Design of High-Efficiency Accelerators for Diverse AI Workloads

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Gopikrishnan Raveendran Nair

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Yu Cao, Advisor

March, 2025

Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Yu Cao for his invaluable guidance, unwavering support, and insightful mentorship throughout my research journey. His profound expertise, constructive feedback, and encouragement have been instrumental in shaping this work. Prof. Cao has continuously inspired me to push the boundaries of my knowledge and strive for excellence. I sincerely appreciate his patience, motivation, and the countless discussions that have significantly contributed to both my academic and personal growth. It has been a privilege to work under his mentorship and I am truly grateful for his support throughout my Ph.D. journey.

I also would like to thank my committee members Prof. Sachin S Sapatnekar, Prof. Caiwen Ding, and Prof. Yang Katie Zhao for their constructive feedback and thoughtful suggestions that have greatly enriched this work, helping refine my ideas and improve the quality of this thesis. I also want to extend my gratitude to my initial PhD committee members, Prof. Jae-Sun Seo from Cornell Tech, Prof. Chaitali Chakrabarti from Arizona State University, Prof. Frank Liu from Old Dominion University and Dr. Mahantesh Halappanavar from Pacific Northwest National Laboratories, for their guidance and support during my initial years of PhD. I also thank my collaborators Prof. Jeff Zhang, Fengyang Jiang, and Yaotin Liu from Arizona State University. I also want to thank all the staff of the ECE department, namely Ann Raush, for helping with the smooth transition from Arizona State University to the University of Minnesota.

I would like to thank Semiconductor Research Corporation (SRC) for their generous funding and support, which has been instrumental in enabling my Ph.D. research. Their commitment to advancing semiconductor technology has provided me with the resources and opportunities to explore and contribute to this field. I deeply appreciate their support, which has played a crucial role in the successful completion of this work.

I am thankful to the Intel Foundry Team Bryan.K.Casper, Sirisha Kale, Theodore Johnson, and others for their support and guidance during the entire chip tapeout process. I extend my heartfelt gratitude to all my labmates, both past and present, for their support, collaboration, and camaraderie throughout my Ph.D. journey. Their insightful discussions, technical advice, and encouragement have been invaluable in shaping my research. I truly appreciate the shared experiences and the friendships that have made this journey both productive and enjoyable.

I want to thank my wife Dr. Silpa Soman, my parents, my brother, his family, and my friends for their support throughout my PhD journey. Finally, my I want to thank my one month old baby boy whose smiles and gentle presence have given me newfound motivation and purpose, even during the most challenging moments of this journey.

Dedication

I want to dedicate this work to my family and to my baby.

Abstract

Artificial intelligence (AI) has evolved from dense Deep Neural Networks (DNNs) toward a diverse set of models, such as sparse graph convolutional neural networks (GCNs), LLMs. To efficiently process these workloads AI accelerators are inevitable. But designing AI accelerators is challenging as these new models differ in model size, processing flow, memory access patterns, and data/model sparsity. This thesis discusses the key design requirements of AI accelerators, including reconfigurability, heterogeneity, and scalability, to efficiently accelerate AI models. Based on these findings, three works are proposed.

A FPGA based dynamically reconfigurable GCN accelerator that efficiently handles the heterogeneous sparse and dense compute pattern in GCN. Different from deep neural networks (DNNs), GCNs are sparse, irregular, and unstructured, posing unique challenges to hardware acceleration with regular processing elements. To overcome these challenges, we propose an end-to-end hardware-software co-design to accelerate GCNs on resource-constrained FPGAs with the features including: (1) A custom dataflow that leverages symmetry along the diagonal of the adjacency matrix to accelerate feature aggregation for undirected graphs. (2) Unified compute cores for both aggregation and transformation phases, with full support to the symmetry-based dataflow. (3) Preprocessing of the graph in software to rearrange the edges and features to match the custom dataflow. The accelerator is implemented in Intel Stratix10 MX FPGA board with HBM2, and demonstrate 1.3×-110.5× improvement in end-to-end GCN latency operations as compared to the state-of the-art FPGA implementations, on the graph datasets of Cora, Pubmed, Citeseer and Reddit.

To address the need for a heterogeneous accelerator to support the diverse set of AI workloads a new RISCV based reconfigurable heterogeneous accelerator, with the target to balance the computation needs and energy efficiency is proposed. Based on representative DNNs and GCNs, we propose two types of processing elements (PEs): (1) A Latch-based digital IMCs (LIMC) for regular and dense computation, and (2) A digital SIMD array (SIMD) with fine-grained control for irregular and sparse workloads.

To integrate both types of PEs and dynamically manage the data flow, we design reconfigurable modules of scatter/gather and buffers, supporting different types of memory access and compute patterns. The new heterogeneous accelerator has been designed and taped out at 16nm. Based on 16nm design data, it achieves an 11× improvement in latency compared to baseline homogeneous accelerators, and up to 2.1× and 20× improvement in TOPS/mm^2 and TOPS/W, respectively, as compared to state-of-the-art accelerators.

With AI models evolving at a rapid pace and along with a huge amount of data volume, it is critical that AI accelerators also be easily scalable to meet the performance and data bandwidth requirements. Traditional computer vision tasks in autonomous machines and AR/VR rely on high-speed links, such as MIPI CSI-2, to transfer data from sensors to computing units.While these systems have struggled in the past to meet the growing demands on high bandwidth and low latency, today's advanced packaging technologies that allow for multiple tiers of sensing and computing chiplets to be stacked together have the potential to support real-time processing of these tasks. In this work, we utilize advanced packaging technology, specifically 3D die stacking with high-density copper (Cu) pillars, to develop a 2-tier hardware-software co-design for an AI ViT accelerator. Ultimately, this 2-tier accelerator will be integrated with the sensing tier to process continuous data streams. Our 3D stacking approach, featuring face-to-face bonding with a 5µm pitch, offers two key advantages: (1) higher compute density than what is offered by 2D / 2.5D packaging and (2) higher connection density than conventional TSV-based stacking. Our synthesis results at 28nm demonstrate an 18x improvement in latency and a 127x reduction in energy consumption compared to conventional 2D designs and an 11x improvement in latency compared to similar 3D architecture.

Contents

Acknowledgements **i**

Dedication **iii**

Abstract **iv**

List of Tables **ix**

List of Figures **x**

1 Introduction **1**

1.1 Design Requirements of AI accelerators 3

1.2 Thesis organization 4

2 FPGA Acceleration of GCN in Light of the Symmetry of Graph Adjacency Matrix **7**

2.1 Introduction 7

2.2 Background and Motivation 10

2.2.1 GCN Operations 10

2.2.2 Hardware Acceleration of GCNs 10

2.3 FPGA Architecture and Dataflow 12

2.3.1 Architecture Overview 12

2.3.2 Symmetry of Adjacency Matrix 14

2.3.3 Custom Data Flow based on Tiles 16

2.3.4 Mapping Dataflow to Hardware 17

2.4 Results . . . 19
2.4.1 Experimental Setup . . . 19
2.4.2 Quantization of GCN Algorithms . . . 20
2.4.3 Speedup with the Symmetry Property . . . 21
2.4.4 Comparison with State-of-the-Art Results . . . 21
2.5 Conclusion . . . 24

3 A 16nm Heterogeneous Accelerator for Energy-Efficient Sparse and Dense AI Computing 25
3.1 Introduction . . . 25
3.2 Background and Motivation . . . 28
3.3 Architecture . . . 31
3.3.1 Architecture Overview . . . 31
3.3.2 Latch-based IMC (LIMC) Core . . . 32
3.3.3 Mapping of Workloads to LIMC . . . 40
3.3.4 Mapping of Aggregation to SIMD . . . 41
3.4 Results and Discussion . . . 42
3.4.1 Workload Mapping: LIMC vs. SIMD . . . 44
3.4.2 Improvement of LIMC from Input Skipping and Booth Multiplier 45
3.4.3 Comparison with LIMC-only Accelerators . . . 45
3.4.4 Energy Inefficiency of LIMC with Sparsity . . . 46
3.4.5 Comparison with Prior Works . . . 46
3.5 Conclusion . . . 48

4 HW-SW Co-Design of A 28nm 2-Tier ViT Enabled by Advanced 3D packaging 49
4.1 Introduction . . . 49
4.2 Background and Motivation . . . 52
4.3 2-layer ViT Algorithm Design . . . 53
4.4 3D Architecture . . . 55
4.4.1 Architecture Overview . . . 55
4.4.2 System Co-optimization . . . 57
4.4.3 Algorithm Partition and Mapping . . . 58

4.4.4 Physical Design Considerations 60
4.5 Results 63
4.6 Conclusion 66

5 Conclusion and Discussion **68**

References **70**

List of Tables

2.1 Characteristics for four different GCN datasets. 20
2.2 Resource utilization of FPGA for different datasets. 22
2.3 Comprehensive evaluation of execution time for different datasets across different implementations. 22
2.4 Hardware resource used by various designs. 23
3.1 GCN and SpMV workloads are dramatically sparse. 26
3.2 Workload characteristics of CNNs, GCNs and SpMV. 30
3.3 Improvement in compute latency achieved from the proposed ping-pong IMC architecture. 36
3.4 Instructions and the hardware used. 39
4.1 2 layer ViT Model optimization results 55
4.2 Comparison of 2D vs 3D stacked accelerator system. 64
4.3 Comparison with prior work. 66

List of Figures

1.1 Shows how the thesis is organized. The thesis starts with an introduction to a set of AI workloads. Chapter 2, 3 explains the design criteria for AI accelerators based on these workloads characteristics and Chapter 4 describes how to build an scalable accelerator based on these design principles 5

2.1 Data processing in GCN: (a) Input graph, (b) Transformation of two nodes, and (c) Aggregation of a node. 9

2.2 The dynamically reconfigurable FPGA accelerator for GCNs. 13

2.3 Aggregation with tiles: (a) Adjacency matrix with compute tiles highlighted; (b) Symmetry on a pair of transpose tiles with the direction of aggregation; (c) Vertical and horizontal aggregation on a tile. 15

2.4 Custom dataflow based on symmetric tiles: (a) Adjacency matrix with tiles numbered according to the order of execution; (b) Dataflow with input and output at each iteration step. 16

2.5 Dataflow on FPGA: (a) Mapping of transformation into the PE array; (b) The adjacency matrix tile showing the interval partition; (c) Edge buffer for core 1 and mapping of edges from the buffer to the core; (d) Layout of node features inside a banked on-chip memory. 17

2.6 Quantization of the GCN models. 8-bit is selected before experiencing any accuracy loss. 21

2.7 The symmetry property effectively helps reduce the number of compute tiles in GCN acceleration. 23

3.1 Mapping of sparse workloads on IMCs is inefficient, while our proposed SIMD cores effectively improve computational efficiency and data access. 27

3.2 Comparison of Heterogeneous AI models: (a) Conv MAC operations in CNN with high data reuse predictable memory access patterns (b)GCN model showing feature transformation(MAC) and aggregation(non-MAC) with random memory access and sparse compute. 28
3.3 Architecture of the heterogeneous accelerator with ping-pong LIMC and SIMD cores. 31
3.4 Microarchitecture of the proposed LIMC bank with the booth multiplier. Operations are skipped at the coarse level if the weights/activations are zero and at the fine level if the booth encoding bits are 000/111. 33
3.5 A smaller number of LIMC banks have smaller area due to less number of multipliers, but will also have more iterations (the size of the input/the number of banks) at the input of LIMC increasing latency. 34
3.6 (a) Conventional IMC core where computation and weight updates are sequential. (b) Proposed Ping-Pong based LIMC core to efficiently hide the IMC weight update latency by overlapping the computation and IMC weight updates across the ping-pong cores. 35
3.7 Proposed ping-pong IMC hiding data movement latency by overlapping weight load and computation. 36
3.8 Micro-architecture of SIMD core. 37
3.9 Micro-architecture of SIMD core. 38
3.10 Example of a LD/ST instruction. 39
3.11 Example of mapping convolution into the LIMC core. The IFMs are mapped across the rows within a LIMC macro and the OFMs are mapped across the macros. 40
3.12 Mapping of a tile adjacency input matrix to the compute units for aggregation. Sparse buff stores the edges in COO format. The control fetches and assigns each edge to a PE row. Banked Intermediate buffer stores the features of the node to be accessed parallelly by the SIMD cores. . . 41
3.13 Overview of the system level SOC architecture 43
3.14 Post layout of the proposed accelerator. 43

3.15 At lower level of sparsity LIMC achieves better energy efficiency whereas at higher sparsity SIMD achieves better energy efficiency for identical workload. 44
3.16 Reduction in the number of computations by exploiting sparsity at the input banks of LIMC by using operation skipping and booth multiplier. 45
3.17 End-to-end GCN and CNN inference performance comparison between the LIMC-only homogeneous accelerator and heterogeneous architecture. 46
3.18 LIMC vs SIMD energy consumption for GCN FA and SpMV. Proposed SIMD achieves high energy efficiency for sparse GCN FA and SpMV workload. 47
4.1 (a) Existing In Sensor Compute solution with a sensor layer stacked on top of compute die using TSVs (b) Proposed 3D stacked solution using face-to-face bonding with two logic die to support 2-layer ViT for small object detection. 50
4.2 MIPI-CSI2 cannot support the high bandwidth requirement of sensors. Proposed 3D stacked design with face-to-face bonding with high density interconnects provides sufficient bandwdith to support the growing sensor resolutions. 51
4.3 Proposed 2 layer ViT for small object detection. 53
4.4 (a) Top Die architecture with the F2F connections highlighted using dots (b) Bottom die architecture and the corresponding F2F bonds from top die highlighted using same color (c) 3D view of the architecture with the density hybrid bond shown in green(d) Dataflow of the proposed architecture highlighting localized parallel computation for one input channel. We partition the input into 4 tiles, ensuring localized intra-tile data transfer with no inter-tile communication to achieve maximum parallel computation and exploit the high bandwidth provided by the 3D system. 56
4.5 Shows three different algorithm partition and mapping strategies. The die layers are represented by different colors. 58

4.6 Top figure shows the evaluation metrics for different algorithm partition and hardware mapping strategy. Bottom figure shows the volume of generated at the output of each stage of algorithm. Our mapping strategy ALG_MAP2 exploits the high connection density of 3D packaging to minimize high volume data movement latency across dies, thus minimizing the end to end latency with moderately balanced area and power density ratio of top to bottom die. 59
4.7 IO circuitry for spice simulation of regular IO pads and for F2F CU pillar hybrid bonding pad. Components in light gray are left out for F2F bonding. 60
4.8 Resulting Gate Voltage from simulated ESD event. An ESD voltage of 250V was used for the regular pad, and 5V for the F2F bonding case. The minimum required diode size was chosen for these examples. Gate-oxide breakdown voltage for this process is approximately 3.8V, scaled from the values in [1] 61
4.9 Illustrative power density distribution of the top and bottom stack combined. At the edges the power density is higher since there are two compute cores PEMbed and SWIN1 stacked on top, and the power density is lower in the middle since there are only SWIN2 cores. The power density is within the acceptable range of 1W/mm^2 for 3D designs to avoid thermal issues [2, 3]. 62
4.10 Spice simulation waveform of sending a clock signal across the die stack through Cu pillar hybrid bond. The propagation delay is very negligible, in the range of picoseconds thus resulting in negligible clock signal variations. 63
4.11 Energy per pixel (transmission + compute) for 2D and 3D accelerators at different FPS. The 2D accelerator uses MIPI-CSI2 to transfer the entire image frame, resulting in high transmission energy. The 3D stack with TSVs and F2F bond has lower transmission energy. The lower compute density of the 2D accelerator increases compute latency per frame and hence, compute energy compared to the 3D accelerator. 64
4.12 Power density of the proposed 3D stacked accelerator vs 2D accelerator of same area at different frequencies. The power density is within the acceptable range of 1W/mm^2 for 3D designs to avoid thermal issues. . . 65

Chapter 1

Introduction

Over the last decade deep learning has evolved from a research field to an omnipresent technology in many application domains such as image classification, graph analysis, autonomous driving, and natural language processing. The primary factors behind the exponential growth in deep learning can be attributed to the three factors: algorithmic development, availability of a large amount of training data, and the sheer increase in computing power.

A wide variety of deep learning models have been proposed over the years. Early deep learning models started with Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs). CNNs use a series of convolution layers that extract features from the inputs. Each layer extracts a different set of features and with a deep model, complex tasks beyond human capabilities can be solved. But CNNs are not good at analyzing sequential data like text, where the order of the input vector also matters.

Thus, Reccurent neural networks(RNNs) were introduced to extract information out of time-dependent and sequential information. For RNNs, each layer receives the input from not only the current timestep but also the previous hidden state. Each layer computes a weighted sum of the present input and the previous hidden state, and a non-linear function is applied to generate the current hidden state. Thus, unlike CNNs, each layer of the RNN output is a function of current and previous state. Both CNNs and RNNs are good at extracting latent information from grid-like structures like images or vectors, but a large number of applications use nongrid-like data representations like graphs. To handle the irregularity of graphs, Graph Neural Networks(GNNs) have been

introduced. In general, GCNs consist of two main operations: aggregation and transformation. GCNs take a high-dimensional representation of graph data and transform it into a lower-dimensional representation, without altering the graph structure. In feature aggregation, each node takes the feature vector associated with its neighboring nodes and aggregates them to produce a new feature vector. GNNs are commonly used for personal recommendation systems, social media platforms, etc.

In recent years, another type of deep learning model, called transformers and large language models (LLMs), has changed the field of natural language processing and image processing. Unlike prior models, the core of these models is self-attention, which calculates the relation between different parts of the input sequence, thus achieving much more contextual information. Thus, LLMs are capable of handling complex tasks like text generation, language translation, chatbots, image generation, etc.

Over the years, all of these models have grown in size and complexity. Since AI models are sophisticated pattern detectors, the more training data, the higher its accuracy will be. Therefore, the datasets used for training have been growing 3x every year from a little over 1 million to about 9 trillion datapoints over the last decade [4]. The number of model parameters follows a similar trend. Model parameters have been doubling every year since 2010 from little less than 1 million to around 175 billion parameters for latest GPT models [4].

As AI models scale up, they require more computational hardware resources. From the hardware standpoint the critical aspects of AI accelerators involve computation, data storage and transport. The compute capabilities have been consistently increasing over the years, but the memory bandwidth has failed to keep up with it, thus causing a memory bottleneck. Van Nuemann architecture-based high performance multicore CPUs can no longer support the computation requirements of the ever growing AI models due to it's low compute parallelism and the memory bottleneck. Therefore, AI accelerators with specialized parallel processing capabilities are critical to efficiently accelerate AI models. Different platforms like GPUs, FPGAs, and ASICs are used to develop accelerators for AI models. GPUs are extremely efficient at executing large number of parallel execution threads, but due to its high power requirements, they are mainly used for large-scale intensive tasks like training or datacenter-based AI applications. FPGAs based AI accelerators can leverage the reconfigurability of FPGAs to dynamically modify the hardware

based on the algorithm requirements. FPGAs also provide energy efficiency, but at the cost of processing speed compared to GPUs. ASICs have a longer design cycle and much less flexibility, but provide optimum computational speed and power consumption.

