MLC RRAM [68] for the buffer RRAMs. After the serial streaming of the 16-bit inputs are complete, the analog partial sums stored in the buffer RRAM are accumulated before the final A/D conversions.

To support in-RRAM accumulation of partial sums, we propose R-Mapping scheme as illustrated in Figure 4.5. Consider LSB-first bit-serial input streaming and a  $64 \times 16$  subsection of a MAC-RRAM that stores a 16-bit weight vector. First, the dot products are computed for the input bit vector 0 and the 16-bit weight vector. The outputs are 16 analog partial sums (one per BL). These 16 analog partial sums are written to a buffer RRAM at address  $i$ . Next, the dot products are computed for the input bit

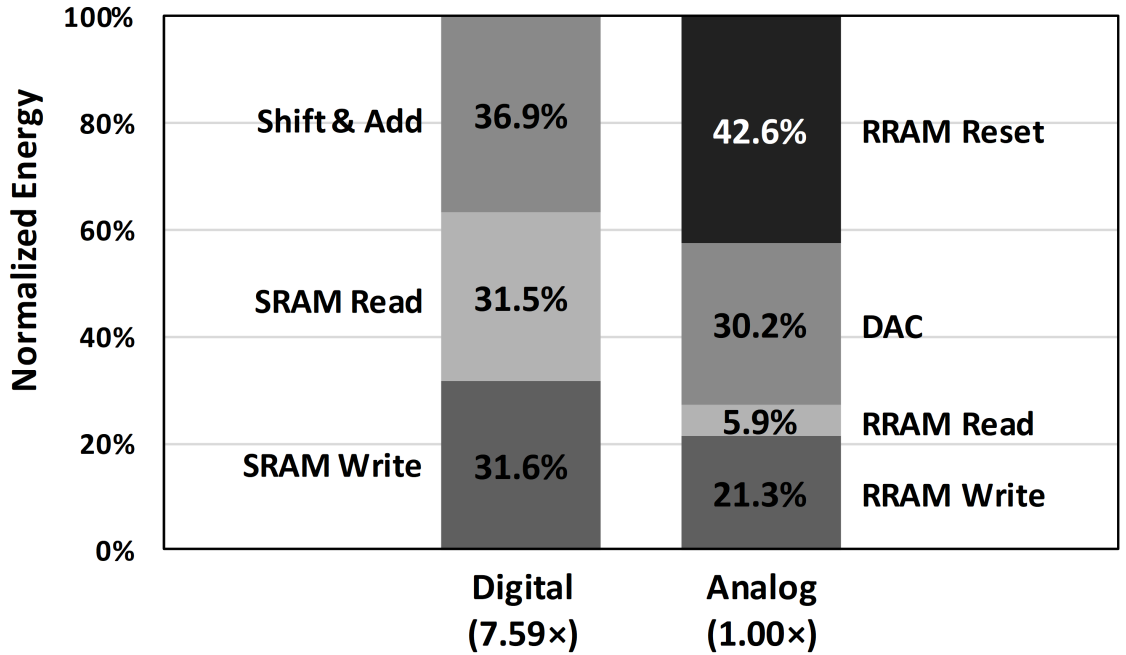


Figure 4.6: Normalized energy consumption of digital partial-sum accumulation and analog in-RRAM partial-sum buffering and accumulation.

vector 1 and the 16-bit weight vector. The outputs of 16 analog partial sums are left-shifted by 1 and written to the buffer RRAM at address  $i + 1$ , as shown in Figure 4.5. At the completion of the bit-serial input streaming, the partial sums are stored in 16 rows and 31 columns of the buffer RRAM (or 15 rows and 30 columns if the inputs are signed numbers). The R-Mapping scheme allows the final accumulation to be done in one read of the buffer RRAM described in Section 4.3.4.

In CASCADE, we assume signed inputs, and the partial sums computed by a  $64 \times 16$  subsection of a MAC RRAM are written to 15 rows and 30 columns of a buffer RRAM. We connect one  $64 \times 64$  MAC RRAM to two  $15 \times 30$  buffer RRAMs, as shown in Figure 4.3, so each buffer RRAM stores the partial sums from two subsections of the MAC RRAM.

Figure 4.6 shows the energy breakdown of the digital accumulation of partial sums and the analog in-RRAM buffering and accumulation. The analog in-RRAM buffering and accumulation is estimated to use  $7.59 \times$  less energy than the digital partial-sum

accumulation. The significant savings are mainly attributed to the elimination of repeated SRAM read and write accesses required by the digital approach.

With analog in-RRAM buffering and accumulation, the A/D conversion is only exercised after the final partial-sum accumulation, as illustrated in Figure 4.4. In addition, the number of A/D conversions is limited to the number bits needed for the final output, eliminating redundant conversions of intermediate values.

**Buffer RRAM Write Consideration.** A standard RRAM write uses a relatively high voltage. It costs high energy and is the primary reason for the limited endurance [69]. In this work, we propose to use a lower voltage to write to buffer RRAMs, and 1T1R RRAM with a transistor to control the write current [70]. These lead to improved endurance and lower energy. Low-voltage write to RRAM can be non-deterministic [71]. In simulation, the errors and variations due to non-deterministic write are incorporated as part of the noise analysis in Section 4.3.5. We allocate one clock period for write to avoid pipeline stalling.

### 4.3.3 TIA Interface between MAC RRAM and Buffer RRAM

A TIA is used to convert an input current to a proportional output voltage. A conventional TIA is constructed using an operational transconductance amplifier (OTA) with a resistor in the feedback connection. The conventional design can be slow in settling, and consume a large on-current. It is also difficult to set the output voltage range appropriate for driving the buffer RRAM.

We design a new TIA circuit as shown in Figure 4.7(a) to convert MAC RRAM’s BL current to voltage, and holds the voltage for driving the buffer RRAM.<sup>1</sup> The TIA circuit operates in two phases: sensing and transfer. In the sensing phase, SW0 and SW1 are closed to enable a feedback loop to convert the BL current to voltage on  $C_{\text{amp}}$ ; SW2 is open and SW3 is closed to detach  $C_{\text{out}}$  and precharge it to  $V_{\text{DD}}$ .

---

<sup>1</sup>The TIA circuit was implemented by Jacob Botimer.

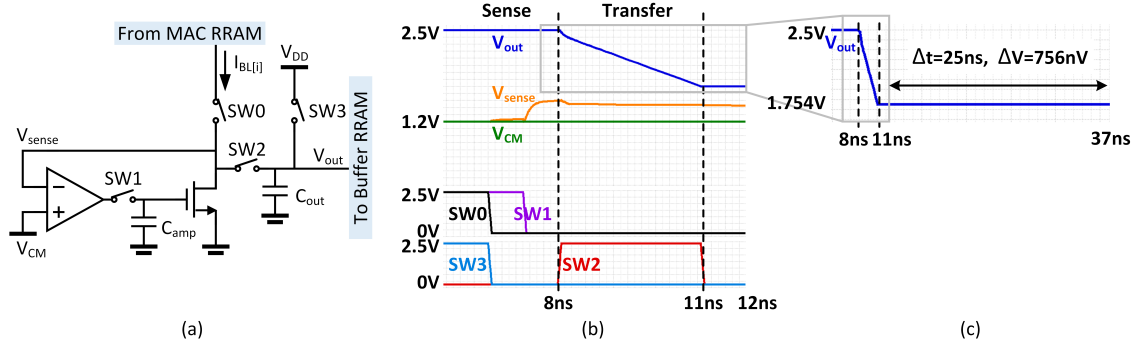


Figure 4.7: (a) Schematic of TIA for converting an input current to a proportional output voltage; (b) simulation waveforms of the TIA designed in a 65nm CMOS technology and the charge leakage of the output capacitor over 25ns of the buffer RRAM write period; and (c) zoom-out view of the TIA sensing and transfer phase.

After the sensing phase, SW0 and SW1 are open to detach the BL from the TIA; and SW3 is open to complete the precharge. In the transfer phase, SW2 is closed to allow the sampled voltage on  $C_{amp}$  to drive the NMOS to reproduce the BL current to discharge  $C_{out}$ . After the transfer is complete, SW2 opens and the voltage is held on  $C_{out}$  for driving the buffer RRAM.

Compared to the conventional TIA design, the proposed TIA offers a faster settling time, lower energy per conversion, and flexibility in setting the output voltage range for driving the next stage. The TIA circuit is designed and simulated as shown in Figure 4.7(b) to obtain realistic parameters for system evaluations. We used metal-oxide-metal capacitor for  $C_{out}$ . The simulation shows that the leakage on  $C_{out}$  results in a negligible voltage drop of 756nV over the 25ns RRAM write period as shown in Figure 4.7(c).

#### 4.3.4 Final Accumulation and A/D Conversion

Following the R-Mapping scheme, analog partial sums from one subsection of a MAC SRAM are stored in 15 rows and 30 columns of a buffer RRAM. After summed together, 10 A/D conversions are needed with resolutions ranging from 6 to 10 bits.

The final accumulation is illustrated in Figure 4.8. The 30 BLs can be divided

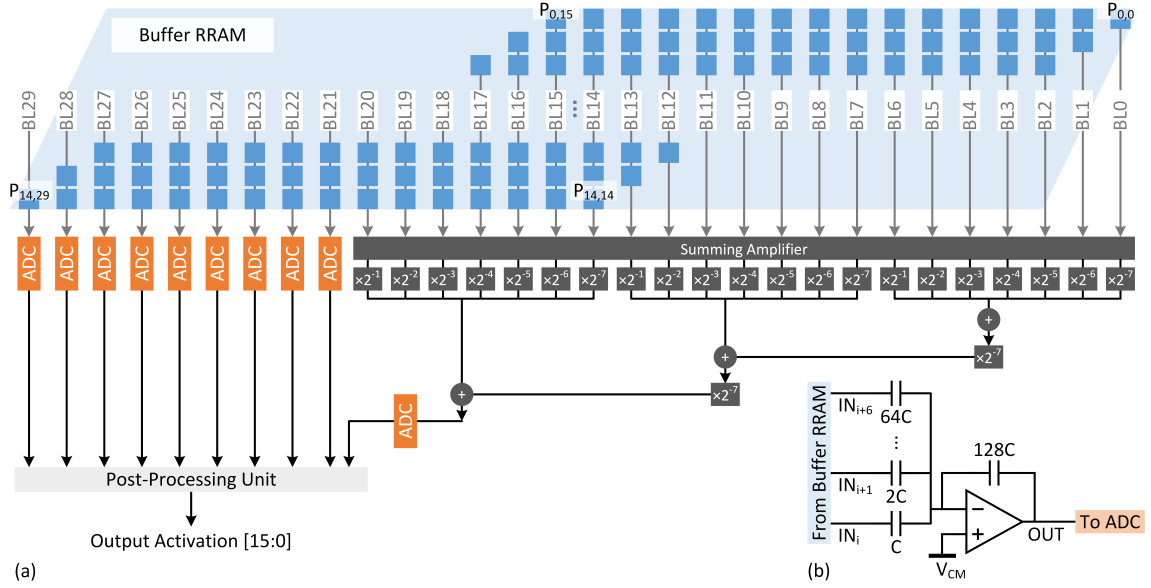


Figure 4.8: (a) Illustration of the final accumulation; and (b) schematic of a summing amplifier.

into groups to efficiently obtain the output of a required resolution. For example, if a 16-bit output is required, the 9 BLs in the MSB group directly contribute to the required 16-bit output resolution, and they are digitized by suitable 6-bit to 10-bit ADCs. The BLs in the LSB group are connected to analog summing amplifiers as shown in in Figure 4.8(b), with each BL current appropriately scaled before summing. The analog sum of a low-order group is fed as the input to the next high-order group. In this way, the low-order groups are compressed to one carry-in to the MSB group. The digital values are then added together to produce the final sum.

With the analog accumulation scheme, the number of A/D conversions is reduced, and the number of digital summations is also reduced. The low-order accumulations are done more efficiently in the analog domain. Although analog accumulation can be less precise than digital accumulation, it produces only a carry-in to the MSB group and the imprecision becomes negligible. With fewer A/D conversions, we can choose an ADC of a lower sampling frequency to reduce the energy per conversion step [3]. Fewer number of A/D conversions also makes it possible to share ADCs to reduce

area.

### 4.3.5 Noise Tolerance

Since the CASCADE architecture connects MAC RRAMs with buffer RRAMs, and relies on the analog dataflow from MAC RRAMs to buffer RRAMs, it is critical to check the variation and noise tolerance of the end-to-end system. Analytically, we can lump the variation, noise and non-idealities of analog circuits as effective noise on the MAC RRAM BL, and measure the signal-to-noise ratio (SNR). The SNR affects the classification accuracy as shown in Figure 4.9. In this example, we used a 2-layer MLP as the workload. Different system configurations require different levels of SNR. A higher SNR means a lower margin for noise tolerance.