1.1 Design Requirements of AI accelerators

Since the earlier CNNs models are mostly comprised of dense MAC operations, AI accelerator design space so far has been mostly focusing on having efficient MAC cores. As AI models have evolved over the last years, this simplistic approach of optimizing hardware architectures for MACs is no longer efficient. It is critical to analyze how AI models have evolved to understand the design requirements of AI accelerators. The computational patterns of the AI models can be heterogeneous. Heterogeneity exists within an AI model, such as in GCNs. GCNs have alternating dense and sparse computation patterns. Hence, an AI accelerator designed for dense computation is inefficient for the sparse computation phase. Therefore, hardware should be dynamically reconfigurable depending on the computation phase.

Heterogeneity exists across AI models as well. When deploying an AI accelerator as a co-processor in a system, it should efficiently support the heterogeneous nature of computations of the different AI models. Different AI workloads have different computation patterns; it can be dense convolutions, vector matrix multiplications, or sparse matrix computations. Therefore, AI accelerators should have heterogeneous compute cores with dynamic workload mapping to efficiently handle the heterogeneity across AI models.

Finally, the costs of monolithic hardware designs are increasing. The die sizes have reached the reticle limit and the cost per yielded die mm^2 is also increasing. With Moore's law slowing down, the guaranteed increase in transistor count and thereby the compute performance are also slowing down. The transformer-based models' extensive memory and data bandwidth requirements put strain on existing memory architectures and on chip interconnects. Moreover, with the ever-evolving AI landscape, it is important to have an easily scalable hardware architecture to reduce design times and keep pace with algorithm development. Thus, it is important to think beyond the monolithic design space for the design of next-generation scalable AI accelerators enabled by advanced

packaging methodologies, including 2.5D and 3D integration.

This thesis discusses the key design requirements of the AI accelerators mentioned above, including reconfigurability, heterogeneity, and scalability, to efficiently accelerate AI models. Based on these findings, the thesis proposes the following.

- A Dynamically reconfigurable FPGA-based accelerator for GCNs which can efficiently handle the heterogeneity within the AI model
- A heterogeneous RISC-V-based AI accelerator with dense and sparse compute core to support the heterogeneous compute nature that exists across AI workloads.
- A heterogeneous AI accelerator architecture featuring 3D-stacked sensor-compute layers (using TSV and F2F bonding with Cu pillars), achieving a fully scalable solution for efficient edge compute acceleration of Vision Transformers.

1.2 Thesis organization

Figure 1.1 shows how the thesis is organized. In this thesis, we analyze a set of AI workloads, and based on their characteristics, we divide the accelerator design criteria into Chapters 2 and 3, each targeting different kinds of heterogeneity across AI workloads. Finally, in Chapter 4 discusses how a scalable accelerator system can be built up using advanced packaging using the design criteria, namely hybrid dataflows and heterogeneous compute cores.

Chapter 2: FPGA Acceleration of GCN in Light of the Symmetry of Graph Adjacency Matrix

The chapter provides the fundamental background of GCN and discusses why existing architectures are not efficient for GCN acceleration due it's heterogeneous compute patterns. Then the chapter discusses about the proposed solution, the hardware architecture and it's dataflow, and finally the results.

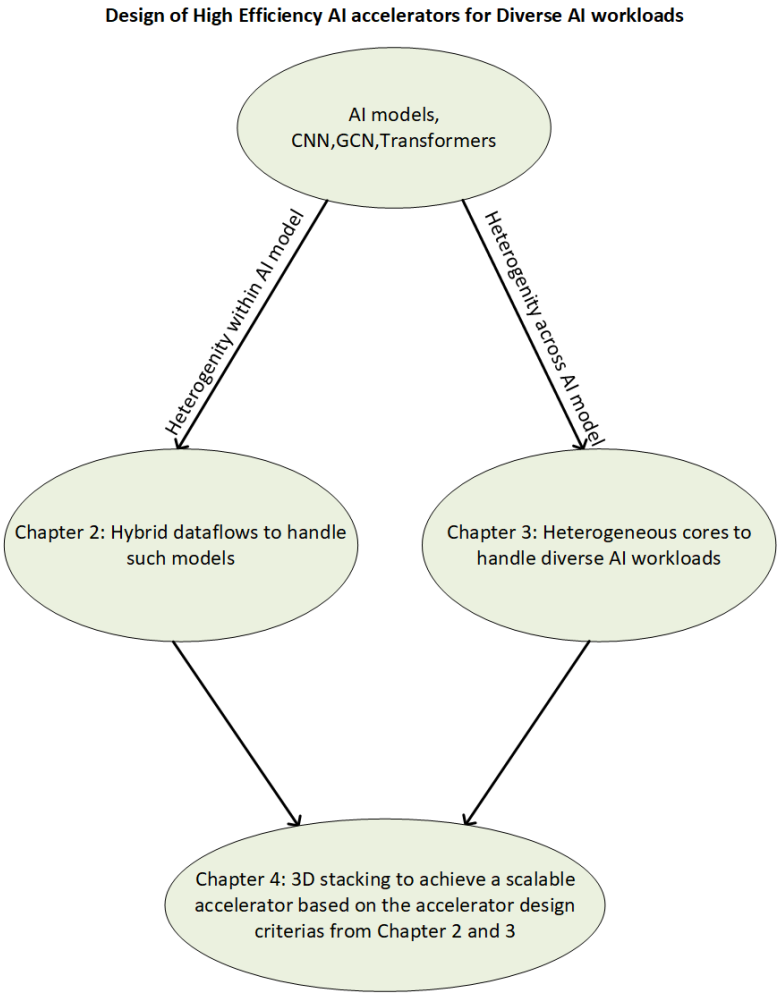

Figure 1.1: Shows how the thesis is organized. The thesis starts with an introduction to a set of AI workloads. Chapter 2, 3 explains the design criteria for AI accelerators based on these workloads characteristics and Chapter 4 describes how to build an scalable accelerator based on these design principles

Chapter 3: 16nm Heterogeneous Accelerator for Energy-Efficient Sparse and Dense AI Computing

This chapter introduces the existing AI accelerator architectures and how it is inefficient in handling the compute heterogeneity that exists in today's AI model. Later the chapter discusses about the proposed solution, the hardware architecture, and finally the discussion of results from the taped-out design.

Chapter 4: HW-SW Co-Design of a 28-nm 2-Tier ViT Enabled by Advanced 3D Packaging

The chapter gives an introduction to advanced packaging and why it is important in the current design of the AI accelerator architecture. Then the chapter discusses Vision Transformers and the proposed scalable 3D stacked accelerator architecture, it's dataflow and finishes with a discussion on results.

Chapter 5

Presents a final discussion of the analysis presented in the thesis.

Chapter 2

FPGA Acceleration of GCN in Light of the Symmetry of Graph Adjacency Matrix

2.1 Introduction

Graph Convolutional Neural Networks (GCNs) are the state-of-the-art machine learning method to process the graphs. GCNs take a high dimensional representation of graph data and transform it into lower dimension representation, without altering the graph structure [5,6]. The generated embedding can be used for tasks such as link prediction [7–9], recommendation systems [10], and fraud detection [11]. The need for graph data processing and the learning capabilities of GCNs have made it one of the main research topics in graph-based learning. This in turn led to the development of software-based graph learning libraries [12, 13] and hardware accelerators for GCNs [14–16]. In general, GCNs consist of two main operations: aggregation and transformation, as shown in Figure 2.1. In feature aggregation, each node takes the feature vector associated with its neighboring nodes and aggregates them to produce a new feature vector. This can be represented as a product between the adjacency matrix (A) and the feature matrix (X) of the graph. Aggregation operation depends on the graph structure. The feature transformation phase transforms the feature matrix of the nodes to a new dimension,

using a weight matrix. This operation is similar to that in a multi-layer perceptron (MLP). A non-linear function, such as ReLU, is then applied to the output Z. Finally, these operations are repeated for all layers within the GCN. A typical GCN uses 2–3 layers [5, 6].

Acceleration of GCNs on hardware platforms is challenging due to its sparsity, randomness, and non-uniformity. The adjacency matrix of a GCN is a sparse data structure leading to irregular memory accesses, low spatial/temporal data locality and reuse [6]. Power-law like unbalanced degree distributions in real-world graphs create significantly different processing times for each node, resulting in workload imbalance. Consequently, the execution time and memory access patterns of each node in aggregation are unpredictable and irregular. On the other hand, feature transformation is a dense matrix multiplication with regular memory access and high data reuse.

Therefore, GCN accelerators should be capable of handling both the sparsity and irregularity of aggregation, and the dense and regular computation in feature transformation. The irregularity makes GCNs not suitable for current CPUs, whose architecture is based on multilevel caches and data prefetching [14], are inefficient to manage irregularity and low data reuse between compute units. On the other hand, GPUs are inherently optimized for compute-intensive workloads with regular execution patterns [17] and thus, are suitable for feature transformation but not aggregation. Other accelerators that are optimized for graph traversal require large volume of external memory accesses, and thus unsuitable for GCN acceleration [15,18]. Existing papers [14–16] have deployed dedicated engines for feature transformation and aggregation but the heterogeneous computing pattern of GCNs will lead to workload imbalance between the two phases of GCN computations making pipeline balancing a difficult task.

Different from these approaches, we used a dynamically configurable unified compute core to do aggregation or transformation. As feature aggregation is unique to GCNs and presents a major challenge, we propose a custom execution dataflow to efficiently accelerate feature aggregation in this work. By exploiting the symmetry of adjacency matrices in undirected graphs, we utilize either the upper or the lower triangular matrix to perform aggregation in GCN. Furthermore, we develop a dynamically reconfigurable compute core to unify the processing for aggregation and transformation. This approach minimizes the balancing issues in workload and pipelining in GCN operations.

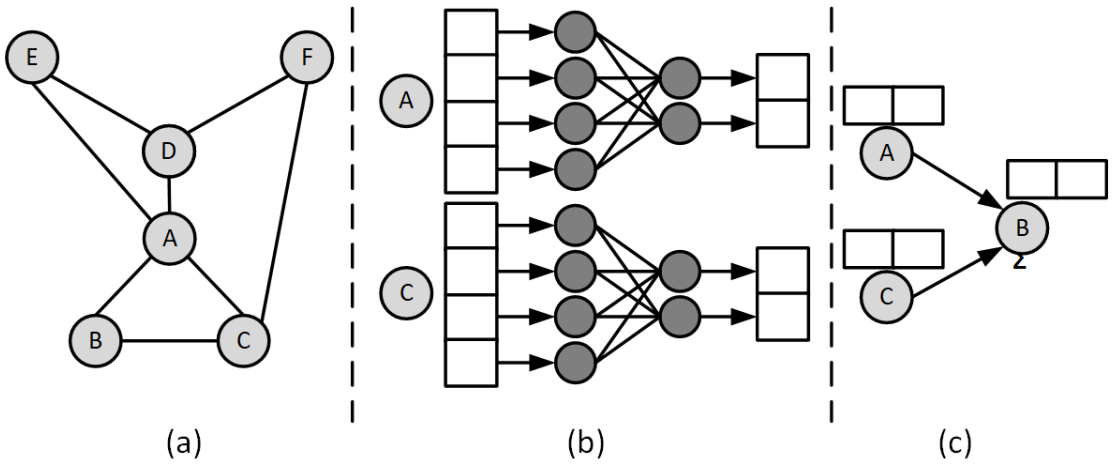

Figure 2.1: Data processing in GCN: (a) Input graph, (b) Transformation of two nodes, and (c) Aggregation of a node.

The major contributions of this work are as follows:

- We propose a dataflow to accelerate feature aggregation by exploiting the symmetry of the adjacency matrix. Such a dataflow reduces redundant data movement and improves data reuse.
- We design a dynamically reconfigurable compute core and associated control logic to support the heterogeneous workload across aggregation and transformation, as well as the symmetry nature of feature aggregation.
- We design a preprocessing method to rearrange the edges and features of the graph in order to match the data flow. This step ensures more regular memory access and better data reuse in the aggregation phase.
- We implement our design on Intel Stratix 10 MX FPGA board with HBM2, and achieve significant improvement in end-to-end latency: 1,101× and 202× over PyG [12] and DGL [13] on CPUs, 13× and 17× over PyG and DGL on GPUs, and 1.3×-110.5× over prior FPGA accelerators [15, 16] on four graph datasets.

2.2 Background and Motivation

2.2.1 GCN Operations

GCNs have a neighborhood aggregation scheme where the features of neighbouring nodes are recursively transformed and aggregated to generate a new representation for the target nodes, which is commonly deployed in GCN [5], GraphSage [19], and GAT [20]. In this work, we focus on the inference of one the most popular model, GCN in [5].

Primary operations in the GCN include transformation and aggregation, as shown in Fig. 2.1. The output of a layer ($\mathbf{X}^l$) can be represented as: $\mathrm{X}^l = \mathrm{ReLU}(\tilde{A}\mathrm{X}^{(l-1)}W^l)$. $\tilde{A}$ is the normalized adjacency matrix, given by $\tilde{A} = \mathrm{D}^{-1/2}\mathrm{A}\mathrm{D}^{-1/2}$, where D is the degree matrix of the graph. Here the transformation operation can be represented by $\mathrm{X}^{l'} = \mathrm{X}^{(l-1)}W^l$, and the aggregation operation is represented by $\mathrm{X}^l = \tilde{A}\mathrm{X}^{(l')}$.

In the transformation phase, the features ($\mathbf{X}$) of the graph nodes are multiplied with a weight matrix ($\mathbf{W}$) to transform the features. During aggregation, each node's feature is aggregated with the feature of its neighboring nodes to generate a new representation. The output is then passed through an activation function to produce the output of a layer in GCN. These operations are repeated for all layers within a GCN. Transformation is similar to a multi layer perceptron(MLP) and is well known as a dense matrix multiplication. Therefore is it more compute intensive.Where as aggregation is dependent on graph structure which is random and sparse. This leads to random memory accesses and poor data reuse.

2.2.2 Hardware Acceleration of GCNs

Recent works demonstrate effective hardware acceleration of GCNs over CPUs and GPUs [14–16]. A GCN accelerator should be able to manage both the dense transformation phase and the sparse aggregation phase. To that end, design challenges remain open on the following aspects:

Order of Computation

GCN's computations follow associative law i.e. A.(X.W) is same as (A.X).W. Therefore, an analysis for the execution time for a single layer of GCN on all datasets using the graph

processing library PyG [12] on a CPU. The transformation followed by the accumulation is denoted as *FaFt* and the accumulations followed by the transformation as *FtFa*. *FtFa* results in a lower latency than *FaFt*. This is because Ft operation transforms the feature matrix into lower dimension and thus resulting in lower number of computations for aggregation. The accelerator will be using *FtFa* as the computation order in the design. The design is also capable of supporting the *FaFt* if required.

Imbalanced workload

GCNs have a heterogeneous workload. Feature transformation has dense computations with regular memory accesses and predictable execution times, whereas aggregation is a sparse computation with random memory accesses and irregular execution times. This leads to a difficult task of balancing the pipeline between the two different computations.

Existing work uses separated compute engines for transformation or aggregation [14–16]. Since the latency of these two stages is imbalanced, such a design usually encounters pipeline stalls [15]. In [15], when the pipeline stalls occur, the generated output of a compute engine has to be written back to external memory, at the cost of more external memory accesses. To overcome this, we unify the compute core, which can be dynamically configured into an aggregation or transformation engine based on the computation flow. The dataflow is designed such that the output produced from one operation will be consumed by the subsequent operation. EnGN [21] uses a unified core based on a Ring-Edge-Reduce dataflow. However, this can lead to idle PE cycles as each PE has to wait for the data from the other PEs to propagate. We overcome this by writing the output to a chip buffer and making it available to all other PEs to access instantly.

Data reuse under the symmetry

For transformation, the same weights are reused by all the vertices in a layer. This ensures a high level of data reuse by default. The randomness of the aggregation makes it difficult to ensure data reuse. Distinguished from [14, 15], our hardware architecture takes advantage of symmetry in the adjacency matrix to maximize data reuse. Given an undirected graph, the upper half and the lower half of its adjacency matrix contain the same information, and thus, in principle, we only need one half of the adjacency matrix

to complete the aggregation phase. We further address this in data pre-processing on software to match the custom dataflow.

GCN quantization

Previous works suffer from high memory footprint due to large graph size and 32-bit floating point data representation [14–16]. GCN has very few layers compared to DNNs but has high computational complexity. Previous works [14–16, 22] use 32-bit floating-point to represent data. This leads to higher memory footprint for bigger graphs, higher latency for data movement and more computational energy. To that end, we adopt quantization in GCN training to reduce data representation from 32-bit floating point to 8-bit fixed point. This solution helps reduce memory requirement on FPGA, without sacrificing the inference accuracy.

2.3 FPGA Architecture and Dataflow

In this section we'll discuss about our unified core architecture and the symmetry based dataflow.

2.3.1 Architecture Overview

Fig. 4.4 shows the architecture of the GCN accelerator. It consists of six compute cores, a task scheduler, a reduction engine, the DMA controller, off-chip HBM2 memory, and on-chip buffers.

Each core in the accelerator supports the unified operation for aggregation and transformation. With a local control logic, it can work independently on any task. The entire compute units can also be assigned for either aggregation or transformation. It consists of a scheduler for aggregation or transformation, an array of processing elements (PEs), and a local buffer. The aggregation and transformation scheduler generates corresponding control signals to the core. The local buffer stores the partial results of computation and feature data of nodes, which are reused by the PE array; it is exclusive for each core. The PE array is organized into R rows and C columns. They can work collectively as a systolic unit for transformation, or as individual rows of PEs for aggregation. Each

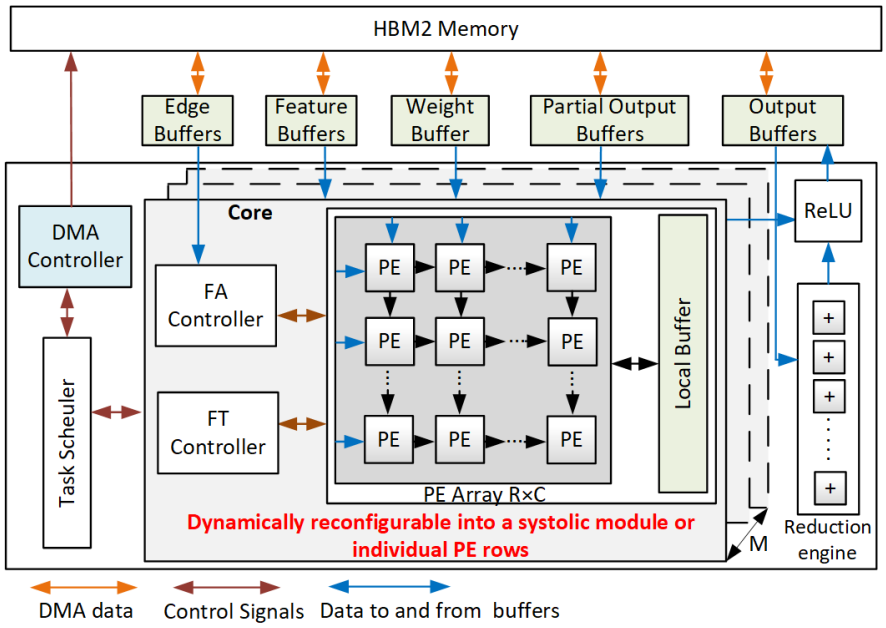

Figure 2.2: The dynamically reconfigurable FPGA accelerator for GCNs.

PE is an 8-bit fixed point multiply-and-accumulate unit. For all cores inside the compute engine, the task scheduler generates control signals to enable either the aggregation scheduler or transformation scheduler in each core; the reduction engine accumulates the output produced across multiple cores. It is a vector accumulator unit.

We use multiple on-chip buffers for various purposes. The buffers are banked in such a way it will allow the cores to access the buffers to access in parallel with out conflicts. The edge buffer contains the edge information of the graph in compressed co-ordinate format (COO). It is also used as an instruction buffer for the aggregation scheduler to execute aggregation of a tile. It also have other control information such as which location of on chip memory to read from, start and end of aggregation, configuration instructions like number of iterations etc. Each core has its own edge buffer allowing parallel edge access. The feature buffer stores the node features for aggregation and transformation. The weight buffer saves the weights for feature transformation. It is a banked memory with the number of banks equal to C in the PE array. The partial output buffer stores the partial output generated in the previous iteration to be consumed by the current iteration. The output buffer holds the output generated from the core for

reduction engine. It is also a banked buffer with the number of banks equal to the number of compute cores (M). The data flow is designed to ensure that no inter-core communication is needed.

2.3.2 Symmetry of Adjacency Matrix

For an undirected graph, the upper and lower half of its adjacency matrix contain the same information, i.e., it is symmetric. Therefore, in principle, we only need one half of the adjacency matrix to complete the aggregation phase of GCN. Fig. 2.3(a) presents an example of the adjacency matrix of an undirected graph with six nodes. For a large graph, we need to partition the matrix into smaller tiles to fit into on-chip memory. As shown in Fig. 2.3(a), the six nodes are grouped into three vertex groups, V1, V2, and V3. Each interval has got two nodes in it. The adjacency matrix is organized into 2×2 tiles, as (V1, V1), (V1, V2), etc.

In an undirected graph, the i^{th} row and i^{th} column of the adjacency matrix is the transpose of each other. This property is true for the tiles as well, e.g., tile (Vi, Vj) and tile (Vj, Vi) are the transpose of each other and contain the same information. Therefore, we only need to compute one of them and can skip the other, saving approximately half of the adjacency matrix in computing as highlighted in Fig. 2.3(a). Extending this to an entire graph, we only need to compute approximately one half of the adjacency matrix as highlighted in Fig. 2.3(a).

The aggregation operation is then performed tile by tile. Since the entries of the adjacency matrix are ones and zeros we can add the features of the non zero entries in a row and skip the multiplication with feature matrix. For the tile (V1, V2), the features of the neighbors or source vertex group V2 are added to the destination vertex group V1 along the horizontal direction, as shown in Fig. 2.3(b). This produces a partial output (PO) denoted as POV1. Similarly, for the transpose pair (V2, V1), a partial output POV2 is produced. Due to the symmetry, we only need to compute one tile from a transpose pair. To compensate the horizontal aggregation PO from the discarded tile, we conduct an aggregation in the vertical direction in the selected transpose pair tile as shown in Fig. 2.3(c), i.e., the features of V2 is getting added to V1. For vertical aggregation, the source and destination vertices are interchanged as compared to the horizontal aggregation. Thus, except for the diagonal tiles, each tile produces two partial

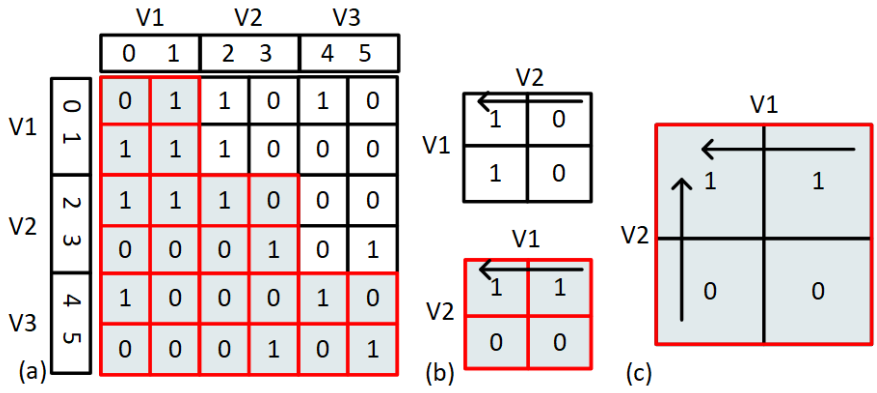

Figure 2.3: Aggregation with tiles: (a) Adjacency matrix with compute tiles highlighted; (b) Symmetry on a pair of transpose tiles with the direction of aggregation; (c) Vertical and horizontal aggregation on a tile.

outputs, one from the vertical aggregation and the other from the horizontal aggregation. For the diagonal tiles there are no transpose pairs as the transpose of a diagonal tile is the same. Therefore, there is only one output, either from the horizontal or vertical aggregation.

Since the adjacency matrix is extremely sparse, it is stored in the coordinate format (COO). This helps us skip zero entries inside the adjacency matrix. In the COO format, each edge is represented as (*row*, *col*, *value*). Storing the adjacency matrix in the COO format further helps with the vertical and horizontal aggregation operation of tiles. For horizontal aggregation, the *row* entry becomes the destination node and *col* entries as its neighbors, where for vertical aggregation the *col* entry becomes the destination and *row* entries as its neighbors. Thus, without any storage overhead, we can compute vertical and horizontal aggregation using the COO format.

In summary, by using the symmetry property for aggregation, we produce twice the number of outputs for non-diagonal tiles, as compared to the normal aggregation, ensuring more edge data reuse for the same set of edge inputs.

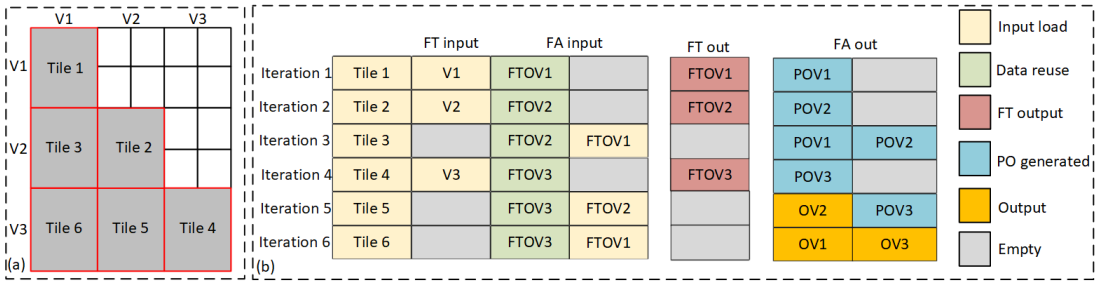

Figure 2.4: Custom dataflow based on symmetric tiles: (a) Adjacency matrix with tiles numbered according to the order of execution; (b) Dataflow with input and output at each iteration step.

2.3.3 Custom Data Flow based on Tiles

Fig. 2.4 presents the dataflow of GCN acceleration based on the symmetry property. The required inputs and generated outputs at each step of iteration are shown in Figure 2.4(b). *FT input* is the input required for transformation and *FA input* for aggregation. Tile i represents the adjacency matrix of tile in the COO format and Vi represents the features of the corresponding vertex group.

For Iteration 1 in Fig. 2.4(b), the inputs are Tile 1 and features of V1. It produces a transformation output FTOV1. This output is then reused as the input for aggregation and produces one partial output POV1. Since it is a diagonal tile, only horizontal aggregation output is produced.

For Iteration 3 which is a non-diagonal tile, we have both vertical aggregation and horizontal aggregation. There is no transformation operation for Tile 3 as the feature inputs required for horizontal aggregation are the output of transformation of Tile 1 and feature the input for vertical aggregation is the transformation output of Tile 2. Thus, for a non-diagonal tile, we have only aggregation but no transformation. The aggregation of Tile 3 produces two sets of output POV2 and POV1. Here POV1 and POV2 is accumulated on top of the POV1 and POV2 from the previous iteration.

In Iteration 5, the aggregated output for vertex group V2 (OV2) is produced, and by the end of Iteration 6, the complete aggregated output for vertex group V1(OV1) and V3(OV3) is produced. As shown in Fig. 2.4(b), the order of computation is selected to maximize input reuse. By computing the diagonal tiles first and then the non-diagonal

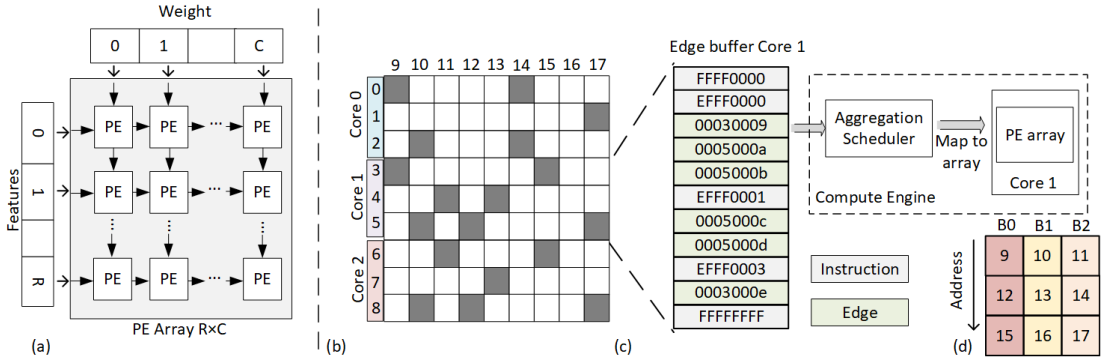

Figure 2.5: Dataflow on FPGA: (a) Mapping of transformation into the PE array; (b) The adjacency matrix tile showing the interval partition; (c) Edge buffer for core 1 and mapping of edges from the buffer to the core; (d) Layout of node features inside a banked on-chip memory.

tiles in the same row, we ensure maximum data reuse of the output from the diagonal tiles. The process continues until all tiles are computed. The dataflow holds true for adjacency matrix of any size. Using the symmetry property we are reusing the same edges within the non diagonal tiles to produce two outputs. Since we are exploiting the symmetry property, our method requires up to 50% less compute tiles to complete a GCN layer.

2.3.4 Mapping Dataflow to Hardware

Transformation

For transformation, the core is reconfigured into the systolic module. Each core has a PE array of R rows and C columns. Thus, each core can handle R rows of the feature matrix and C columns of the weight matrix. We have M such cores operating in parallel.

Fig. 2.5(a) shows the dataflow for transformation. Weights are fed from the top and is pushed to the systolic module, and features are fed from the left and pushed to the right in every cycle. To access data parallelly in every cycle, we have a feature matrix with banks set to R and weight matrix with banks set to C. For M parallel cores, we

have M on-chip feature buffers. Since weights are the same within a GCN layer, we only need one weight buffer and the data is broadcasted to all cores. The FT scheduler generates the address to read data from the feature buffer and the weight address. Since all M cores produce output at the same time, we have the output buffer with M banks to enable parallel output writes. The peak parallelism we achieve here is M×R×C.

Aggregation

At the tile level, to improve the regularity in memory access and data reuse in the aggregation phase, we further partition each tile into smaller intervals. For illustrative purposes we assume that we have three cores in the design. The destination nodes in each tile is equally split according to the number of cores. In this example we split into three sets and the destination nodes and its associated edges in each set is assigned to a unique core for computation as shown in Figure 2.5(b). The edges in each set represented in the COO format is populated inside the edge buffer of each core. The edge buffer has additional instructions/configurations information that is required for the core to compute the set of edges mapped to it. The entries in the edge buffer with MSB 2 bytes as FFFF and EFFF are configuration instructions and memory read instructions respectively to the core and the lower two bytes are the associated information such as number of edges, read memory location etc.

For horizontal aggregation, the neighbors or source nodes of a tile are the same for all cores. Since all the cores compute in parallel, accessing the features of all the source nodes for aggregation will result in a memory bottleneck. Therefore we use a banked on-chip memory to store the source nodes. As the maximum number of parallel read requests equals to the number of cores, the number of banks for the on-chip memory is then set to the number of cores. Each consecutive source node feature is statically mapped into a unique bank as shown in Fig. 2.5(d). The edges fed into each core's edge buffer are arranged in the ascending order of source nodes. This increases the probability of multiple edges with the same source nodes or with the next source nodes to be computed one after the other. To exploit data reuse and locality in this data flow, whenever a core makes a read request to a specific address location, features of the source nodes at that address location across all banks are fetched and moved to that core's local buffer.

This we ensure high data reuse and exploits data locality with in a small segment inside the tile.

For non diagonal tiles vertical aggregation is also computed. For vertical aggregation, the destination and source nodes are interchanged and each set of the source nodes is unique to a core. Therefore, a separate on-chip memory is set up for each core to store the features for vertical aggregation. In the case of vertical aggregation, each core produces an aggregated partial output for the same destination nodes, which needs to be combined to from the final output. We use a reduction engine which adds up the partial output from each core into a single output. Moreover, the reduction engine computes the output from the previous tile while the current tile is being executed, thereby hiding its latency. The edges are fed into the core sequentially. For each edge, it takes 4 clock cycles for the PE to compute. In each clock cycle, the aggregation scheduler fetches a new edge from the edge buffer and decodes the data to check whether it is a configuration instruction or an edge data. For each edge, the aggregation scheduler checks the PE status within the core for the availability of a free PE row. If all PE rows are busy, then aggregation of that corresponding core will be stalled until a PE row becomes available, or else the edge is assigned to a free PE row and its status is updated. To compute vertical and horizontal aggregation for non-diagonal tiles, the scheduler checks if two PE rows are free; if not, aggregation of that corresponding core will be stalled until two PE rows become available.

To improve the compatibility of input data to the custom flow, we further develop pre-processing codes using PyG [12]. The pre-processing codes accept the number of cores and tile size as input and generate the edges in order at the tile level following the custom dataflow. The generated output is then used to populate the main memory. The DMA module along with the top level scheduler will move the data tile by tile per the computation order.

2.4 Results

2.4.1 Experimental Setup

We implement our design in Verilog and use the Intel Stratix10 MX board with HBM2 as our target platform. The FPGA has 8Gb of HBM2 memory, 2,073k logic elements,

Table 2.1: Characteristics for four different GCN datasets.

	Cora	**Citeseer**	**Pubmed**	**Reddit**
Nodes	278	3327	19717	232965
Edges	10556	9228	88651	114,615,892
$\mathbf{X}^0$	1433	3703	500	602
$\mathbf{X}^1$	128	128	128	128
$\mathbf{X}^2$	7	6	3	41
Layers	2	2	2	2

702k ALMs, 134Mb of M20K memory, 11Mb of MLAB memory and 3,960 DSPs. We use Intel Quartus 19.4 to map the design to FPGA. We use 8-bit fixed-point precision to represent weights and features, and 32-bit to represent each edge in COO format.

We fix our design parameters, including the number of rows of PE array in a core to 10 and the number of columns of PE array to 66. Thus a single core will use 660 DSPs and we can have 6 such cores totalling the DSP count to 3,960 on the FPGA board. Increasing the number of columns in a PE array will result in higher parallelism along the feature dimension; increasing the number of rows of a PE array will increase the number of nodes that can be computed in parallel. On the other hand, if the size of a column or a row is too big, it will negatively impact such parallelism. Therefore, in our design across the 6 cores, we select our total array as 60×66 to ensure a balanced parallelism. The tile size we use to partition the graph is set to 1,020. The impact of the tile size on the design are studied and reported in [15]. Similar to that, we select the smallest tile size to highlight the speedup from the symmetry-based acceleration. We measure the end-to-end latency of a two-layer GCN for the four widely used datasets: Cora, Citeseer, PubMed, and Reddit. The details of the datasets are shown in Table 2.1.

2.4.2 Quantization of GCN Algorithms

To reduce the memory footprint, we experiment GCNs with lower precision. Fig. 2.6 presents the accuracy of the GCN for four different datasets with various bit precision values for both weights and features. We use the GCN structure in [5] and Deep Graph Library (DGL) [13] with PyTorch in this experiment. The accuracy drop is negligible for all datasets when we move from 32-bit floating point to 8-bit fixed-point precision.

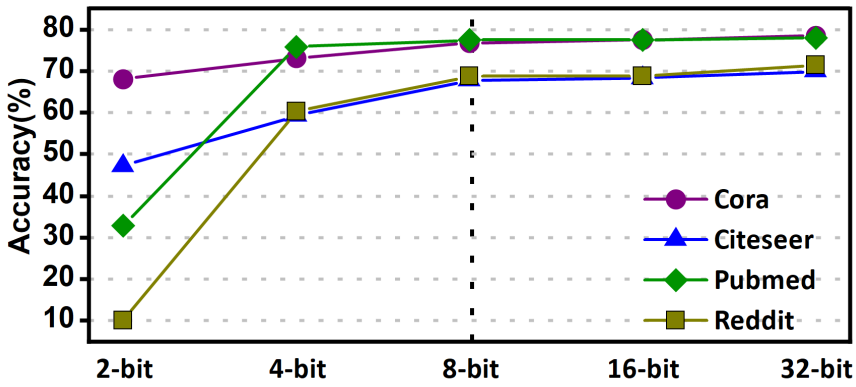

Figure 2.6: Quantization of the GCN models. 8-bit is selected before experiencing any accuracy loss.

The difference between the baseline accuracy and accuracy at 8-bit is less than 3% for all four datasets. Beyond 8-bit, the accuracy rapidly drops for all datasets. Thus we adopt 8-bit quantization for weights and features since it provides comparable accuracy with 32-bit precision.

2.4.3 Speedup with the Symmetry Property

In this section, we compare the performance of two implementations, one with the symmetry property and the other without. Fig. 2.7 presents the number of tiles that need to be computed for the four datasets under the baseline case without the symmetry property and with the symmetry property. Our method requires up to 50% less compute tiles to complete end-to-end GCN computation. The reduction is more pronounced for larger datasets. Furthermore, we set up a same implementation on FPGA, except the symmetry flow is not adopted. For Cora, the end-to-end latency without the symmetry property is 108.81µs. The adoption of the symmetry is 1.7× faster than that.