Suppose we aim at a 90% classification accuracy, a 6-bit BL resolution (e.g., 1 input bit/cycle, 1-bit MAC RRAM cell, and 64-row MAC RRAM as in CASCADE) requires a minimum SNR of 25 dB, while a 11-bit BL resolution (e.g., 1 input bit/cycle, 4-bit MAC RRAM cell, and 128-row MAC RRAM as in PipeLayer [19]) requires a minimum SNR of 35 dB. The noise tolerance of the PipeLayer configuration is 10 dB lower than the CASCADE configuration. The CASCADE architecture adopts a 6-bit BL resolution to ensure the robustness of the end-to-end analog and in-memory computation.

## 4.4 Evaluation

We first establish the reference architectures based on ISAAC and PRIME. We then provide an exploration of the CASCADE design space to show the capabilities of the architecture as well as its limitations. Finally, an instance of the CASCADE architecture is evaluated using realistic workloads with comparisons made against the references.

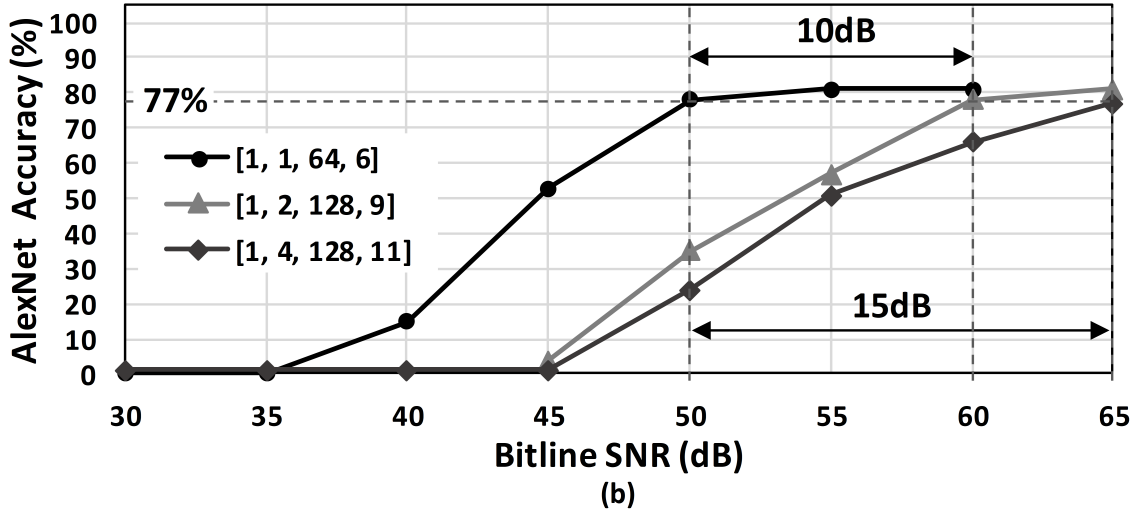
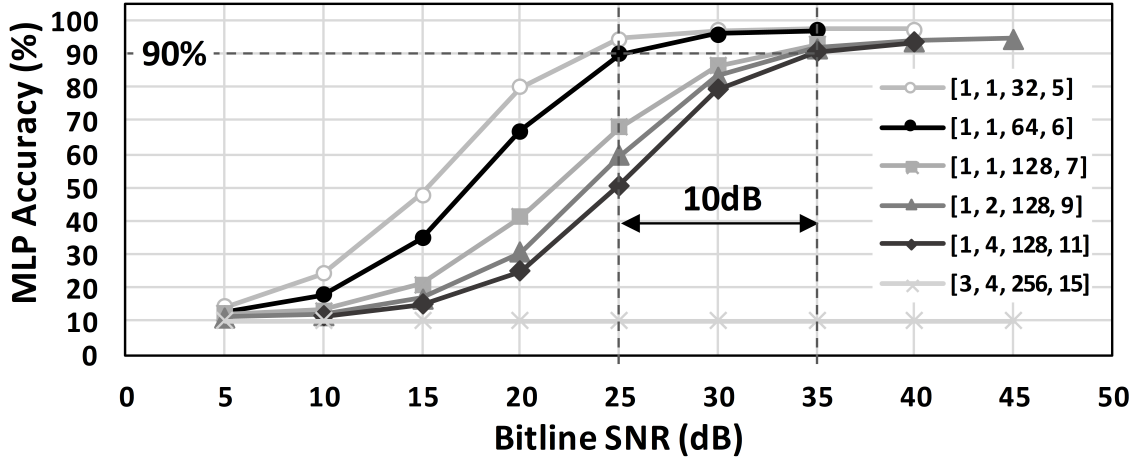


Figure 4.9: (a) Two-layer MLP classification accuracy for different configurations noted by  $[b_{WL}, b_{cell}, N_{rows}, b_{BL}]$ , and (b) AlexNet top-5 classification accuracy for CASCADE and configurations based on ISAAC and PipeLayer.



	CASCADE	ADC-based	SA-based
In-RRAM dot product	Input: 16 bits, bit-serial streaming Weight: 16 bits, binary mapping RRAM array size: $64 \times 64$		
Partial-sum accumulation	Analog in-RRAM	Digital sum SRAM	Digital sum SRAM
A/D resolution	10 bits	6 bits	6 bits
A/D activity	10 times per 16 cycles	256 times per 16 cycles	256 times per 16 cycles
A/D normalized latency	1	1	$2^6$
Number of ADCs	7 ADCs per 80 arrays	80 ADCs per 80 arrays	$80 \times 64$ SAs per 80 arrays

Table 4.2: Configurations of the CASCADE, ADC-based and SA-based reference architectures.

#### 4.4.1 Methodology

**Reference Architectures.** We use two reference architectures: 1) an ADC-based architecture adapted from ISAAC [17] and an SA-based architecture adapted from PRIME [18]. To make a fair comparison, the architectures all employ RRAM crossbar arrays of the same size, and all utilize bit-serial input streaming and binary weight mapping.

Table 4.2 summarizes the three architectures for comparison. The key difference is that CASCADE performs in-RRAM buffering and accumulation, while the two reference architectures perform digital accumulation after converting the analog partial sums using ADCs and SAs. With an efficient TIA interface and analog buffering and accumulation, CASCADE reduces the number of A/D conversions from 256 per 16 cycles to 10 per 16 cycles.