2.4.4 Comparison with State-of-the-Art Results

We compare our work with the state-of-the-art graph learning frameworks PyG [12] and DGL [13]. We also compare our design with the state-of-the-art FPGA implementations ASAP2020 [16] and BoostGCN [15]. We synthesize our design at 220MHz. The resource utilization on the FPGA board is summarized in Table 2.2. Table 2.3 summarizes the

Table 2.2: Resource utilization of FPGA for different datasets.

Dataset	ALMs	RAMs(MB)	DSPs
Cora	257208 (36.6%)	5.23 (28.86%)	3960 (100%)
Citeseer	243624 (34.67%)	6.12 (33.77%)	3960 (100%)
Pubmed	300288 (42.73%)	6.34 (34.98%)	3960 (100%)
Reddit	292441 (41.59%)	15.96 (88.06%)	3960 (100%)

Table 2.3: Comprehensive evaluation of execution time for different datasets across different implementations.

Datset	PyG CPU	PyG GPU	DGL CPU	DGL GPU	HyGCN	ASAP2020	BoostGCN	Our Work	Over BoostGCN
Cora	17.1ms	945µs	12.7ms	1.1ms	21µs	3.5ms	76.5µs	**62.77µs**	**1.20x**
Citeseer	22ms	1.5ms	17ms	1.2ms	300µs	11.1ms	125.8µs	**100.37µs**	**1.25x**
PubMed	229ms	3.4ms	22ms	1.3ms	640µs	9.5ms	1140µs	**882µs**	**1.28x**
Reddit	81s	out of mem	3.2s	390ms	289ms	598.7ms	98.1ms	**73.51ms**	**1.33x**

comparison with the latest GCN accelerators, with Table 2.4 evaluates the hardware resources across multiple design. Our resource is close to that in BoostGCN [15].

Compared to PyG and DGL, our work achieves up to 1,101× and 202× improvement in end-to-end latency in CPU and up to 13× and 17× in GPU implementations, respectively. Despite the vast difference in computing resources, our FPGA design achieves significant speedup over both the CPU and GPU implementations. These results confirm that CPUs and GPUs are more suitable for dense computations with regular memory access. On the other hand, the regular structure in CPUs and GPUs limits their capability in managing the heterogeneous and irregular nature of GCN computations.

We also achieve up to 110.5× improvement compared to ASAP2020 [16]. ASAP2020 uses a two-phase graph pre-processing, by removing the edge connections of high degree nodes and merging common neighbours together, and then use graph reordering algorithms to improve data locality.

We achieve up to 1.3× improvement over BoostGCN [15]. The speedup increases as the dataset size increases. This is because as the graph gets bigger, the reduction in the number of compute tiles from the symmetry property is more pronounced. BoostGCN employs sparse feature transformation which skips zero elements in the feature vectors

Table 2.4: Hardware resource used by various designs.

	CPU	GPU	HyGCN [14]	ASAP2020 [16]	BoostGCN [15]	Ours
Compute Resources	Intel Xeon Gold5120 CPU 28 cores and 56 threads	Titan Xp 3840 cores	32 SIMD cores 8 systolic module	128/256 accumulators 24x24 systolic array	294k ALMs 3840 DSPs	**292k ALMs** **3960 DSPs**
Frequency	2.20GHz	1405MHz	1GHz	250MHz	250MHz	**220MHz**
Memory	20.8MB cache	48KB L1 cache 3MB L2 cache	24MB	N/A	18MB	**18MB**

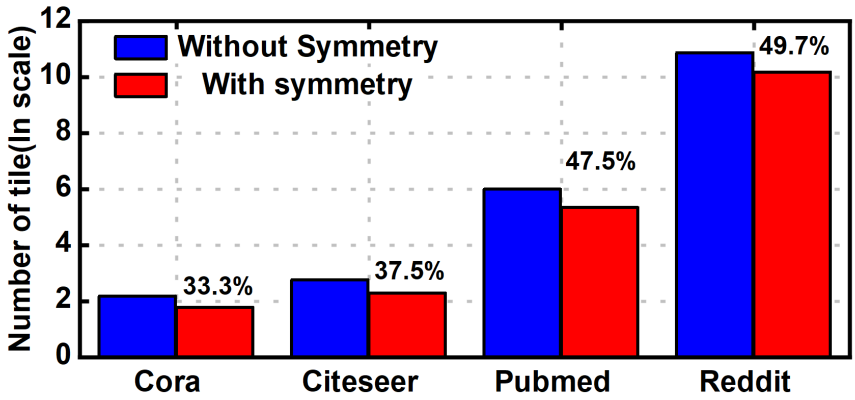

Figure 2.7: The symmetry property effectively helps reduce the number of compute tiles in GCN acceleration.

during transformation. For Cora, Citeseer and Pubmed, it stores the entire input feature matrices on the memory. BoostGCN also partitions the graphs with a tile size of 4,096 for Cora, Citeseer, and Pubmed, and 16,384 for Reddit. For Reddit, BoostGCN reports a 21% reduction in external memory access for the tile size of 16,384 over the tile size 1,024.

One thing we can further improve is the implementation of double buffering, i.e., the ping-pong strategy to hide the latency in data movement from off-chip memory. Both ASAP2020 and BoostGCN use this strategy for acceleration. They also use dedicated cores for transformation and aggregation. Compared to ASAP2020 and BoostGCN, our improvement mainly comes from the reconfigurable core architecture, which helps overcome the pipeline stall, and the design with the symmetry property, which enables us to produce the same number of outputs by using approximately half the number of

compute tiles. In the future, we plan to add double buffering since we still have enough memory resource.

Finally, we evaluate our result with HyGCN [14], a ASIC based implementation in TSMC 12nm CMOS. It has 4,096 32-bit fixed-point multipliers and 512 32-bit fixed-point ALUs running at 1 GHz, as compared to our FPGA design with 3,960 DSPs and at 220MHz. Despite having a vast difference in terms of computing resources and frequency, our FPGA design outperforms HyGCN for Citeseer and Reddit, by 3× and 3.9×, respectively. This is because HyGCN does not consider the inherent property of the graph, such as the symmetry property, which limits its performance for bigger datasets.

2.5 Conclusion

This chapter an FPGA GCN accelerator that can adapt to the heterogenity that exists within an AI model. Sparse feature aggregation is the primary bottleneck in the acceleration of GCNs. The work leverages the symmetry property of the adjacency matrix in an undirected graph to accelerate feature aggregation, achieving end-to-end GCN acceleration. A custom dataflow is proposed based on the symmetry property and designed a hardware accelerator architecture to support the dataflow. This architecture is paired with a software-based pre-possessing method to maximize data locality and to reduce memory access. The design achieves up to 1,101×, 17×, 110.5× improvement over CPU, GPU and other FPGA implementations. Further improvement can be achieved through the implementation of double buffering and lower-precision in PE design.

Chapter 3

A 16nm Heterogeneous Accelerator for Energy-Efficient Sparse and Dense AI Computing

3.1 Introduction

Deep learning has transformed application domains such as image processing, autonomous driving, and language processing. Various types of AI models, like Transformers [23] and GCNs [5], have been proposed for natural language processing and graph processing tasks. DNN models involve MAC computations, which are both compute and memory-intensive [24]. Dedicated accelerators have been proposed to achieve efficient CNN acceleration [25, 26]. With bigger models, having separate memory and compute units poses great challenges to these traditional accelerator architectures [26]. To address this, In-Memory Compute (IMC) was introduced to move the computation to the memory where data is stored. While these custom accelerators achieve state-of-the-art performance, they are only optimized for a limited set of DNNs, lacking programmability to keep up with the rapid evolution of DNNs [24, 27]. IMCs are efficient at dense compute bound MAC operations with high operation intensity (i.e., the number of compute operations performed on each of the loaded data) [28]. Other computations such as

Table 3.1: GCN and SpMV workloads are dramatically sparse.

Dataset	**Domain**	**Dimension**	**Sparsity**
Cora	GCN	2708×2708	99.82%
Citeseer	GCN	3327×3327	99.89%
Consph	2D/3D Problem	83334×83334	99.9%
Ohne2	Semiconductor Device	181343×181343	99.9%

sparse matrix vector multiplication (SpMV) and feature aggregation in GCNs are characteristically different than dense MACs. This reduces the emphasis on data movement cost and high compute efficiency that motivates IMC use [28].

These challenges cannot be addressed using conventional IMC-based CNN accelerators. CNN models exhibit high data reuse and predictable memory access patterns, whereas the random distribution of graph edges in GCNs leads to poor data reuse and unpredictable memory access during aggregation operation in GCNs. GCNs and SpMVs involve computations on sparse matrices, and storing sparse matrices on IMCs is inefficient [29]. Since these workloads are 99% sparse, as shown in Table 3.1, transferring these zeros from memory and storing them in Latch-based IMCs (LIMCs) lead to significant bandwidth, energy wastage and underutilization of LIMCs. To address this, we propose the use of SIMD-based compute cores to effectively handle the sparse computations. Figure 4.2 bottom, shows the volume of data needed to do sparse computation using LIMC and SIMD cores for GCN and SpMV. For evaluation with LIMCs, we follow the same method as in [30], where the extremely sparse matrix is mapped entirely to the IMC crossbar. SIMD cores operate on sparse data represented in compressed formats like COO, CSR, eliminating the need to move zeros from memory, achieving up to a 99% reduction in the volume of data fetched compared to IMCs. SIMD cores achieve up to 149x more operational intensity (i.e. the number of compute operations performed on each of the loaded data), than LIMCs as shown in Figure 4.2 top. Since the adjacency matrix is 99% sparse, LIMCs effectively compute only 1%, whereas with SIMD cores, only the non-zero entries of sparse matrices are fetched from memory. This results in higher operational intensity. Additionally, to exploit activation/input sparsity at the input of LIMCs we propose the use of operation skipping and booth multiplier to achieve coarse and fine grained operation skipping. Though IMCs are efficient for

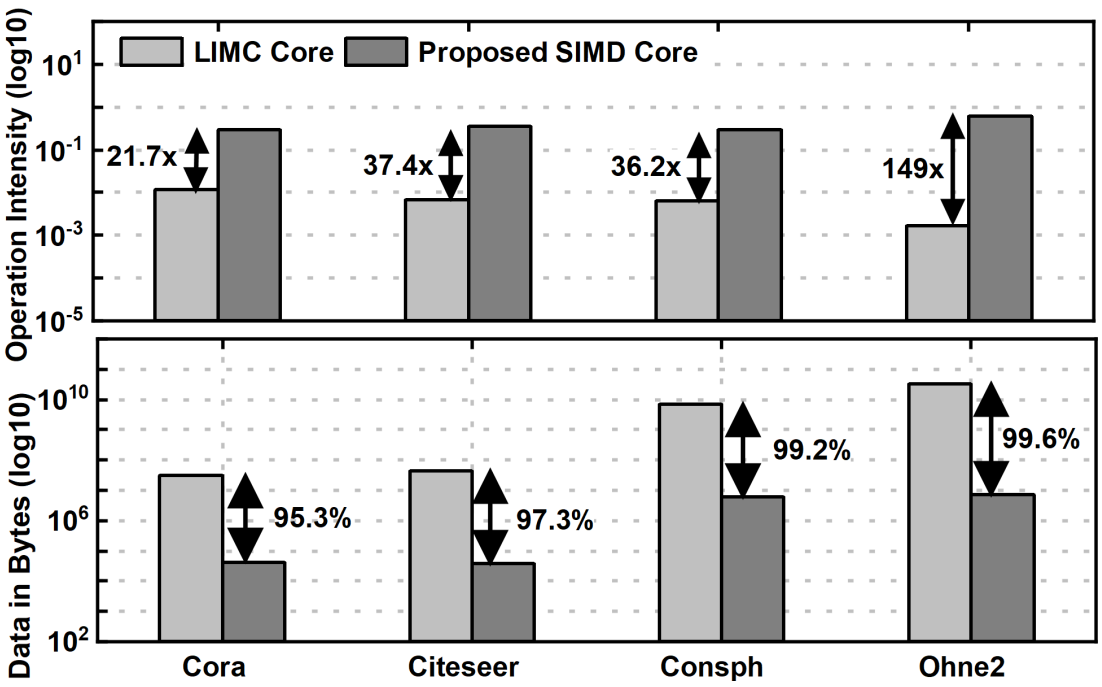

Figure 3.1: Mapping of sparse workloads on IMCs is inefficient, while our proposed SIMD cores effectively improve computational efficiency and data access.

dense MACs [27], they still incur hardware virtualization overhead, such as data loading leading to significant performance loss [28, 31]. To this end, we propose the use of a ping-pong LIMC core to efficiently hide the weight update cost by overlapping it with the computation. Since CNN and GCN involve both dense and sparse computations, we chose them as representative heterogeneous AI workloads.

Additionally, the development of highly flexible DNN accelerators is crucial for making hardware adaptable to software changes [24]. To address these issues, we have designed a programmable heterogeneous architecture capable of simultaneously accommodating the regularity of CNN computations and the irregularity of GCN operations. We have developed an instruction set architecture (ISA) that supports various memory access patterns, dataflows, and computation patterns, enabling efficient mapping of workloads to the accelerator. The major contributions of this work are:

- A heterogeneous RISCV based SOC accelerator, with a LIMC to manage the dense operations, and a SIMD core for the sparse operations.

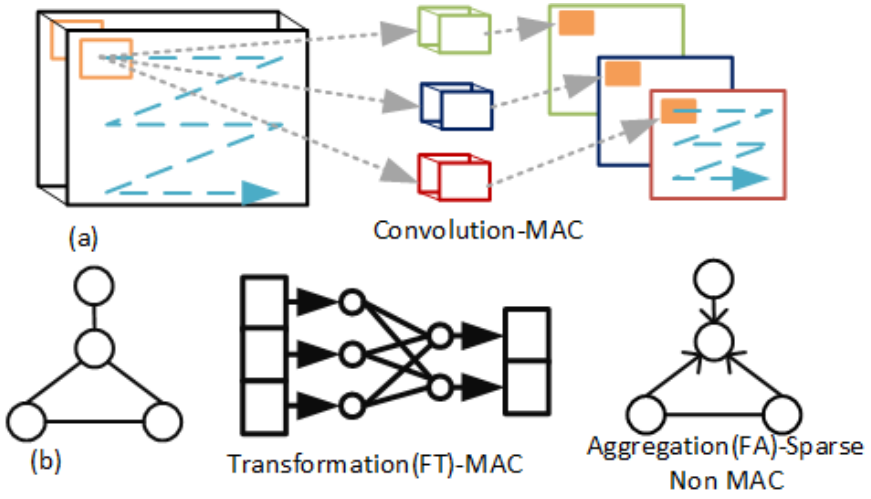

Figure 3.2: Comparison of Heterogeneous AI models: (a) Conv MAC operations in CNN with high data reuse predictable memory access patterns (b)GCN model showing feature transformation(MAC) and aggregation(non-MAC) with random memory access and sparse compute.

- Operation skipping and booth multiplier to handle activation sparsity at the input of LIMC.
- Ping-Pong LIMC design to efficiently handle weight update overheads of IMC accelerators.
- 16nm implementation and tapeout of the proposed accelerator with comprehensive evaluation of the performance with state-of-the-art accelerator designs.

3.2 Background and Motivation

CNNs are primarily designed for grid-like data structures such as images. CNNs leverage convolutional layers to extract local features and hierarchical representations from the input data. They use shared weights and sliding convolutional filters to exploit spatial locality as shown in Figure 3.2 (a). On the other hand, sparse AI models such as GCNs are specifically designed to operate on graph-structured data, where the data are represented as a set of interconnected nodes and edges, as shown in Figure 3.2 (b). Aggregation, which depends on the graph structure is a sparse operation and

leads to very random memory accesses and poor data reuse opportunities. Existing CNN accelerators developed to achieve high-throughput CNN inference by exploiting the predicatble memory access patterns and data reuse do not generalize well to sparse workloads like GCNs or SpMVs because of the unpredictable nature of sparse workloads. Also the sparsity in the sparse workloads are much higher than the typical sparsity in CNNs.

CNN accelerators deploy a wide range of compute units like PE array [26], Systolic arrays [32], IMC-based systems [33]. These compute units excel at operations that involve extensive data reuse and data parallelism, such as the MAC operations in convolutions and fully connected(FC) layers. Therefore, they are well-suited for the feature transformation operation in GCNs, which involves performing dense matrix multiplications, but they are unsuitable for sparse random memory access non-MAC operations like the aggregation in GCN.

CNN accelerators also deploy methods like Run-Length Compression [26] to exploit the sparsity in activation generated in the deeper layers. They are mostly aimed at saving energy and DRAM bandwidth. Thus aggregation needs hardware that offers flexibility, finer control over sparse operations, and programmable element-wise operation as compared to the massively parallel MAC engines.

Several accelerators [6, 14, 34, 35] have been proposed to enhance the efficiency of GCN acceleration. Accelerators such as IGCN and [35] propose novel algorithms to accelerate feature aggregation in GCNs. IGCN [34] introduces the concept of islandization, harvesting the data locality exposed through islandization and avoiding redundant aggregation among shared neighbors. [35], on the other hand, exploits the symmetry property of graphs. By recognizing and leveraging symmetric patterns in the graph structure, reduces redundant computations during feature aggregation, thereby enhancing the efficiency of GCN inference.

These algorithms contribute to accelerating feature aggregation in GCNs and improving the overall efficiency of GCN accelerators. The architecture and data flow these of accelerators are specifically designed for efficient GCN inference and are not suitable for accelerating CNNs. They are optimized to exploit the characteristics and computational requirements of feature aggregation and the pipeline flow between feature transformation

Table 3.2: Workload characteristics of CNNs, GCNs and SpMV.

Workload	**Kernel**	**Type**
Convolution	MAC	Dense/Sparse
Fully connected, MLP	MAC	Dense/Sparse
Feature transformation	MAC	Dense/Sparse
Feature Aggregation	Sparse matrix	Very Sparse
SpMV	Sparse matrix	Very Sparse

and aggregation. Therefore, their design and optimizations are tailored toward GCN inference rather than CNN acceleration. COIN [30] proposes an IMC-based solution to accelerate GCNs. They store the extremely sparse adjacency matrix in the IMCs which is inefficient. It leads to huge energy wastage in moving the zeros from the main memory and writing them into IMC crossbars. Thus IMCs are extremely inefficient in handling non-MAC sparse computations like aggregation.

Thus to handle the heterogeneity in the AI models we need a reconfigurable heterogeneous accelerator that can adapt to the memory access patterns and the computation needs of the AI models. A thorough analysis of the workloads is conducted to understand their characteristics and requirements. Table 3.2 highlights the main differences in CNN, GCN and SpMV workloads. In CNNs, 99% of computations are MAC operations [24]. In GCNs, FT is a MAC operation, whereas FA and SpMV are sparse computations.

Other operations like activation functions are common to AI models and make up only a small fraction of the total computation. Input sizes differ across different models and within a model as well. In the case of GCNs, the dimensions of the adjacency and feature matrices vary depending on the dataset. To accommodate the heterogeneous nature of AI models with varying memory access patterns, model sizes, and computation kernels, it is necessary to have reconfigurable heterogeneous hardware that can effectively support these requirements. Reconfigurable heterogeneous hardware offers the flexibility to adapt to different models and their specific characteristics, enabling efficient execution of diverse AI workloads.

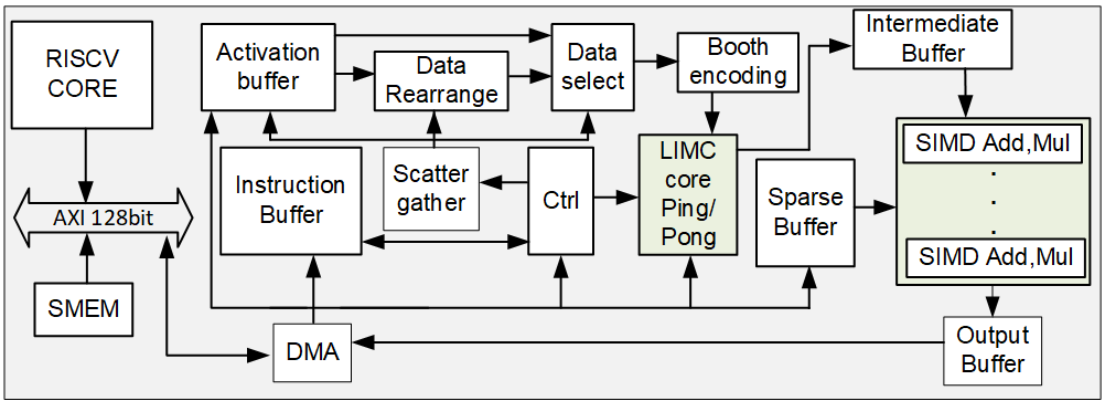

Figure 3.3: Architecture of the heterogeneous accelerator with ping-pong LIMC and SIMD cores.

3.3 Architecture

3.3.1 Architecture Overview

To achieve a balanced operational intensity across dense and sparse workloads, we integrate a heterogeneous AI accelerator with LIMC and sparse SIMD core.

IMCs, with their massive parallelism, can efficiently perform MAC operations in CNNs and FT in GCNs [27, 30]. IMC based architectures accelerate MACs by fusing memory access and computation [30], However, the integration of IMCs into a programmable processor has suboptimal efficiency and throughput, due to the overhead in updating the weights of IMCs, inefficient weight mapping in the crossbars, and the substantial volume of activation data required to keep IMCs fully utilized [36]. To address the weight update overhead, we introduce ping-pong IMCs to hide the data movement latency. Then, we define an ISA that is used to efficiently map the workload to IMCs, and generate different memory access patterns and data flow for efficient data delivery, ensuring full IMC utilization. IMCs are inefficient in computing the sparse FA in GCNs due to a lack of support for compressed data formats and poor operational intensity. In this work, we propose the use of SIMD adder units to execute non-MAC sparse computations. They offer fine-grained control and high operational intensity to manage the irregular and sparse workloads of GCNs. To accommodate the heterogeneous nature of AI models and address the overheads in IMC accelerators, we design a programmable

heterogeneous architecture with ping-pong latch-based IMC and SIMD cores. We use programmable scatter-gather and control logic to handle the different memory access patterns and computations associated with the heterogeneous workloads. Figure 4.4 shows the architecture of the SOC system. It comprises of a RISCV core,latch-based IMC (LIMC) and SIMD compute cores, global buffer, instruction buffer, edge buffer, activation buffer (AB), and reconfigurable scatter-gather (SG) units and a top control logic. The LIMC is responsible for the MAC operations. The non-MAC FA is assigned to the SIMD core. The edge buffer stores the edges of the graph. The weights and activations for CNN and GCN are stored in the global buffer. The activation buffer holds the tile of activation under computation. The instruction buffer stores the instructions. The reconfigurable scatter-gathering (SG) units will be configured according to the instructions, and the SG feature unit handles the memory access patterns of CNN and GCN, ensuring efficient data retrieval according to the workload requirements and provides it as input of LIMC. The data rearrange module consists of line buffers and logic to select data for convolutions. DMA is responsible for fecthing the data from the global buffer and populating the activation buffer and IMCs based on the instructions being executed. The accelerator and global buffer SMEM is connected through a 128 AXI interconnect. The global buffer is a 512KB SRAM memory. We use the open source CVA6 RISCv. It is a 64 bit CPU core with a 5 stage pipeline.

3.3.2 Latch-based IMC (LIMC) Core

LIMC architecture

The building block of LIMCs is a latch and is equivalent to the bitcell in SRAM IMCs. LIMC supports both 8-bit and 4-bit computations. For 4-bit, either the upper or lower nibble of weights and activations are selected based on the mode signal set by the ctrl unit. To balance the area constraint and parallel MAC operations, we divide the LIMC macro into banks, and computations happen parallelly across banks. Each bank is connected to a modified booth multiplier. With the booth, the number of partial products is halved [37] compared to bit-serial computing in SRAM or RRAM-based IMCs, resulting in faster computations.

The number of banks is optimized to achieve a balance between the area and the compute latency. As the number of banks increases, the number of booth multipliers

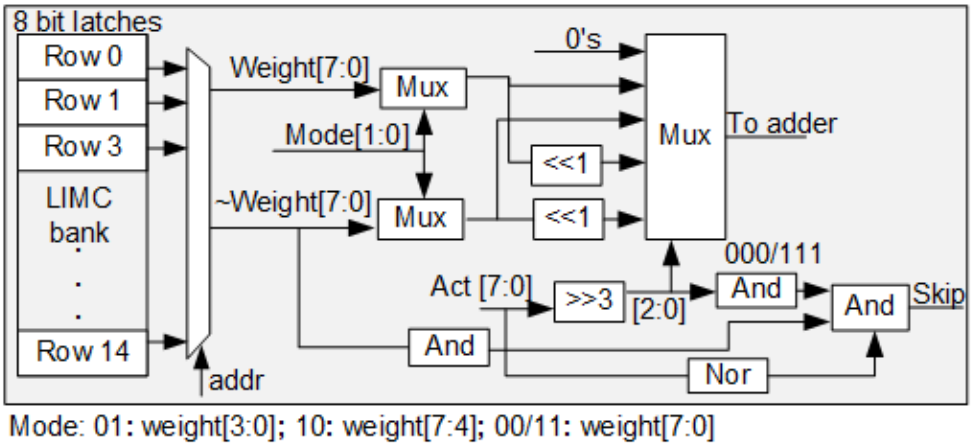

Figure 3.4: Microarchitecture of the proposed LIMC bank with the booth multiplier. Operations are skipped at the coarse level if the weights/activations are zero and at the fine level if the booth encoding bits are 000/111.

required for parallel computation increases, resulting in an increase in area. The compute latency decreases with the number of banks as the number of iterations required is lower, leading to lower latency. The number of iterations is defined as the size of the input in the LIMC divided by the number of banks. As illustrated in Figure 3.5 we select the number of banks to be 30, to achieve an optimum balance between area and latency. The microarchitecture of a LIMC bank with booth multiplier is as illustrated in Figure 3.4. Eight columns of 1-bit latches share one set of booth multiplier. To achieve maximum weight mapping efficiency in LIMC for the common convolution use cases, the number of rows in a bank is set to 15, a multiple of 3 and 5, the most common kernel sizes. Thus, with 30 banks containing 15 rows each, the total number of rows in the LIMC is 450. The system bus is 128 bits wide. Therefore, we set the number of columns in the LIMC to 128, which is equivalent to 16 columns of 8-bit latches, and with an 8-bit latch, the number of macros inside the LIMC core is set to 16 to match the interconnect bandwidth. The output generated by the LIMC is stored in a banked intermediate buffer with 16 banks, facilitating parallel LIMC output writes. Iterations are skipped when all the data at the input of the banks are zero. For a smaller number of banks, it is easier to find groups of zero inputs, leading to a higher number of iterations skipped.

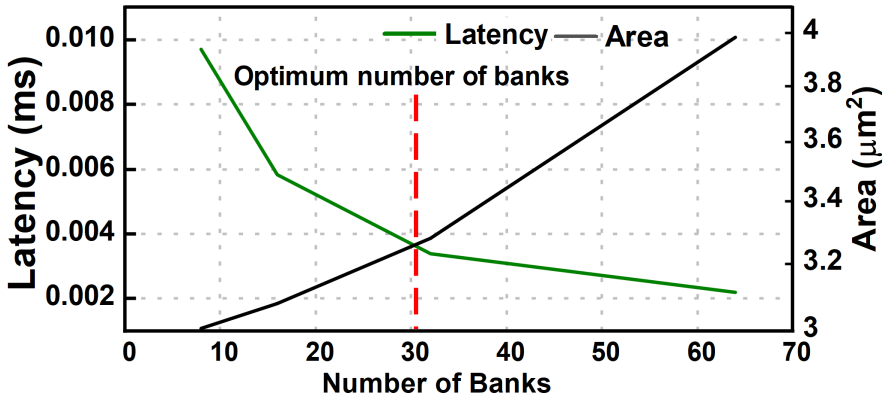

Figure 3.5: A smaller number of LIMC banks have smaller area due to less number of multipliers, but will also have more iterations (the size of the input/the number of banks) at the input of LIMC increasing latency.

Activation Sparsity

LIMC core dynamically handles the activation sparsity at the input by using operation skipping and the booth multiplier. This ensures sparsity handling at a coarse and fine granularity. At the coarse level, if all activations at the LIMC bank input are zero, the computations will be skipped. At finer granularity, if the encoding of the booth is 000 or 111 the computations will be skipped, as shown in Figure 3.4.

Ping-Pong IMC

For efficient integration of IMC-based accelerators, the overhead of weight updates should be addressed. Updating IMC weights between computations leads to a significant performance loss [31]. Existing work on IMC-based accelerators, such as [30,38,39], does not address this overhead. The IMC macro-based work does not consider the overheads of the efficient integration of macros into a system, whereas multicore or multichiplet designs can store the entire model or layer in IMCs, reducing weight update overheads. This work explores the efficient integration of IMC-based accelerators in an area-constrained system, where only a limited number of IMCs can be placed. Therefore, the overhead of writing the IMCs with a new set of weights for different operations should be addressed. Efficient mapping of workload into the IMCs and timely delivery of compute data are

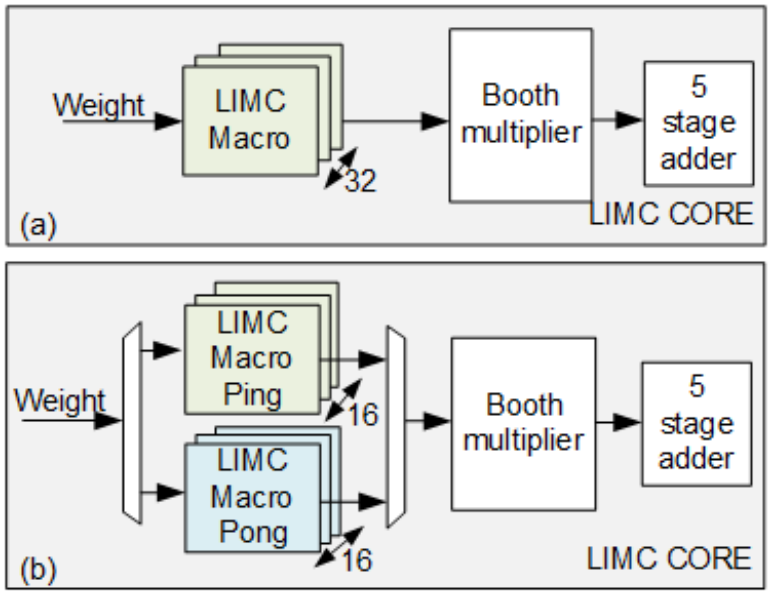

Figure 3.6: (a) Conventional IMC core where computation and weight updates are sequential. (b) Proposed Ping-Pong based LIMC core to efficiently hide the IMC weight update latency by overlapping the computation and IMC weight updates across the ping-pong cores.

critical to achieving high crossbar utilization and compute efficiency. To this end, we proposed the use of a ping-pong LIMC core to efficiently hide the weight update cost by overlapping it with the computation. Figure 3.7 shows an illustrative toy example with three independent workloads. With only one LIMC core as in Figure 3.6(a), weight writes and execution can only occur sequentially. To hide this overhead, we use the ping-pong LIMC architecture Figure 3.6(b), where one core will perform MAC computations while the other will load the new set of weights for subsequent computation. Only the LIMC structure is replicated, the adder tree, and the booth encoding logic is common for both LIMCs, and data is selected appropriately using mux. The amount of improvement will also depend on the nature of the workload. If the IMC write overhead is negligible compared to the execution time, then it will only bring modest improvements. In these scenarios, users can decide whether to use ping-pong or disable that feature by turning off one of the ping/pong LIMC using instructions.

Table 3.3 shows the improvement from the proposed ping-pong LIMC for matrix

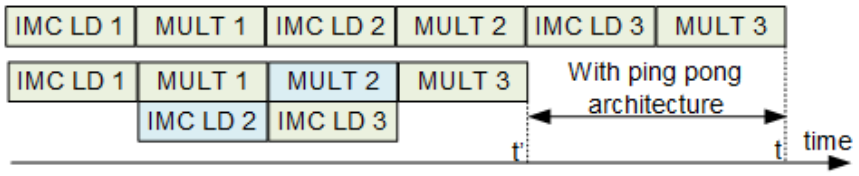

Figure 3.7: Proposed ping-pong IMC hiding data movement latency by overlapping weight load and computation.

Table 3.3: Improvement in compute latency achieved from the proposed ping-pong IMC architecture.

Mat A	**Mat B**	**Clock Cycles w/ ping-pong**	**Clock Cycles w/o ping-pong**	**Reduction in Clock Cycles**
(10,1433)	(1433,16)	3794	6320	39.9%
(100,1433)	(1433,16)	33494	44570	24.8%
(400,1433)	(1433,16)	132494	172070	23%
(1000,1433)	(1433,16)	339102	422820	19%

multiplication workloads. We achieved up to 39.9% reduction in clock cycles with the proposed ping-pong architecture. Matrix B is mapped to the LIMC, and its size is set to be larger than the number of rows in the LIMC to ensure ping-pong operations. When Matrix A is smaller, writes to the LIMCs dominate the latency, maximizing the benefits of the ping-pong solution. However, as the size of Matrix A increases, execution time becomes the dominant factor, reducing the gains of the ping-pong architecture.

SIMD Core and Sparse Buffer

SIMD cores compute the sparse workloads. Figure 3.8 illustrates the architecture of a SIMD core and the microarchitecture of a Processing Element (PE) within the core. The core comprises of PE rows with N columns, where each PE has its own scratch pad (SPAD) for storing partial results for reuse. The SIMD core has both intra-PE row and inter-PE row parallelism. For CNN workloads, the SIMD core is utilized to reduce the partial sums generated by the LIMC across the input channels. During feature transformation, it reduces the partial results obtained from matrix multiplication.

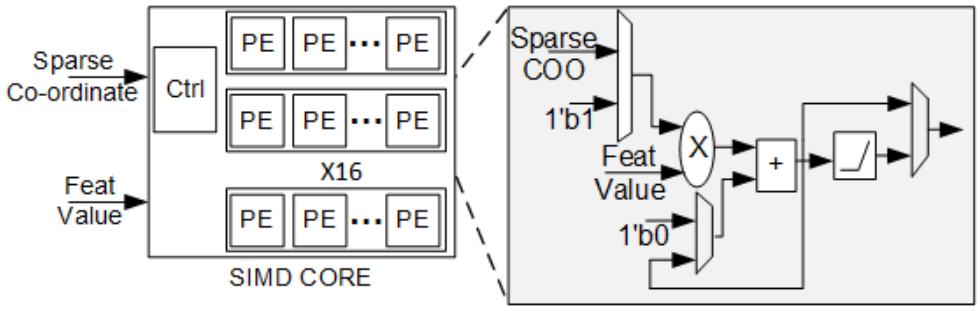

Figure 3.8: Micro-architecture of SIMD core.

During aggregation, the SIMD core can scale the feature values of neighboring nodes using an edge value, the scaling factor. If no scaling is required, the edge value can be set to one. Subsequently, the SIMD core performs the addition of scaled features of each target node's neighbors to itself. The edges stored in the Coordinate (COO) format within the edge buffers will be decoded by the control unit of the SIMD core to identify the get target node and its neighboring node. To obtain information on neighboring nodes, the SIMD core control unit decodes the edges stored in the Coordinate (COO) format within the edge buffers. The edge buffer holds the edges of the graph in COO format. The adjacency matrix is partitioned into smaller tiles to fit into on-chip memory [35]. The sparse buffer holds the edges of the graph or the entries of the sparse matrix in COO format. For aggregation, based on the edge information, the control logic initiates read to the intermediate buffer to fetch the features of the neighboring nodes and then issues the aggregation to one of the PE rows. During SpMV the control logic fetches the vector elements from the intermediate buffer based on the sparse matrix entries in COO format. The intermediate buffer can be populated with LIMC output or the DMA engine can populate it with the required inputs. The size of the edge buffer is set to 16KB. The graph features or vector elements are stored in the intermediate buffer. With 16 banks for the intermediate buffer, 16 reads can be performed in parallel. Therefore, the control logic can issue 16 edges or 16 rows of sparse matrix in a cycle. Thus, the number of PE rows is set to 16. For each SIMD core, the number of columns in a PE is set to 128, supporting the feature dimensions of typical graph datasets. In cases where the dimension exceeds 128, it will be executed iteratively. The SIMD core can also be used as a ReLU activation unit for convolution operation, where the LIMC output can

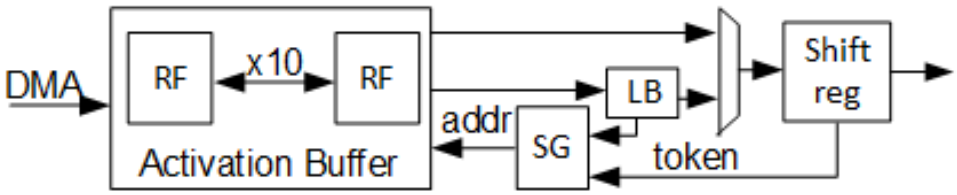

Figure 3.9: Micro-architecture of SIMD core.

be fed to the SIMD core through the intermediate buffer.