**Component Models.** Our evaluations were done using a 65nm technology and a 65nm RRAM model from [49]. The SRAM model is constructed based on the results obtained from a memory compiler. We adopted most of the circuit component models from ISAAC [17] and scaled them to 65nm. The analog components including ADC, SA, summing amplifier and S&H were obtained from recent literature and scaled to

AlexNet	VGG-A	VGG-B	VGG-C	MSRA-A	MSRA-B	MSRA-C	DeepFace	NeuralTalk
$11 \times 11, 96/4$ (1)	$3 \times 3, 64$ (1)	$3 \times 3, 64$ (2)	$3 \times 3, 64$ (2)	$7 \times 7, 96/2$ (1)	$7 \times 7, 96/2$ (1)	$7 \times 7, 96/2$ (1)	$11 \times 11, 32$ (1)	FC-2400
$5 \times 5, 256$ (1)	$3 \times 3, 128$ (1)	$3 \times 3, 128$ (2)	$3 \times 3, 128$ (2)				$9 \times 9, 16/2$ (1)	FC-8791
$3 \times 3, 384$ (1)	$3 \times 3, 256$ (2)	$3 \times 3, 256$ (3)	$3 \times 3, 256$ (4)	$3 \times 3, 256$ (5)	$3 \times 3, 256$ (6)	$3 \times 3, 384$ (6)	$9 \times 9, 16$ (1)	
$3 \times 3, 384$ (1)	$3 \times 3, 512$ (2)	$3 \times 3, 512$ (3)	$3 \times 3, 512$ (4)	$3 \times 3, 512$ (5)	$3 \times 3, 512$ (6)	$3 \times 3, 768$ (6)	$7 \times 7, 16/2$ (1)	
$3 \times 3, 256$ (1)	$3 \times 3, 512$ (2)	$3 \times 3, 512$ (3)	$3 \times 3, 512$ (4)	$3 \times 3, 512$ (5)	$3 \times 3, 512$ (6)	$3 \times 3, 896$ (6)	$5 \times 5, 16$ (1)	
							FC-4096	
							FC-4096	
							FC-4030	

Table 4.3: Benchmarks for evaluation. Convolution layers are denoted as  $R \times S, K/D$  ( $L$ ), where  $R, S$  and  $K$  correspond to the notations used in Figure 4.2, and  $L$  denotes the number of such layers.

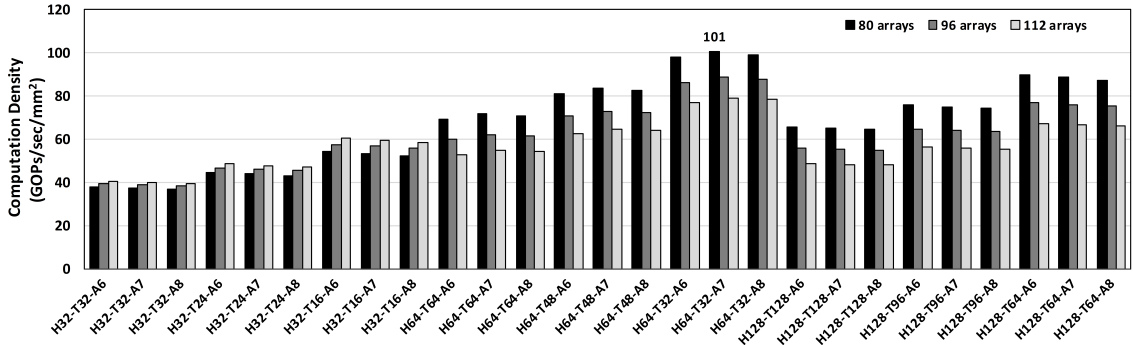


Figure 4.10: Computation density across the design space. A notation, e.g., H64-T32-A7 R80, represents 80  $64 \times 64$  RRAM arrays with 32 TIAs per array and 7 ADCs shared by the arrays.

65nm. In particular, we used the successive approximation (SAR) ADC from [72], same as in ISAAC. The area and energy of a SAR ADC and the resolution scaling follow [67]. The SA model was adapted from [73]. The summing amplifier was from [74]. The TIA was designed in a 65nm CMOS technology and simulated in Cadence Spectre to obtain power, latency and variation.

**Benchmarks.** We used 11 benchmarks including 10 DNNs and 1 RNN to evaluate the CASCADE architecture and compare it with the references. The details of the DNNs and RNN are listed in Table 4.3. We used ImageNet image classification dataset for AlexNet [59], ResNet [75], 3 types of VGG [76], GoogLeNet [60], and 3 types of MSRA [77], face recognition for DeepFace [78], and image captioning for NeuralTalk [79].

#### 4.4.2 CASCADE Design Space Exploration

The CASCADE architecture is parameterized by 4 variables: 1) the size of the RRAM array  $H \times H$ , simply denoted by  $H$ ; 2) the number of RRAM arrays, denoted by  $R$ ; 3) the number of TIAs per array, denoted by  $T$ ; and 4) the number of ADCs, denoted by  $A$ . We assume that the total weight storage capacity is  $40\text{KB} \times 80$  blocks = 3.2MB and a DDR4 I/O bandwidth of 25.6GB/s between the CASCADE chip and external memory.

The computation density measured in GOPs/s/mm<sup>2</sup> is shown in Figure 4.10. In general, using larger RRAM arrays provides a higher computation density due to the more dot products and accumulations that can be performed in RRAM arrays at the same time. However, the larger the array size, the higher the I/O bandwidth, the ADC resolution, and the cost of interface circuitry, including S&Hs, TIAs, summing amplifiers, and ADCs. The optimal number of RRAM arrays in one APU and the optimal number of APUs are limited by the I/O bandwidth. The peak performance of 101 GOPs/s/mm<sup>2</sup> can be achieved by 80  $64 \times 64$  RRAM arrays, 32 TIAs per array