Data Delivery Modules

The DMA, scatter-gather (SG), data rearrange, data select and activation buffer (AB), and a 512KB global buffer make up the data delivery modules as shown in Figure 3.9. Since feature fetches of FA are dynamic, the read logic for SIMD is embedded inside the SIMD control logic. One of the overheads associated with IMC-based design is the volume of activation data that must be delivered consistently at the IMC input. We have designed a dataflow with multiple hierarchies of memories to ensure pipelined data delivery at the input of the SIMD core. The load instruction will configure the DMA module to fetch the tile of data for computation from the global buffer to the activation buffer. AB is double-buffered. SG will generate the address based on the workload data access patterns. For convolution, the data is fetched from AB and then stored in the line buffers (LB) inside the data rearrange module. Based on the stride and kernel size, the data will be selected from the LB. The data select module will choose the data directly from the activation buffer for matrix multiplication or from the data rearrange logic for convolution and write them into the shift register (SR) inside the booth encoding block. The SR is a two-entry flip-flop register. The entire data flow is orchestrated with the help of token-based handshaking. The LB and SR have a token register (TR) to indicate full status. If TR is not full, the SG unit will write new data to it. This decouples the compute latency of the IMC from the dataflow and ensures that the data is always present in the buffers for computation. The accelerator consists of a memory-mapped 512KB global buffer. The entire address space of the global buffer can be divided between activation, weight, and instructions. The addresses can then be encoded into the LD/ST instruction fields, achieving high programmability.

Table 3.4: Instructions and the hardware used.

Instruction	Opcode	HW used
Load	001	DMA
Store	010	DMA
Matmul	011	SG, LIMC
Conv	100	SG, LIMC, Data Rearnage
Agg	101	SIMD

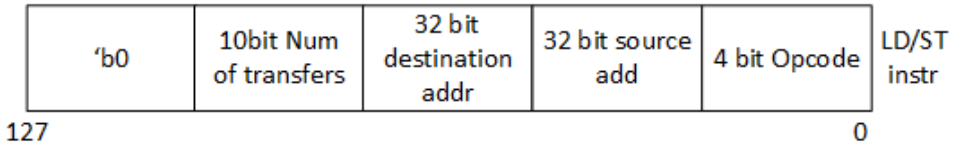

Figure 3.10: Example of a LD/ST instruction.

If the token is zero, the LB is empty, and if it is equal to the size of LB then full. TR is incremented when a write is made to the buffers and decremented when read, and it automatically puts back pressure to the previous modules, ensuring no new writes are done when LB is full.

Control and Instruction buffer

To develop highly flexible DNN accelerators it is important to make hardware adaptable to software changes. To achieve this, we have defined instructions that can be used to map the workloads to the underlying hardware. The control logic is responsible for the in-order fetching and decoding of the instructions stored in the instruction buffer, and issuing them. We defined five 128-bit instructions, as shown in Table 3.4, to execute different workloads described in Table 3.2.

Figure 3.10 shows an example of the Load/Store instruction for DMA. The source address will be either the global buffer or the output buffers based on whether is a LD/ST instruction. In a similar fashion destination address can be LIMC, activation buffers, intermediate buffers or global buffer. The LIMC and the buffers are memory mapped and the address can be used in the instruction to access it. To execute a matrix multiplication on the hardware the sequence of instructions will be LD LIMC -> LD

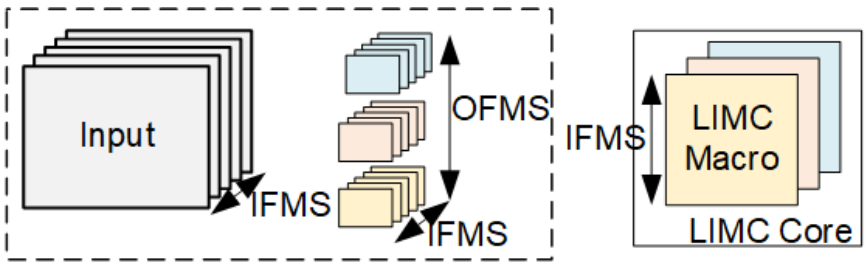

Figure 3.11: Example of mapping convolution into the LIMC core. The IFMs are mapped across the rows within a LIMC macro and the OFMs are mapped across the macros.

AB -> Matmul -> ST. To exploit the ping-pong feature the sequence of instructions will be Load LIMC -> Load AB -> Matmul -> Load LIMC -> Matmul. The second LD LIMC instruction will be executed parallelly along with the Matmul instruction, hiding the data movement latency.

3.3.3 Mapping of Workloads to LIMC

Matmul Operations

To multiply two matrices A and B, matrix B will be mapped to the LIMC core crossbar. At a time, we can compute 16 columns of the output matrix. With 450 (30 banks and 15 rows per bank) entries in each macro, a maximum of 450 rows of matrix B can be mapped to the LIMC. If bigger, then it is computed iteratively. Matrix A stored in the global buffer, is fed to the input banks of LIMC and computation happens tile by tile. If matrix A is larger than the activation buffer, a tile of data will be moved into the activation buffer. We support a tile size of 1024x450. The corresponding tile of matrix B that is required to do the computation is written to the LIMC. Feature transformation in GCN is a matrix multiplication operation. The matmul operations can be setup with instructions, and the user has full control over how to partition the data across global and activation buffer. Based on the partitions, the address can encoded into the instructions.

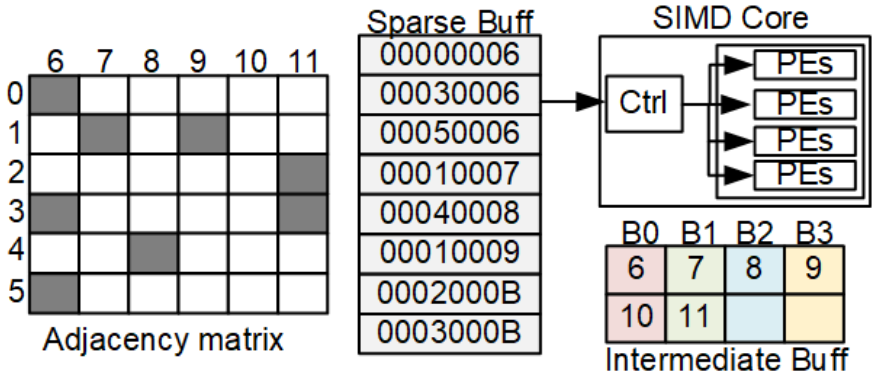

Figure 3.12: Mapping of a tile adjacency input matrix to the compute units for aggregation. Sparse buff stores the edges in COO format. The control fetches and assigns each edge to a PE row. Banked Intermediate buffer stores the features of the node to be accessed parallelly by the SIMD cores.

Convolution

Mapping convolutions to the LIMC requires high crossbar mapping efficiency to achieve optimal performance. The weight's IFMs are mapped across the rows within a macro, and the weight's OFMS are mapped across the columns of the LIMC as shown in Figure 3.11. The activations are provided at the input banks of the LIMC for computation. To achieve maximum weight mapping efficiency in LIMC, the number of rows in a bank is set to 15, a multiple of 3 and 5. A single macro can support 50 and 30 IFMs for 3×3 and 5×5 kernels, respectively at 8-bit precision. Our design is specifically tuned to achieve maximum mapping efficiency for the kernel sizes of 3×3 and 5×5. We can support kernel sizes up to 15, with the mapping efficiency less than 100%. Initial layers of CNN with fewer IFMs will also result in lower mapping efficiency but as the layer gets deeper we achieve 100%efficiency. If a layer has more IFMs then it needs to be split into multiple iterations. When there isn't data to fill up all the memory segments the segments of memory can be turned off hence saving power

3.3.4 Mapping of Aggregation to SIMD

Aggregation can happen in two ways: either FT followed by FA where the output of LIMC will be directly consumed by the SIMD core from the intermediate buffer

(IB), or standalone FA where the DMA will populate the IB with node features. Since transformation followed by aggregation results in less number of computations [35] we only support that dataflow. The aggregation of each node is statically assigned to a unique PE row, and the partial results of that node are stored locally in the scratch pad to avoid inter-core communication. The target node id modulo the number of SIMD cores 16, gives a unique PE row for that node. For a tile size of 1024 each core will get 256 nodes each. If the number of edges in a tile for a dataset exceeds the maximum entry of edge buffer, then it needs to be split into multiple iterations. The edges fed into the sparse buffer are arranged in ascending order of neighboring nodes as shown in Figure 3.12. This introduces both temporal and spatial locality within the aggregation flow and thus helps in generating predictable memory accesses. To support this pattern of memory access, neighboring nodes are arranged such that they fall into consecutive banks of an intermediate buffer, enabling parallel read access. Offline preprocessing of graph data to further improve the regularity in memory access and data reuse in the aggregation phase is well studied in [35] and is beyond the scope of this work. These off-line pre-processing methods can be used in conjunction with our hardware to further improve GCN acceleration.

3.4 Results and Discussion

To benchmark the performance of our programmable heterogeneous architecture, we implemented the entire design in RTL, synthesized, and completed the tapeout process using Intel's 16nm for 200 MHz. The SOC arch is shown in Figure 3.13 and the post-layout is shown in Figure 3.14. Full SOC level simulations of CNN and GCN workloads were performed. The SOC system consists of two parts at the top level, hl_sandbox_top and hl_chassis_top. The sandbox_top consists of the RISCV core, the accelerator engine, and the shared global memory SMEM and is explained in depth in Section 3.3. The hl_chassis_top has the testing interface logic. The global buffer is written using the JTAG and AHB bus. Innovation Port(InnPort) is an FPGA Mezzanine Card (FMC) connection, that can be used to connect to a FPGA and to it's DRAM for extended memory storage. CCU and Cfg are the clock management unit and the set of configuration registers that can be used to control the clock generation frequency.

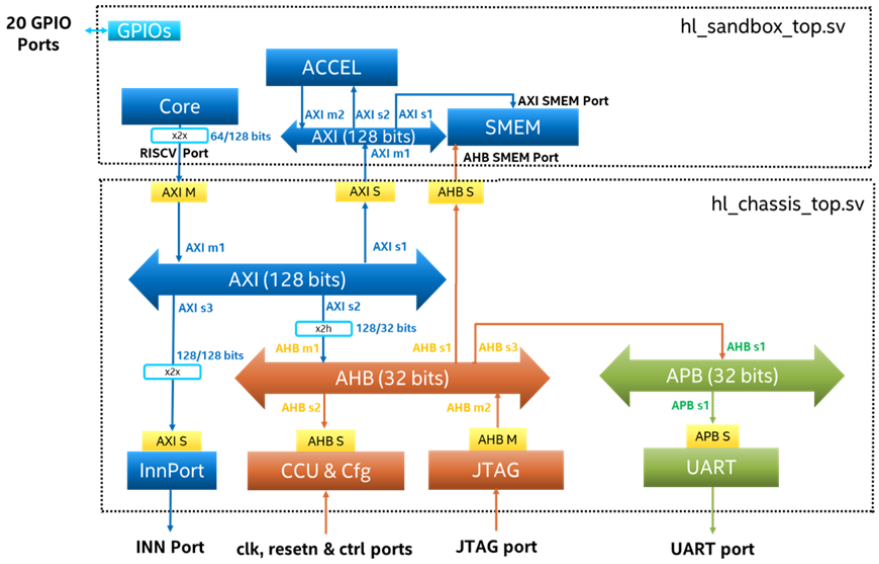

Figure 3.13: Overview of the system level SOC architecture

Instructions and data are populated in the global buffer. Once the reset is released using the configuration registers,RISCV core will fetch the instructions from SMEM and execute it. Once it encounters an AI workload instruction like for example convolution, it will write into the configuration register of the control logic inside the accelerator the start address of the instruction in SMEM. The control logic uses the DMA engine to fetch the instructions from the global memory and will execute them. For baseline comparison, SIMD cores in the heterogeneous design were substituted with LIMC cores

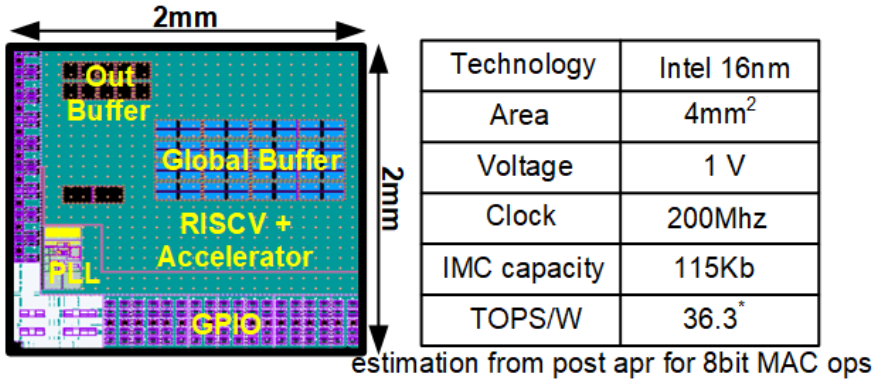

Figure 3.14: Post layout of the proposed accelerator.

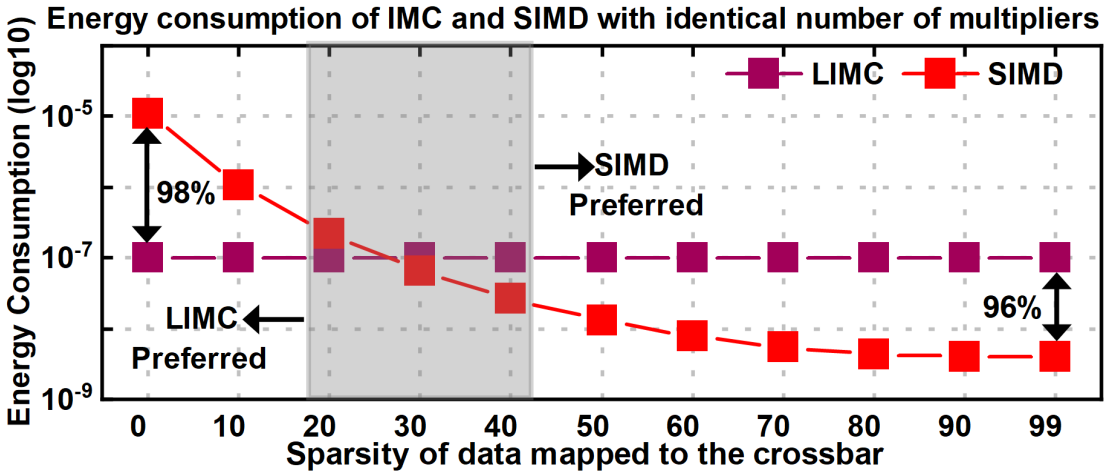

Figure 3.15: At lower level of sparsity LIMC achieves better energy efficiency whereas at higher sparsity SIMD achieves better energy efficiency for identical workload.

that have identical compute resources to create a homogeneous accelerator.

3.4.1 Workload Mapping: LIMC vs. SIMD

In this experiment, we make the LIMC and SIMD units identical, with the same number of multipliers, and perform a matrix multiplication operation. Identical inputs are fed to both LIMC and SIMD. The sparsity of the input matrix mapped to the LIMC crossbar is varied from 0 to 99%, while the sparsity of the other input matrix remains fixed at 0%. At sparsity levels below 20%, LIMC achieves up to 98% better energy efficiency compared to SIMD. Between 20% and 40% sparsity, both LIMC and SIMD achieve similar energy efficiency. For sparsity levels above 40%, SIMD achieves up to 96% improvement in energy efficiency. Regardless of the sparsity of the input mapped to the LIMC crossbar, LIMC always takes the same number of clock cycles to perform computations. In contrast, with SIMD, using compressed data representations allows us to operate solely on non-zero elements, thereby achieving lower latency. This results in lower energy consumption. Therefore, based on the sparsity of the workload being mapped to the LIMC crossbar, we can determine to which compute unit it should be mapped, as illustrated in Figure 3.15.

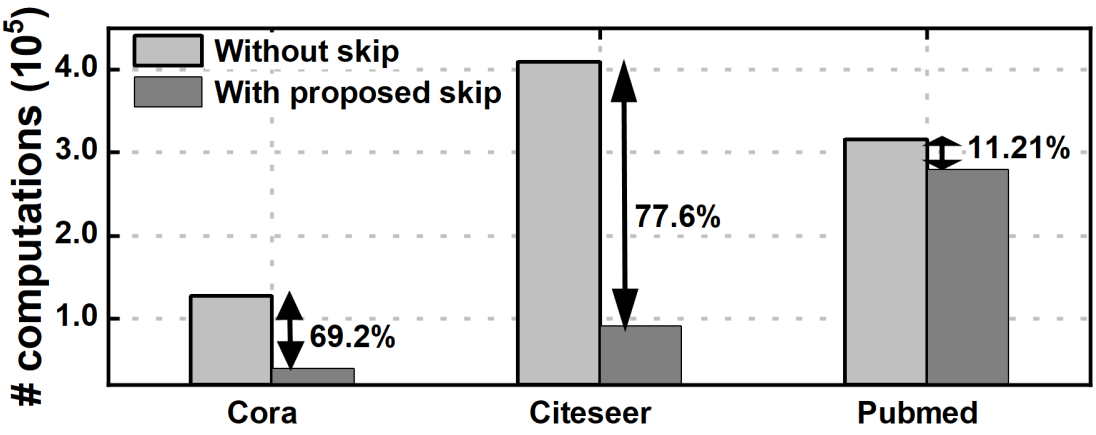

Figure 3.16: Reduction in the number of computations by exploiting sparsity at the input banks of LIMC by using operation skipping and booth multiplier.

3.4.2 Improvement of LIMC from Input Skipping and Booth Multiplier

Figure 3.16 shows the reduction in the number of computations achieved by exploiting the sparsity at the input banks of LIMC using booth multiplier and operation skipping. We evaluated the first layer of feature transformation of GCN on the graph datasets Cora, Citeseer, and Pubmed. The dense weights with zero sparsity are mapped to the LIMC core and graph feature matrix is the input to the LIMC. We achieved up to a 77% reduction in the number of computations because the initial feature matrix of the graph is sparse. This holds true for CNN as well, where the activation of later layers contain more zeros due to ReLU.

3.4.3 Comparison with LIMC-only Accelerators

To benchmark the design, we compared our proposed heterogeneous accelerator against the baseline homogeneous accelerator, which consists solely of LIMC cores, using GCN (Cora) and CNN (MNIST) as the workloads. In the case of GCN using the homogeneous accelerator, both FT and FA are mapped to LIMC. In contrast, with the heterogeneous accelerator, dense FT is mapped to LIMC, and sparse FA is mapped to SIMD. For CNN, since convolution weights are dense, they are mapped only to LIMC for both the heterogeneous and homogeneous accelerators. The number of cycles required for GCN

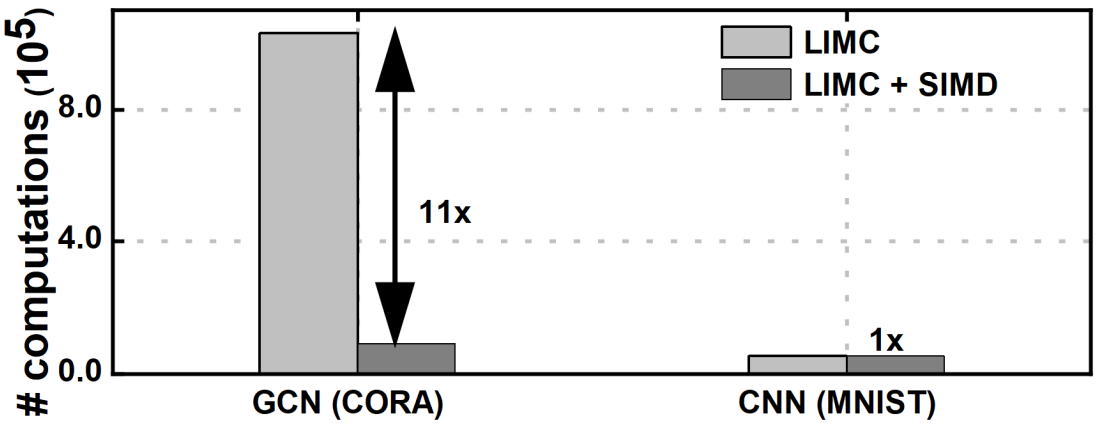

Figure 3.17: End-to-end GCN and CNN inference performance comparison between the LIMC-only homogeneous accelerator and heterogeneous architecture.

with a homogeneous accelerator is 11x more than with the proposed heterogeneous architecture, as illustrated in Figure 3.17. The poor operational intensity of LIMC for the sparse FA of GCN contributes to the overall latency of GCN inference when using a homogeneous accelerator. For convolution, both the heterogeneous and homogeneous architectures have the same performance since convolutions are only mapped to LIMCs.

3.4.4 Energy Inefficiency of LIMC with Sparsity

In Figure 3.18, the energy consumption of LIMC and SIMD cores for sparse GCN FA and SpMV workload is depicted. For sparse workloads, SIMD achieved up to a 98% improvement in energy efficiency as compared to LIMC cores.

3.4.5 Comparison with Prior Works

Table **??** shows the comparison of our accelerator with the state-of-the-art IMC based accelerators. The neural networks (NN) used to evaluate the performance are the same as in [40]. [36] is primarily an IMC based accelerator which can support CNNs. This works achieves better performance on CIFAR10, which can be attributed to larger scale design with 400x more IMC compute capacity. Since they use IMC for compute, it will suffer from low operation intensity and energy efficiency for sparse workloads. They have also utilized a SIMD core primarily for element-wise activation function operations.

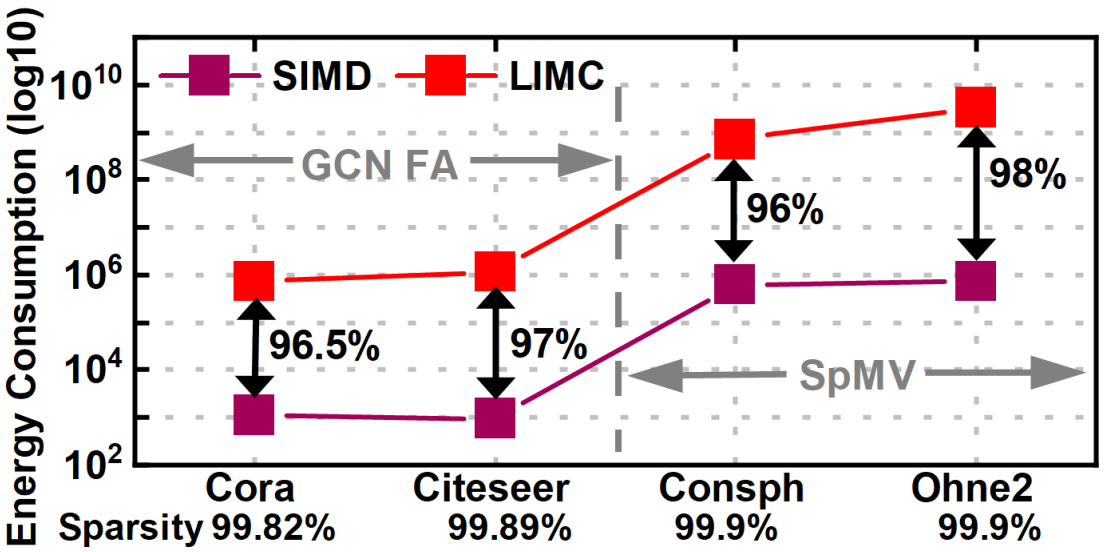

Figure 3.18: LIMC vs SIMD energy consumption for GCN FA and SpMV. Proposed SIMD achieves high energy efficiency for sparse GCN FA and SpMV workload.

[31], also uses ping-pong for weight updates by using alternate banks as ping and pong with 16 rows in each bank with a total of 128 rows. But here the size of the ping and pong array is extremely small with 64 rows each thus leading to negligible update overhead when compared to execution time. Thus for small array sizes the improvement from ping-pong is not significant. We achieved 5x and 2.3x performance improvement over [40] with frequency scaled down to 100MHz. [40] is a Stochastic computing (SC) based IMC and SC suffers from high cost of binary to stochastic number conversion and compute error.

Our architecture achieves 9x better performance than [39] with the operation frequency of our design scaled down to 10 MHz. [39] is an all-IMC design, making it inefficient for sparse workloads. Despite having 67x more resources and 1GHz frequency, our accelerator achieves comparable performance with [30] for a two-layer end to end GCN inference latency. Full IMC based GCN accelerators like [30] are extremely inefficient in executing sparse FA aggregations, due to low operational intensity and wasted memory bandwidth from moving zeros. IMCs-based accelerators can also employ sparse representations while transferring data from memory to conserve bandwidth, albeit with the additional hardware overhead of decoding and accurately mapping it to the crossbars. However the operational intensity still remains low. [35], a FPGA implementation

achieves better performance since it deploys symmetry based offline pre-processing to improve locality in graph. It also uses HBM with 512GB/s bandwidth which reduces the data movement latency. We achieve a 2.1x improvement in TOPS/mm^2 over [36] at 16nm and upto 20x improvement in TOPS/W over others.

3.5 Conclusion

IMCs are inefficient in accelerating extremely sparse workloads, due to their low operation intensity and energy efficiency. In this work, we propose a scalable heterogeneous accelerator with LIMC and SIMD cores, to balance the computation needs and the efficiency of CNN, GCNs and SpMVs. We design an architecture which efficiently integrates a ping-pong based LIMCs for MAC operations and the SIMD cores for sparse operations. We also define a workload mapping strategy to determine which core to utilize based on the sparsity of the workloads. We implement our design with Intel 16nm PDK at 200 MHz and successfully taped out a complete SOC accelerator based on the new design. For the CNN, GCN and SpMV benchmark workloads we achieve up to 11× improvement in performance over homogeneous accelerators and 2.1× improvement in TOPS/mm^2 over [36] at 16nm and 20× improvement in TOPS/W, as compared to state-of-the-art accelerators. Since AI models can be expressed as a combination for dense and sparse computations, the design strategy and analysis in this work can be extended to accelerate more diverse AI workloads.

Chapter 4

HW-SW Co-Design of A 28nm 2-Tier ViT Enabled by Advanced 3D packaging

4.1 Introduction

Deep learning has evolved into different application domains such as computer vision, natural language processing, health care etc. Although convolutional nueral networks(CNNs) were quite popular for a long period of time, recent superior performance achieved by transformer-based AI models has caused a shift towards it.

The attention-based transformer is an attractive strategy for providing additional relationship contexts during the modeling process. Compared to static convolutional filters that have limited receptive fields and are fixed for all contexts once trained, self-attention allows the dynamic computation of a new set of kernels to extract the relational weightings among global input positions [41]. Therefore, computer vision applications have started using transformer based models. Computer vision applications such as autonomous machines, AR/VR, relies heavily on cloud-based computing solutions, leading to high latencies and privacy concerns. Thus, it is imperative to move computation closer to the source i.e. the image sensors. Conventional edge solutions for computer vision tasks use MIPI CSI-2 interconnects [42] to transfer data from the image sensor, where the

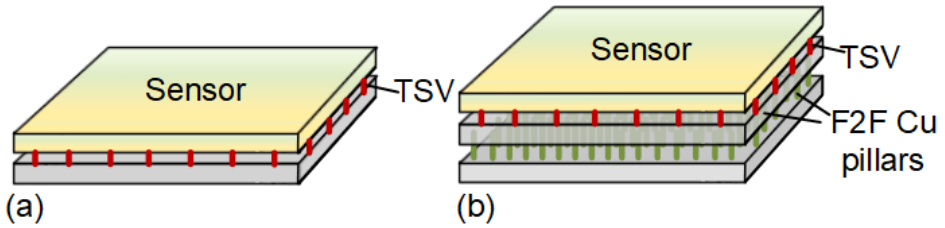

Figure 4.1: (a) Existing In Sensor Compute solution with a sensor layer stacked on top of compute die using TSVs (b) Proposed 3D stacked solution using face-to-face bonding with two logic die to support 2-layer ViT for small object detection.

downstream computation occurs. However, as image sensor resolution increases, MIPI CSI-2 becomes a bottleneck for data bandwidth, latency, and energy consumption. Fig. 4.2 shows the data rate generated under various sampling frequencies at different sensor resolutions. The bandwidth requirement is more than what MIPI CSI-2 can support. At higher sensor resolutions, the MIPI transmission latency will exceed the end-to-end latency requirement for many applications [43]. In addition to latency, power consumption is also a critical factor. MIPI CSI-2 needs 100 pJ to transmit one byte of data [44], consuming 10-60% of the total power budget of VR devices [43, 45].

Although vision transformers can often surpass CNNs with similar numbers of model parameters and operations, their higher computational complexity hinders their usage in hardware inference [41]. With Moore's law slowing, the guaranteed increase in transistor count and thus compute performance slowing down is also slowing down. The die yield is decreasing, and the cost of monolithic design is increasing. Thus, for the next generation of AI accelerators, it is important to look beyond the traditional 2D design space to achieve a scalable design and pack more performance with the same area cost. To achieve this, in this work, we utilize advanced packaging technology, specifically 3D die stacking with high-density copper (Cu) pillars, to develop a 2-tier hardware-software co-design for an AI accelerator. This accelerator can be stacked with sensors using Through-Silicon-Vias (TSVs) along the edges to achieve 3D stacked in-sensor computing for real-time data processing, as shown in Fig. 4.1(b).

We use face-to-face bonding (F2F) with Cu pillars with 5µm pitch to achieve a true 3D stacked design, thus achieving more compute density than 2D/2.5D packaging of similar footprint and higher connection density than TSVs or microbump based stacking.

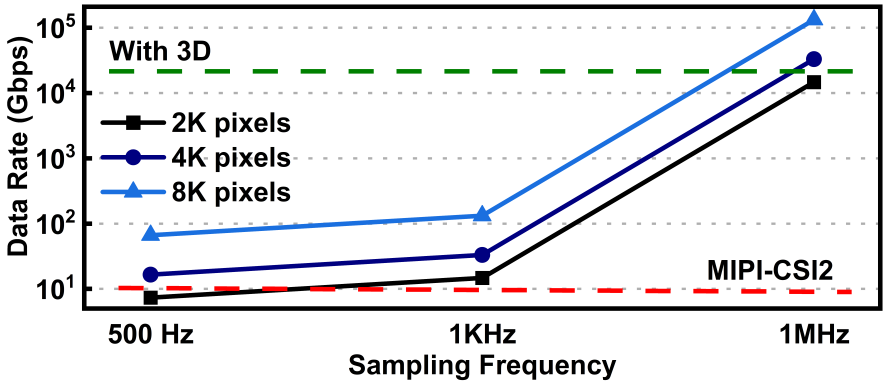

Figure 4.2: MIPI-CSI2 cannot support the high bandwidth requirement of sensors. Proposed 3D stacked design with face-to-face bonding with high density interconnects provides sufficient bandwdith to support the growing sensor resolutions.

As shown in Fig. 4.2, the high connection density provides sufficient bandwidth to support higher sensor resolutions and sampling frequencies. Differentiating from the existing 3D architectures' [46,47] memory-on-logic design partitioning strategy, which is suitable for memory-bound applications, we explore logic-on-logic die stacking to support a two-layer vision transformer (ViT) to perform small object detection. ViT segments images into fixed-sized patches for parallel processing and efficient data movement. It can leverage the high-density bandwidth of Cu pillars for improved data transfer across the compute dies. The ViT is pre-trained for small object detection and quantized to fit within the two-die structure. The accelerator's dataflow is co-optimized with the integration technology to achieve maximum efficiency. We also address different critical physical implementation issues in 3D die stacking, such as ESD requirements for F2F bonding, clock synchronization, and thermal management. The major contributions of this work are:

- We propose the use of advanced F2F bond 3D packaging technology with 5μm pitch Cu pillar to develop a 2-tier AI accelerator for real time sensor data processing.
- We develop a two layer ViT model for real-time small object detection.
- We utilize logic-on-logic die stacking and efficient algorithm partitioning to maximize the benefits of 3D stacking.
- We address physical implementation issues in 3D die stacking such as ESD, signal

synchronization and thermal issues.

- We implement the design in 28nm technology and achieve 18x improvement in latency and a 127x reduction in energy consumption compared to conventional 2D designs and an 11x improvement in latency compared to similar 3D architecture.

4.2 Background and Motivation

In order to handle large volumes of data, it is crucial to increase processing power without increasing area costs or power consumption. Expanding processing capabilities in two-dimensional designs typically leads to greater area and power demands. Thus, 3D design serves as an effective solution by enhancing computing density while maintaining or reducing the same area and power consumption. In 2D design, each die is laid out on a PCB and is connected using conductive wire paths. With 3D staking these longer wire paths can be replaced with shorter interconnects, thus allowing for a faster data transfer with lower energy consumption. Bonding stacked dies into a single package instead of a multiple package on a PCB increases I/O density by 100x. The energy-per-bit transfer can be reduced to 30x with the latest technology [48]. The compact footprint is suitable for mobile devices and other applications where area cost is critical. With 3D stacking, each die, acting as a chiplet, can be easily scaled up or down or replaced with another chiplet. This capacity and flexibility are ideal for compute-intensive applications such as high-performance computing (HPC), data centers, cloud computing, artificial intelligence (AI), and machine learning (ML).

Existing methodologies for 3D integration, including micro-bumping, hybrid bonding, through-silicon-vias (TSVs) and monolithic 3D (M3D) [49, 50], all enable advanced packaging and heterogeneous integration. In microbump 3D IC, two dies are vertically stacked with a dense array of microbumps, providing high yield and reliability. The heterogeneous 3D IC micro-bonding die stacking provides great flexibility in technology selection and IP configurations [47]. The relatively large geometrical dimensions of today's TSVs and their low manufacturing yield limit TSV-based 3D stacking to designs with a small number of inter-die connections. Hybrid bonding technology uses F2F bond pads to stack two predesigned 2D wafers through the back-end-of-line (BEOL) layers [51]. Since the F2F bond pads are smaller than TSVs, the 3D IC hybrid bonding

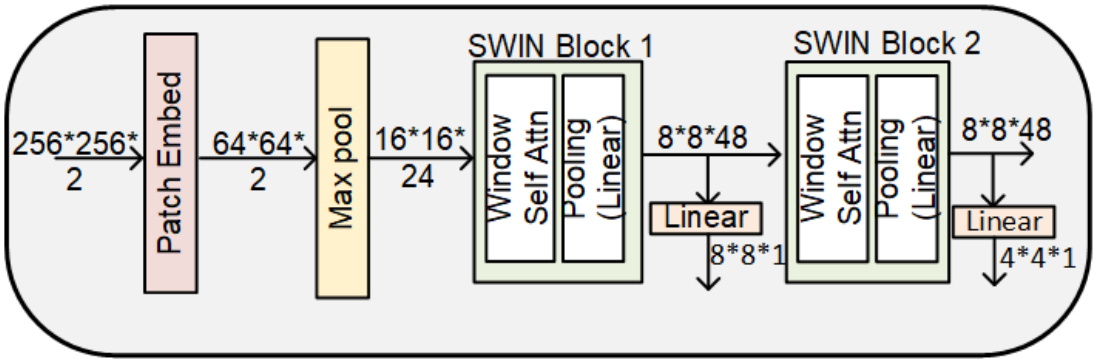

Figure 4.3: Proposed 2 layer ViT for small object detection.

provides high-density vertical integration [47], achieving 18.2Tb/s/mm^2 bandwidth density [52]. Hydrid bonding will also allow for heterogenous integration, where different dies can be at different technology nodes.

M3D is an emerging technology that integrates devices layers sequentially in a vertical direction [53]. It uses monolothic inter-tier vias offering the best finest grained integration. M3D suffers from low yield and high fabrication cost. Micro-bumping requires an additional I/O driver design for inter-die connection and also the microbump size is quite large, 25µm as compared to 1µm and and 5µm of M3D and hybrid bonding [47]. Compared to microbumping, hybrid bonding allows for PHY-less, simple digital logic driven IO circuitry [52]. F2F bond pads in hybrid bonding can be easily integrated using existing technologies as vias during the physical design stage, achieving much lower integration costs and performance that is very close to monolithic designs, with almost no power and signal penalties [54]. Hybrid bonding has better timing performance and lower switching power as well. Thus, in this work, we use hybrid bonding for its lower cost of design, high integration density, and better power, performance, and area (PPA).

4.3 2-layer ViT Algorithm Design

In this work, we chose window-based transformers [55–57], which offer linear computational complexity relative to the number of tokens, as opposed to traditional vision transformers that exhibit quadratic complexity. The model enhances performance across varying object scales in images, by using a pyramid model architecture that generates multi-scale image features. To reduce the complexity of the model architecture and

make it suitable for hardware friendly small object detection, we developed a two-layer model architecture for small object detection, as shown in Fig. 4.3. The input image goes through an initial convolution-based embedding stage with kernel size and stride 4, followed by a 4x4 maxpool. This is followed by two SWIN blocks, which extract the multi-scale image features. Each SWIN block consists of 4 window self-attention blocks [56] followed by a linear pooling layer. The output feature map of each SWIN block passes through a linear projection block to generate the binary selection maps, used for small object detection by selecting positive patches. We trained the model using the Prophesee dataset [58], following the same training methodology as [56, 57].