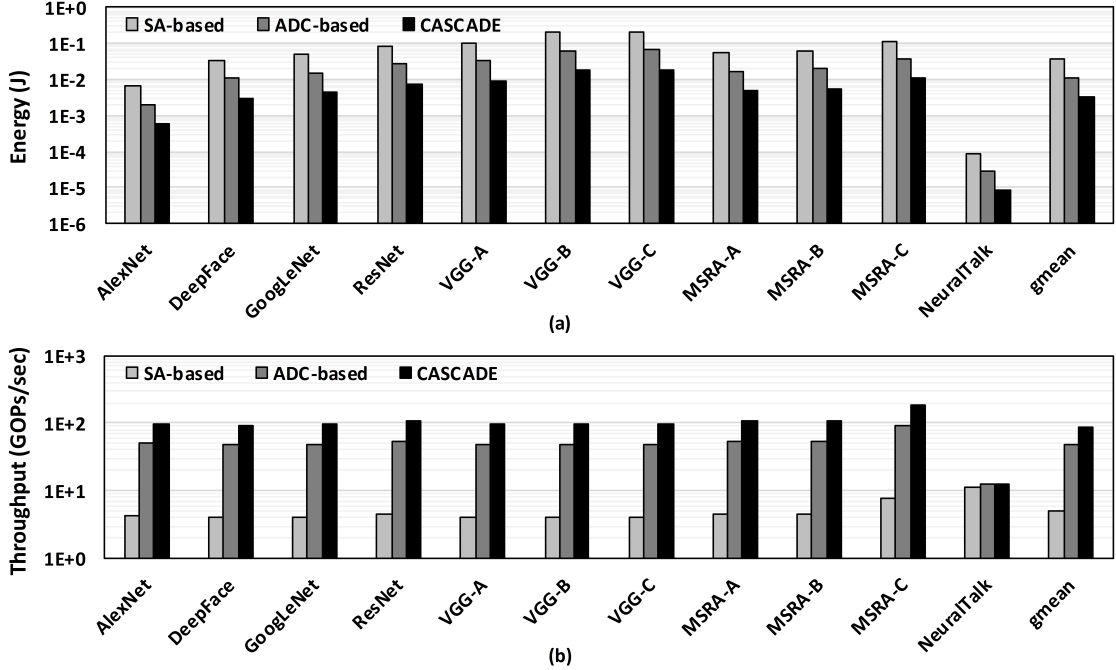


Figure 4.11: (a) Energy consumption of CASCADE compared to reference architectures running DNN and RNN benchmarks; (b) Throughput of CASCADE compared to reference architectures running DNN and RNN benchmarks.

and 7 central ADCs (denoted by H64-T32-A7 R80).

#### 4.4.3 Performance and Energy Consumption

We use 80 APU blocks to evaluate the energy and performance for comparison with the references. Each APU block contains 80  $64 \times 64$  RRAM arrays with 32 TIAs per array and 7 ADCs shared by the arrays.

Figure 4.11(a) shows the energy consumption of CASCADE compared to the two reference architectures for the 10 DNN and 1 RNN benchmarks. The CASCADE architecture achieves an average  $3.5 \times$  lower energy than the ADC-based architecture and  $11.0 \times$  lower energy than the SA-based architecture across all benchmarks. Figure 4.12 shows the energy breakdown for the three architectures to shed light on the competitive advantages of CASCADE. The input buffer and in-RRAM dot products consume the same amount of energy across all three architectures. However, CAS-

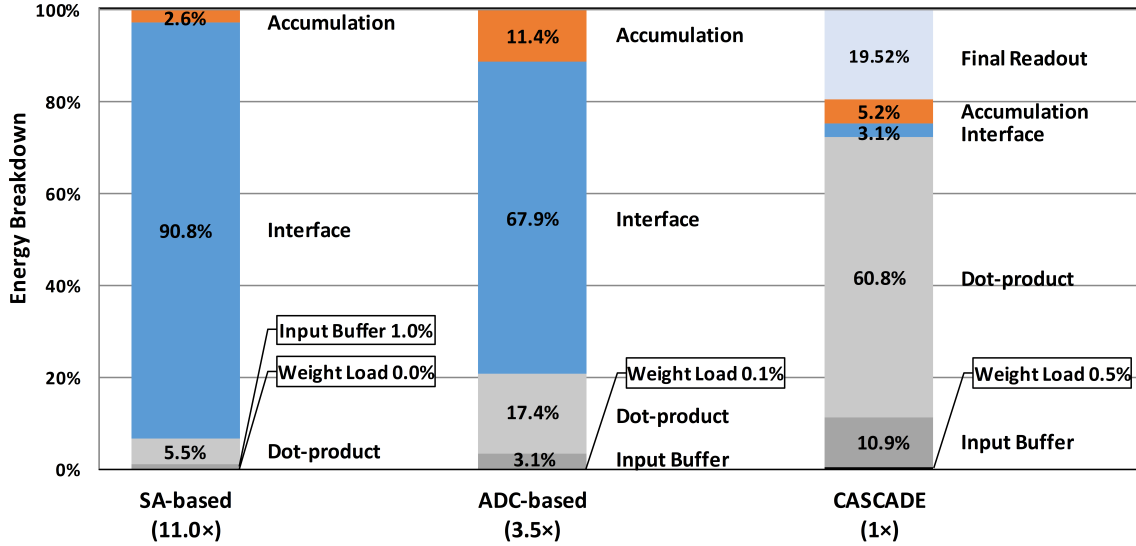


Figure 4.12: Energy breakdown of CASCADE and two reference architectures.

CASCADE's TIA interface consumes  $77.5\times$  lower energy than the ADC interface and  $325.4\times$  lower energy than the SA interface. The latter is due to the long latency of the SA-based A/D conversion.

Figure 4.11(b) shows the throughput of CASCADE compared to the two reference architectures for the 10 DNN and 1 RNN benchmarks. In average, the CASCADE architecture achieves  $1.86\times$  higher throughput than the ADC-based architecture, and  $17.83\times$  higher throughput than the SA-based architecture due to the long latency of A/D conversion.

In summary, CASCADE improves upon the ADC-based architecture in energy. As an example of the ADC-based architecture, ISAAC [17] has already demonstrated improvements of  $14.8\times$ ,  $5.5\times$ , and  $7.5\times$  in throughput, energy, and computation density over DaDianNao [34]. The 64-chip DaDianNao has demonstrated  $450.65\times$  speedup and  $150.31\times$  lower energy than an NVIDIA K20M GPU. Therefore, we expect that the benefits of CASCADE will be on top of the previously demonstrated gains over an ASIC chip or a GPU.

#### 4.4.4 Extension to Spiking Neural Networks

The CASCADE architecture can be adapted to support spiking neural networks (SNNs). The TIA output capacitor can be used as the integration capacitor. A comparator can be added to generate spikes if the voltage on the integration capacitor exceeds a threshold, following the approach in [80, 81, 82, 83]. To implement a SNN, we only need the MAC RRAMs, TIAs, and additional comparators. The buffer RRAMs can be bypassed.

### 4.5 Summary

This work presents CASCADE, an architecture that connects MAC RRAMs for computing dot products with buffer RRAMs for in-RRAM buffering and accumulating partial sums through an efficient TIA interface. Dot products and partial-sum accumulations are the essential operations for implementing a DNN or RNN. Keeping both parts in RRAM and in analog ensures a high energy efficiency by removing the overhead of A/D conversion and digital accumulation. We demonstrate a new R-Mapping scheme to efficiently accumulate partial sums, and an analog summation approach to bypass the A/D conversions of low-order bits. As a result, the CASCADE architecture minimizes the number of A/D conversions and keeps the A/D conversions at the very end of the entire computation. The CASCADE architecture is pipelined to achieve a high performance, and it consumes  $3.5\times$  lower energy than an ADC-based in-RRAM computation architecture in processing DNN and RNN workloads. Built on realistic RRAM technology constraints, the CASCADE architecture offers a higher SNR margin for variation and noise tolerance while keeping a light-weight CMOS periphery circuitry.

## CHAPTER V

### Conclusion

The dissertation presents the design techniques for accelerating machine learning applications with efficiency and scalability. The performance gap between memory and processor results in the memory bottleneck and it is becoming more difficult especially for DNN accelerators with a large amount of data traveling between processing and memory unit. With the approaching to the semiconductor scaling limit, it is challenging because of the high cost and more design time for a chip in 2nm and beyond. In order to address these questions, we studied deep neural networks and perception algorithms and optimized the processing in algorithm, architecture, and circuit perspectives. The proposed two architectures and one silicon prototype were implemented to demonstrate the performance.

The first NetFlex chip is implemented to support CNN-based perception flexibly using the 2.5D technology. The reconfigurable PE designs, process scheduling, and memory mapping allow the chip to connect and form a ring topology for streaming processing. The optimized dataflow for deconvolution operations and skipping module for gating the chiplets provides more energy saving to the system. NetFlex is measured in the 22nm technology and achieves a high throughput for perception workloads. The NetFlex 4-chiplet system demonstrates high energy and area efficiency as well as design scalability.

The second AR-PIM architecture is presented to utilize the SRAM in a 7nm process and exploit the low effective BL range considering the bit-level sparsity of weight and dynamic input activation jointly for DNN. The runtime BL density detection mechanism is proposed to adapt to the low effective BL range. The design reduces the energy portions of ADCs and enhances the sensing margin and capability of variation tolerance. AR-PIM architecture is evaluated and improves energy efficiency and area efficiency.

The third CACSCADE architecture is proposed to use the RRAM-based PIM approach for DNN or RNN to reduce the data movement. The extended analog dataflow through an efficient TIA minimizes the number of A/D conversions and keeps the dot products and partial-sum accumulation in RRAM. CASCAD E architecture is evaluated using a 65nm technology and achieves higher energy efficiency.



## BIBLIOGRAPHY

## BIBLIOGRAPHY

- [1] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018.
- [2] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. A case for intelligent RAM. *IEEE Micro*, 17(2):34–44, 1997.
- [3] Boris Murmann. ADC Performance Survey 1997-2021 (ISSCC & VLSI Symposium). [Online]. Available: <http://web.stanford.edu/~murmam/adcsurvey.html>, 2021.
- [4] Chester Liu, Jacob Botimer, and Zhengya Zhang. A 256Gb/s/mm-shoreline AIB-compatible 16nm FinFET CMOS chiplet for 2.5D integration with Stratix 10 FPGA on EMIB and tiling on silicon interposer. In *IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–2, 2021.
- [5] Brian Zimmer, Rangharajan Venkatesan, Yakun Sophia Shao, Jason Clemons, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. A 0.11 pJ/Op, 0.32-128 TOPS, scalable multi-chip-module-based deep neural network accelerator with ground-reference signaling in 16nm. In *IEEE Symposium on VLSI Circuits*, pages 1–2, 2019.
- [6] Massimo Giordano, Kartik Prabhu, Kalhan Koul, Robert M Radway, Albert Gural, Rohan Doshi, Zainab F Khan, John W Kustin, Timothy Liu, Gregorio B Lopes, et al. Chimera: A 0.92 TOPS, 2.2 TOPS/W edge AI accelerator with 2 MBbyte on-chip foundry resistive RAM for efficient training and inference. In *IEEE Symposium on VLSI Circuits*, pages 1–2, 2021.
- [7] Christoforos Kanellakis and George Nikolakopoulos. Survey on computer vision for UAVs: Current developments and trends. *Journal of Intelligent & Robotic Systems*, 87(1):141–168, 2017.
- [8] Dimitris Chatzopoulos, Carlos Bermejo, Zhanpeng Huang, and Pan Hui. Mobile augmented reality survey: From where we are to where we go. *IEEE Access*, 5:6917–6950, 2017.

- [9] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European conference on computer vision*, pages 834–849. Springer, 2014.
- [10] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.
- [11] Ronald Clark, Sen Wang, Hongkai Wen, Andrew Markham, and Niki Trigoni. VINet: Visual-inertial odometry as a sequence-to-sequence learning problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.
- [12] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. DeepVO: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2043–2050, 2017.
- [13] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6612–6619, 2017.
- [14] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6243–6252, 2017.
- [15] W Chris Chen, Clark Hu, KC Ting, Vincent Wei, TH Yu, SY Huang, VCY Chang, CT Wang, SY Hou, CH Wu, et al. Wafer level integration of an advanced logic-memory system through 2<sup>nd</sup> generation CoWoS® technology. In *IEEE Symposium on VLSI Technology*, pages T54–T55, 2017.
- [16] Noah Beck, Sean White, Milam Paraschou, and Samuel Naffziger. “Zeppelin”: An SoC for multichip architectures. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 40–42, 2018.
- [17] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *Proceedings of the International Symposium on Computer Architecture*, pages 14–26, 2016.
- [18] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *Proceedings of the International Symposium on Computer Architecture*, pages 27–39, 2016.
- [19] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. PipeLayer: A pipelined ReRAM-based accelerator for deep learning. In *IEEE International Symposium on High Performance Computer Architecture*, pages 541–552, 2017.

- [20] Amr Suleiman, Zhengdong Zhang, Luca Carlone, Sertac Karaman, and Vivienne Sze. Navion: A 2-mW fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones. *IEEE Journal of Solid-State Circuits*, 54(4):1106–1119, 2019.
- [21] Ziyun Li, Yu Chen, Luyao Gong, Lu Liu, Dennis Sylvester, David Blaauw, and Hun-Seok Kim. An 879GOPS 243mW 80fps VGA fully visual CNN-SLAM processor for wide-range autonomous exploration. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 134–136, 2019.
- [22] Nikolaus Mayer, Eddy Ilg, Philip Häusser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4040–4048, 2016.
- [23] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.
- [24] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.
- [25] Amir Yazdanbakhsh, Kambiz Samadi, Nam Sung Kim, and Hadi Esmaeilzadeh. GANAX: A unified MIMD-SIMD acceleration for generative adversarial networks. In *Proceedings of the International Symposium on Computer Architecture*, pages 650–661, 2018.
- [26] Koen Goetschalckx and Marian Verhelst. DepFiN: A 12nm, 3.8 TOPs depth-first CNN processor for high res. image processing. In *IEEE Symposium on VLSI Circuits*, pages 1–2, 2021.
- [27] Mihai D Rotaru, Wei Tang, Dutta Rahul, and Zhengya Zhang. Design and Development of High Density Fan-Out Wafer Level Package (HD-FOWLP) for Deep Neural Network (DNN) Chiplet Accelerators using Advanced Interface Bus (AIB). In *IEEE Electronic Components and Technology Conference (ECTC)*, pages 1258–1263, 2021.
- [28] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3642–3649, 2012.
- [29] NVIDIA Grace CPU. [Online]. Available: <https://www.nvidia.com/en-us/data-center/grace-cpu/>, 2021.
- [30] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi

- Ghandi, et al. A configurable cloud-scale DNN processor for real-time AI. In *Proceedings of the International Symposium on Computer Architecture*, pages 1–14, 2018.
- [31] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the International Symposium on Computer Architecture*, pages 1–12, 2017.
- [32] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 269–284, 2014.
- [33] Y.-H. Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the International Symposium on Computer Architecture*, pages 367–379, 2016.
- [34] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. DaDianNao: A Machine-Learning Supercomputer. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 609–622, 2014.
- [35] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the International Symposium on Computer Architecture*, pages 243–254, 2016.
- [36] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the International Symposium on Computer Architecture*, pages 27–40, 2017.

- [37] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in Neural Information Processing Systems*, 28, 2015.
- [38] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29:2074–2082, 2016.
- [39] Pulkit Jain, Umut Arslan, Meenakshi Sekhar, Blake C Lin, Liqiong Wei, Tanaya Sahu, Juan Alzate-vinasco, Ajay Vangapaty, Mesut Meterelliyo, Nathan Strutt, et al. A 3.6 Mb 10.1 Mb/mm<sup>2</sup> embedded non-volatile ReRAM macro in 22nm FinFET technology with adaptive forming/set/reset schemes yielding down to 0.5 V with sensing time of 5ns at 0.7 V. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 212–214, 2019.
- [40] Liqiong Wei, Juan G Alzate, Umut Arslan, Justin Brockman, Nilanjan Das, Kevin Fischer, Tahir Ghani, Oleg Golonzka, Patrick Hentges, Rawshan Jahan, et al. A 7Mb STT-MRAM in 22FFL FinFET technology with 4ns read sensing time at 0.9 V using write-verify-write scheme and offset-cancellation sensing technique. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 214–216, 2019.
- [41] Yu-Der Chih, Yi-Chun Shih, Chia-Fu Lee, Yen-An Chang, Po-Hao Lee, Hon-Jarn Lin, Yu-Lin Chen, Chieh-Pu Lo, Meng-Chun Shih, Kuei-Hung Shen, et al. A 22nm 32Mb embedded STT-MRAM with 10ns read speed, 1M cycle write endurance, 10 years retention at 150°C and high immunity to magnetic field interference. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 222–224, 2020.
- [42] Chung-Cheng Chou, Zheng-Jun Lin, Chien-An Lai, Chin-I Su, Pei-Ling Tseng, Wei-Chi Chen, Wu-Chin Tsai, Wen-Ting Chu, Tong-Chern Ong, Harry Chuang, et al. A 22nm 96KX144 RRAM macro with a self-tracking reference and a low ripple charge pump to achieve a configurable read window and a wide operating voltage range. In *IEEE Symposium on VLSI Circuits*, pages 1–2, 2020.
- [43] Qing Dong, Mahmut E Sinangil, Burak Erbagci, Dar Sun, Win-San Khwa, Hung-Jen Liao, Yih Wang, and Jonathan Chang. A 351TOPS/W and 372.4 GOPS compute-in-memory SRAM macro in 7nm FinFET CMOS for machine-learning applications. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 242–244, 2020.
- [44] Jintao Zhang, Zhuo Wang, and Naveen Verma. A machine-learning classifier implemented in a standard 6T SRAM array. In *IEEE Symposium on VLSI Circuits*, pages 1–2, 2016.
- [45] Sujan Kumar Gonugondla, Mingu Kang, and Naresh Shanbhag. A 42pJ/decision 3.12TOPS/W robust in-memory machine learning classifier with on-chip train-

- ing. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 490–492, Feb 2018.
- [46] Avishek Biswas and Anantha P. Chandrakasan. Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 488–490, Feb 2018.
- [47] Sujan K. Gonugondla, Mingu Kang, and Naresh R. Shanbhag. A variation-tolerant in-memory machine learning classifier via on-chip training. *IEEE Journal of Solid-State Circuits*, 53(11):3163–3173, Nov 2018.
- [48] Tien-Ju Yang and Vivienne Sze. Design considerations for efficient deep neural networks on processing-in-memory accelerators. In *IEEE International Electron Devices Meeting (IEDM)*, pages 22–1, 2019.
- [49] Wei-Hao Chen, Kai-Xiang Li, Wei-Yu Lin, Kuo-Hsiang Hsu, Pin-Yi Li, Cheng-Han Yang, Cheng-Xin Xue, En-Yu Yang, Yen-Kai Chen, Yun-Sheng Chang, Tzu-Hsiang Hsu, Ya-Chin King, Chorng-Jung Lin, Ren-Shuo Liu, Chih-Cheng Hsieh, Kea-Tiong Tang, and Meng-Fan Chang. A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 494–496, 2018.
- [50] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O’Leary, Roman Genov, and Andreas Moshovos. Bit-pragmatic deep neural network computing. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 382–394, 2017.
- [51] Pi-Feng Chiu, Meng-Fan Chang, Che-Wei Wu, Ching-Hao Chuang, Shyh-Shyuan Sheu, Yu-Sheng Chen, and Ming-Jinn Tsai. Low store energy, low VDDmin, 8T2R nonvolatile latch and SRAM with vertical-stacked resistive memory (memristor) devices for low power mobile applications. *IEEE Journal of Solid-State Circuits*, 47(6):1483–1496, 2012.
- [52] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M. Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R. Stanley Williams. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. In *Proceedings of the Design Automation Conference*, 2016.
- [53] Boxun Li, Yi Shan, Miao Hu, Yu Wang, Yiran Chen, and Huazhong Yang. Memristor-based approximated computation. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 242–247, 2013.
- [54] Mirko Prezioso, Farnood Merrikh-Bayat, B. D. Hoskins, Gina C. Adam, Konstantin K. Likharev, and Dmitri B. Strukov. Training and operation of an

- integrated neuromorphic network based on metal-oxide memristors. *Nature*, 521(7550):61–64, 2015.
- [55] Yongtae Kim, Yong Zhang, and Peng Li. A reconfigurable digital neuromorphic processor with memristive synaptic crossbar for cognitive computing. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 11(4), 2015.
- [56] Zhe Chen, Bin Gao, Zheng Zhou, Peng Huang, Haitong Li, Wenjia Ma, Dongbin Zhu, Lifeng Liu, Xiaoyan Liu, Jinfeng Kang, and H.-Y. Chen. Optimized learning scheme for grayscale image recognition in a RRAM based analog neuromorphic system. In *IEEE International Electron Devices Meeting (IEDM)*, 2015.
- [57] Geoffrey W. Burr, Robert M. Shelby, Severin Sidler, Carmelo di Nolfo, Junwoo Jang, Irem Boybat, Rohit S. Shenoy, Pritish Narayanan, Kumar Virwani, Emanuele U. Giacometti, Bülent N. Kurdi, and Hyunsang Hwang. Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses), using phase-change memory as the synaptic weight element. In *IEEE International Electron Devices Meeting (IEDM)*, 2014.
- [58] Peng Yao, Huaqiang Wu, Bin Gao, Sukru Burc Eryilmaz, Xueyao Huang, Wenqiang Zhang, Qingtian Zhang, Ning Deng, Luping Shi, H.-S. Philip Wong, and He Qian. Face classification using electronic synapses. *Nature Communications*, 8:15199, 2017.
- [59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105. 2012.
- [60] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.
- [61] Sung Hyun Jo, Ting Chang, Idongesit Ebong, Bhavitavya B. Bhadviya, Pinaki Mazumder, and Wei Lu. Nanoscale memristor device as synapse in neuromorphic systems. *Nano Letters*, 10(4):1297–1301, 2010.
- [62] Pai-Yu Chen, Deepak Kadetotad, Zihan Xu, Abinash Mohanty, Binbin Lin, Jieping Ye, Sarma Vrudhula, J. Seo, Yu Cao, and Shimeng Yu. Technology-design co-optimization of resistive cross-point array for accelerating learning algorithms on chip. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 854–859, 2015.
- [63] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, Nov 1997.
- [64] Alex Graves, A. Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.



- [65] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry. *Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology*. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 273–287, 2017.
- [66] Cheng-Xin Xue, Wei-Hao Chen, Je-Syu Liu, Jia-Fang Li, Wei-Yu Lin, Wei-En Lin, Jing-Hong Wang, Wei-Chen Wei, Ting-Wei Chang, Tung-Cheng Chang, Tsung-Yuan Huang, Hui-Yao Kao, Shih-Ying Wei, Yen-Cheng Chiu, Chun-Ying Lee, Chung-Chuan Lo, Ya-Chin King, Chorng-Jung Lin, Ren-Shuo Liu, Chih-Cheng Hsieh, Kea-Tiong Tang, and Meng-Fan Chang. *A 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors*. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 388–390, 2019.
- [67] Mehdi Saberi, Reza Lotfi, Khalil Mafinezhad, and Wouter A. Serdijn. *Analysis of power consumption and linearity in capacitive digital-to-analog converters used in successive approximation ADCs*. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(8):1736–1748, Aug 2011.
- [68] Fabien Alibart, Ligang Gao, Brian D. Hoskins, and Dmitri B. Strukov. *High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm*. *Nanotechnology*, 23(7):075201, 2012.
- [69] Dmitri B. Strukov. *Endurance-write-speed tradeoffs in nonvolatile memories*. *Applied Physics A*, 122(4):302, 2016.
- [70] Shyh-Shyuan Sheu, Pei-Chia Chiang, Wen-Pin Lin, Heng-Yuan Lee, Pang-Shiu Chen, Yu-Sheng Chen, Tai-Yuan Wu, Frederick T. Chen, Keng-Li Su, Ming-Jer Kao, Kuo-Hsing Cheng, and Ming-Jinn Tsai. *A 5ns fast write multi-level non-volatile 1 K bits RRAM memory with advance write scheme*. In *IEEE Symposium on VLSI Circuits*, pages 82–83, 2009.
- [71] Giacomo Indiveri, Eike Linn, and Stefano Ambrogio. *ReRAM-based neuromorphic computing*. *Resistive Switching: From Fundamentals of Nanoionic Redox Processes to Memristive Device Applications*, pages 715–735, 2016.
- [72] Lukas Kull, Thomas Toifl, Martin Schmatz, Pier Andrea Francese, Christian Menolfi, Matthias Brändli, Marcel Kossel, Thomas Morf, Toke Meyer Andersen, and Yusuf Leblebici. *A 3.1 mW 8b 1.2 GS/s single-channel asynchronous SAR ADC with alternate comparators for enhanced speed in 32 nm digital SOI CMOS*. *IEEE Journal of Solid-State Circuits*, 48(12):3049–3058, Dec 2013.
- [73] Taehui Na, Byungkyu Song, Jung Pill Kim, S. H. Kang, and S.-O. Jung. *Offset-canceling current-sampling sense amplifier for resistive nonvolatile memory in 65 nm CMOS*. *IEEE Journal of Solid-State Circuits*, 52(2):496–504, Feb 2017.

- [74] Xiaohong Peng, Willy Sansen, Ligang Hou, Jinhui Wang, and Wuchen Wu. Impedance adapting compensation for low-power multistage amplifiers. *IEEE Journal of Solid-State Circuits*, 46(2):445–451, Feb 2011.
- [75] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [76] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [77] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.
- [78] Yaniv Taigman, Ming Yang, Marc’Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1701–1708, June 2014.
- [79] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3128–3137, 2015.
- [80] Stefano Ambrogio, Pritish Narayanan, Hsinyu Tsai, Robert M. Shelby, Irem Boybat, Carmelo Nolfo, Severin Sidler, Massimo Giordano, Martina Bordini, Nathan C. P. Farinha, Benjamin Killeen, Christina Cheng, Yassine Jaoudi, and Geoffrey W. Burr. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature*, 558(7708):60–67, 2018.
- [81] Irem Boybat, Manuel Le Gallo, S. R. Nandakumar, Timoleon Moraitis, Thomas Parnell, Tomas Tuma, Bipin Rajendran, Yusuf Leblebici, Abu Sebastian, and Evangelos Eleftheriou. Neuromorphic computing with multi-memristive synapses. *Nature Communications*, 9(1):2514, 2018.
- [82] Giacomo Indiveri, Bernabé Linares-Barranco, Robert Legenstein, George Deligeorgis, and Themistoklis Prodromakis. Integration of nanoscale memristor synapses in neuromorphic computing architectures. *Nanotechnology*, 24(38):384010, 2013.
- [83] Matthew D. Pickett, Gilberto Medeiros-Ribeiro, and R. Stanley Williams. A scalable neuristor built with Mott memristors. *Nature Materials*, 12(2):114–117, 2013.