We further simplified the model architecture by reducing the number of layers and feature channels [59]. Additionally, we applied model weight quantization, maintaining the same performance while significantly reducing resource demands. Table 4.1 presents the experimental results across different model configurations on the Prophesee dataset [58], which, as a DVS dataset, does not require large-scale models. We evaluated performance using TPR (recall), which reflects selection accuracy, and the computation efficiency using MACs and the number of parameters. The table shows that reducing the number of Swin Blocks from 3 to 2 results in a slight drop in TPR due to the loss of one feature scale, but significantly reduces computational cost, especially in terms of the number of parameters, making the model more feasible for hardware implementation. Reducing from 2 to 1 block leads to a substantial accuracy drop without significant computational gains, so we opted to use 2 blocks. We then analyzed the impact of feature channels, 48 channels have a negligible improvement in TPR, but a huge decrease in computation efficiency. Another hardware bottleneck in ViT is the Softmax used for attention calculation. We replaced Softmax with ReLU without any drop in TPR. To minimize parameters in the embedding layer, we introduced a max-pooling layer after it, allowing the use of a smaller kernel while preserving the output shape. Finally, we quantized the model weights from FP32 to INT4 further reduced computational demands while maintaining performance.

To enhance performance across varying object scales in images, we introduced a pyramid model architecture that generates multi-scale image features.

Specifically, the input image $\boldsymbol{I}_{in} \in \mathbb{R}^{Hin \times W_{in} \times C_{in}}$ (where H_{in}, W_{in}, and C_{in} represent the image height, width, and the number of channels, respectively) is processed through

Table 4.1: 2 layer ViT Model optimization results

#Blocks / # Channels / Window size	TPR	# MACs	# Params
3/ 24 / 7	0.91	5.37M	5.37M
2 / 24 / 7	0.85	4.89M	1.90M
1 / 24 / 7	0.75	4.83M	1.28M
2 / 48 / 7	0.86	10.10M	3.52M
2 / 24 / 7 with ReLU	0.85	4.89M	1.90M
2 / 24 / 7 4bit	**0.82**	**4.89M**	**1.90M**
2 / 12 / 7	0.78	2.41M	1.27M

a hierarchical patch encoder composed of multiple Swin Blocks after the embedding layer. This results in patch representations at three different scales: $\boldsymbol{r}_1 \in \mathbb{R}^{\frac{H}{2}\times\frac{W}{2}\times 2C}$, $\boldsymbol{r}_2 \in \mathbb{R}^{\frac{H}{4}\times\frac{W}{4}\times 4C}$, and $\boldsymbol{r}_3 \in \mathbb{R}^{\frac{H}{8}\times\frac{W}{8}\times 8C}$. Each feature map passes through a projection block to generate the final selection maps, used for small object detection by selecting positive patches. To supervise training effectively, we apply a pyramid-structure labels (such as segmentation labels or bounding boxes) are designed by assigning object-containing patches to be positive and rescaled to align with the corresponding feature map dimensions. The loss is computed across all scales and summed for backpropagation. During inference, we first interpolate the output feature maps to a uniform size, then select the maximum value from each patch across the scales—if any scale predicts a positive selection, the patch is chosen.

4.4 3D Architecture

4.4.1 Architecture Overview

Figs. 4.4 (a) and (b) show the top and bottom die architectures of the proposed two-layer 3D stack accelerator. The accelerator is made up of individual compute cores, each targeting a particular computation of the algorithm model. The primary compute unit in the accelerator is an output stationary systolic array (SA). SA uses a similar data flow for convolution and matrix multiplication as described in [35, 60, 61]. The dimensions of the systolic array of each compute core are carefully chosen to maximize parallelism and reduce latency. The patch embed core (PEembed) performs the initial

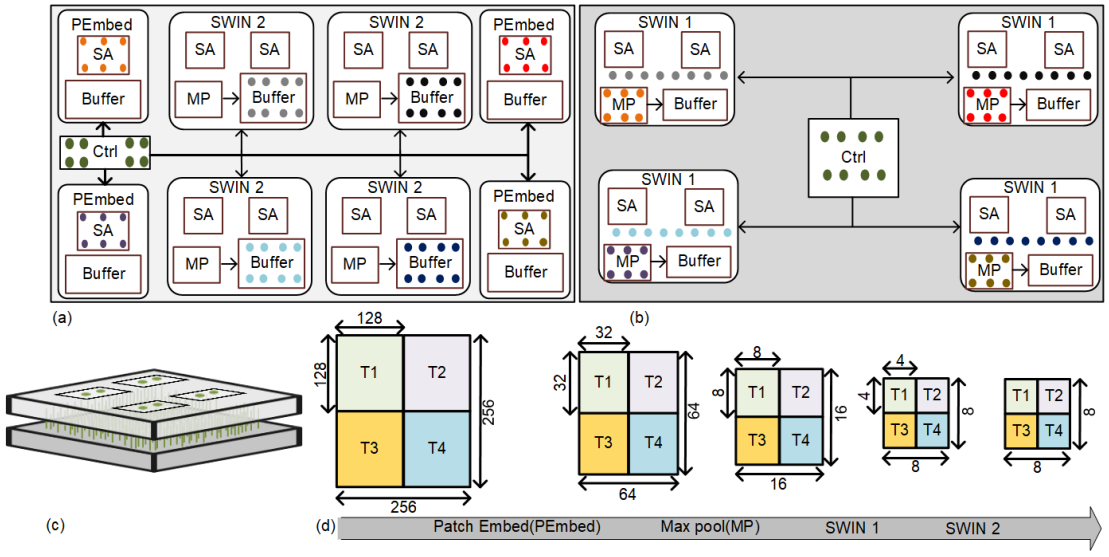

Figure 4.4: (a) Top Die architecture with the F2F connections highlighted using dots (b) Bottom die architecture and the corresponding F2F bonds from top die highlighted using same color (c) 3D view of the architecture with the density hybrid bond shown in green(d) Dataflow of the proposed architecture highlighting localized parallel computation for one input channel. We partition the input into 4 tiles, ensuring localized intra-tile data transfer with no inter-tile communication to achieve maximum parallel computation and exploit the high bandwidth provided by the 3D system.

patch embedding convolution on the input image. SWIN 1 and SWIN 2 cores perform window self-attention of the transformer model using SA and have the maxpool engine (MP) as well. The top die control logic coordinates the entire data flow across the 2 stacks. The bottom die has a secondary control logic to receive from and send back the necessary control signals to the top die. The top die receives the input image pixels and stores them in the buffers in the PEmbed core. Each compute core has it's own buffers to store the corresponding weights and input activations. Compared to 2D accelerator design, logic-on-logic-stacked 3D accelerator provides higher compute density for the same die area. But to fully exploit the system level performance improvement from 3D stacking, the system should be co-optimized with the integration technology used.

- Carefully partition and map the algorithm to hardware so that the high bandwidth

provided by 3D stacking is exploited and high hardware utilization across die stacks is achieved.

- Address physical design challenges like IO bump circuitry, thermal and inter-die data synchronizations.

Our main contribution is how to maximize the system-level performance of 3D stacked architectures by joint co-optimization of algorithm, hardware, and the integration technology. We discuss algorithm partition and mapping, hardware partition and dataflow with the aim of maximizing the benefits of 3D stacking. We leave further optimization of the multihead attention computation and convolutions within the SWIN and PEmbed cores for future work.

4.4.2 System Co-optimization

Hybrid bonding with the Cu pillar provides extremely fine-pitch copper-to-copper interconnection between the dies. This enables high bandwidth and low latency connections between the 3D stack dies, leading to an increase in performance and reduced power consumption [52]. But to achieve the maximum performance of 3D stacked architectures, careful co-design of the system based on the integration technology is required. Certain applications will see a significant gain with 3D stacking, while others may see only a small gain [52]. In this study, we selected ViT because of its ability to compute in parallel and the potential to utilize the high-density bandwidth of Cu pillars for supporting high data transfer rates between compute dies. To translate this inherent parallel computation and data transfer nature of the ViT algorithm into hardware gains, a dataflow was carefully designed at the system level by splitting the input image data into 4 tiles, as shown in Fig. 4.4(d).

Each phase of the computation can be processed concurrently across different tiles. Additionally, the dataflow ensures that there is no inter-tile communication within each computation phase and between different computation phases. For example, at PEmbed, all 4 tile computations happen in parallel, and as we move from PEmbed to Maxpool (MP), the output of PEmbed T1 is the input of MP T1 so on and so forth, as shown in Fig. 4.4(d). Given that each algorithm phase comprises 4 tiles, the hardware level similarly includes 4 compute cores, 4 PEmbed cores, 4 SWIN cores, and equivalents

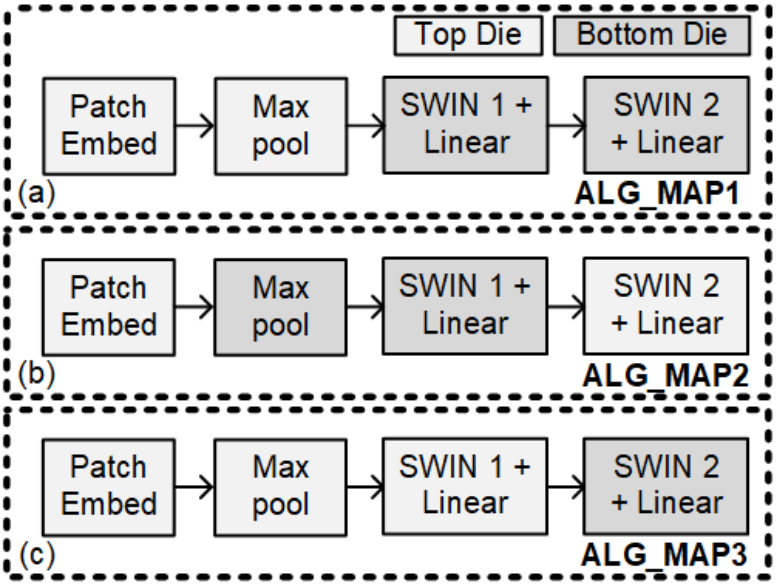

Figure 4.5: Shows three different algorithm partition and mapping strategies. The die layers are represented by different colors.

for each phase to perform their designated computations. Parallel computation with no inter-tile communication helps maximize performance, and also enables sending data parallelly across the compute die without any local data rerouting within tiles at the receiving die. This is highlighted in Fig. 4.4(a) and (b) using color-coordinated dots between a pair of compute cores across the die stack. This simplifies the logic for sending and receiving data across dies, maximizing the benefits provided by 3D interconnection.

4.4.3 Algorithm Partition and Mapping

The partitioning of the algorithm and its mapping to the die stack is important to maximize performance. We consider three mapping strategies, ALG_MAP1, ALG_MAP2 and ALG_MAP3, as shown in Fig. 4.5. Fig. 4.5(a), the patch embedding and maxpool are mapped to the top die, while the SWIN blocks are mapped to the bottom die. In Fig. 4.5(b), the patch embedding and the last layer SWIN block are placed on the top die, and the maxpool along with the first SWIN block are located on the bottom die. Finally, in the last mapping Fig. 4.5(c), the first three layers are mapped to the top die, and the last SWIN layer to the bottom die.

We evaluated the three mapping strategies using three metrics: (1) compute core

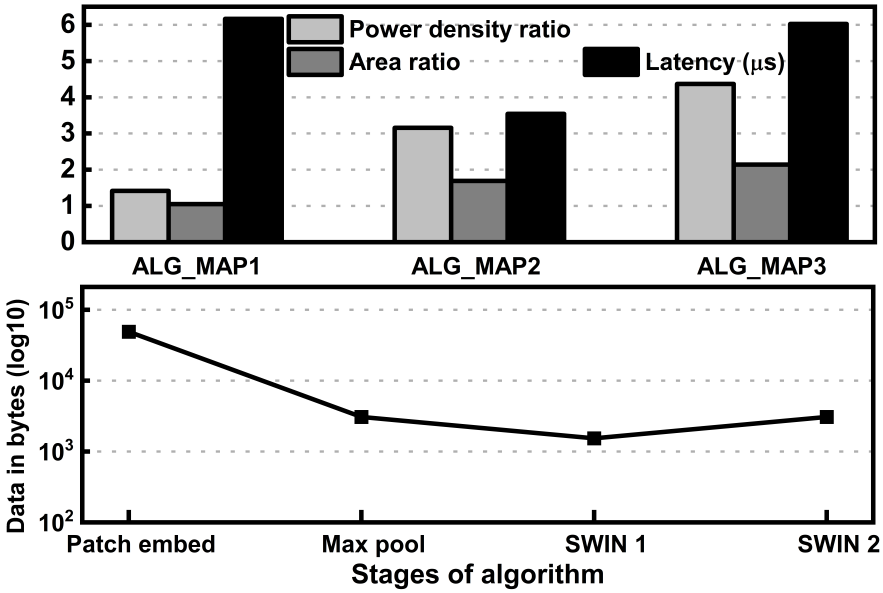

Figure 4.6: Top figure shows the evaluation metrics for different algorithm partition and hardware mapping strategy. Bottom figure shows the volume of generated at the output of each stage of algorithm. Our mapping strategy ALG_MAP2 exploits the high connection density of 3D packaging to minimize high volume data movement latency across dies, thus minimizing the end to end latency with moderately balanced area and power density ratio of top to bottom die.

area ratio of the top to bottom die, (2) power density(W/mm^2) ratio of the top to bottom die, and (3) end-to-end compute latency of one input image, as shown in Fig. 4.6. The results are collected from end-to-end RTL implementation, synthesized in 28nm technology running at 800MHz. ALG_MAP1 provides a balanced ratio of area and power distribution, but suffers from the highest latency. This is due to the mapping strategy not taking advantage of the high-density interconnects of 3D stacking. In a systolic array, the output is taken from the bottom edge of the array sequentially [60]. As illustrated in Fig. 4.5 bottom, the data volume peaks at the output of PEmbed, causing significant latency when transferring data sequentially from PEmbed's SA to the maxpool core. This also reduces the number of parallel operations that maxpool can execute. For the bottom die, although the data movement from SA to SWIN 1 to SWIN 2 is sequential, it does not add to the latency. This is because the SA array needs only one new data every clock cycle and thus can be fed sequentially. The balanced power

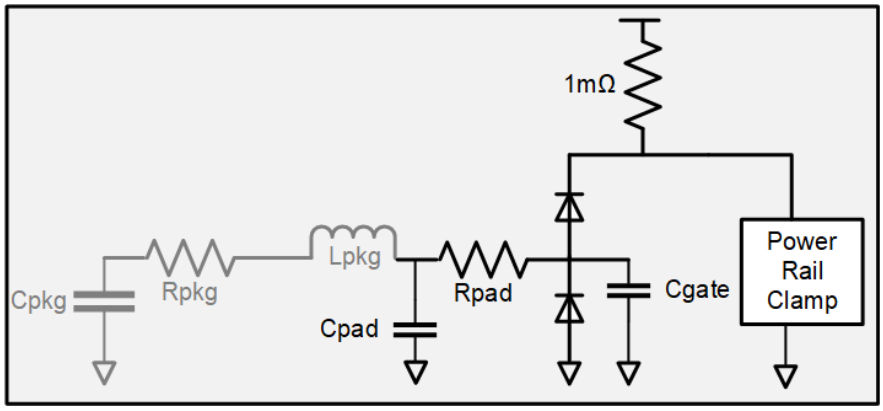

Figure 4.7: IO circuitry for spice simulation of regular IO pads and for F2F CU pillar hybrid bonding pad. Components in light gray are left out for F2F bonding.

distribution will give good thermal performance for ALG_MAP1.

ALG_MAP2 has the best end-to-end latency among all three mapping methodologies. As compared to ALG_MAP1, the maximum data volume output from PEmbed's SA can be taken out in parallel from each systolic unit and can be transferred in parallel to the bottom die using the high density interconnects, reducing data movement latency. The maxpool core can then process all this data in parallel, reducing compute latency as well. ALG_MAP2 has moderately skewed compute area and power distribution ratio. This is because the top die has two main compute cores PEmbed and SWIN 2 as compared to SWIN 1 and the significantly smaller maxpool core. ALG_MAP3 has the worst area and power density ratios, since it got PEmbed, maxpool, and SWIN 1 on the same top die. The latency is also high due to the serial data movement as in explained for ALG_MAP1. Thus, for our design, we adopt ALG_MAP2, since it provides the lowest latency with a reasonably balanced power distribution ratio.

4.4.4 Physical Design Considerations

IO Circuitry

Electrostatic Discharge (ESD) induced die failures can occur during post manufacturing processes of advanced packaging technologies. ESD protection typically involves adding large diodes in I / O cells, increasing capacitive load, potentially limiting bandwidth, and can result in up to 20% increase in I/O area [62]. For hybrid bonding, JEDEC

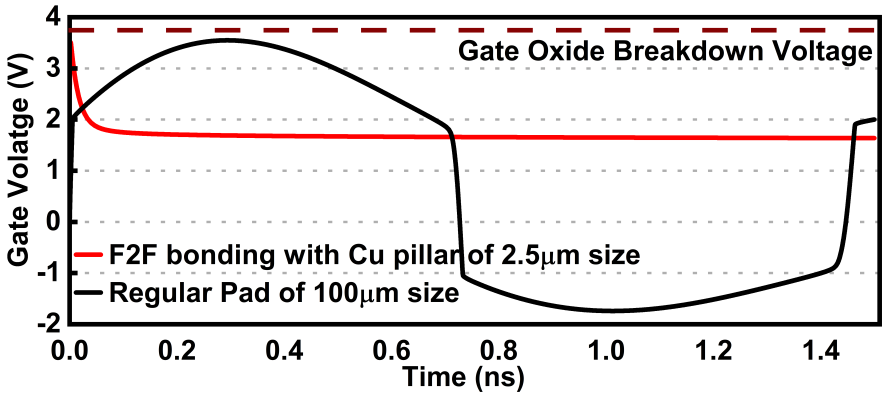

Figure 4.8: Resulting Gate Voltage from simulated ESD event. An ESD voltage of 250V was used for the regular pad, and 5V for the F2F bonding case. The minimum required diode size was chosen for these examples. Gate-oxide breakdown voltage for this process is approximately 3.8V, scaled from the values in [1]

recommends ESD protection target voltages of less than 5V as compared to 250V for traditional pads for the 28 nm process [1]. To compute the ESD diode requirements for F2F bonding, we build a simulation model, following [62, 63], shown in Figure 4.7. The package RLCs are taken from the PDK databook and include components such as the bond wire and output pad. For F2F bonding, the pad RC is estimated from parasitic extraction of the I/O cell, and the components in light gray corresponding to the external package are left out. Voltage is applied to the package and pad capacitance, for the regular and F2F bonding cases respectively, and the gate voltage is recorded, as shown in Fig. 4.8. We sweep the size of the ESD diode to find the minimum size necessary to keep V_{gate} below the gate-oxide breakdown voltage. The diode size needed for an external I/O cell with a regular pad is 600 μm^2, while for the bonding of F2F with Cu pillars it is 6.67 μm^2 per diode. Since the minimum required diode size is quite small, we can remove ESD diodes for F2F bonding thus simplifying the IO circuitry.

Thermal

Thermals are critical in 3D architecture because die stacking can dramatically increase power density if two highly active regions are stacked on top of each other [64]. Fig. 4.9 shows the combined power density distribution of the top and bottom dies of the accelerator. The power of individual compute cores was obtained from synthesis and

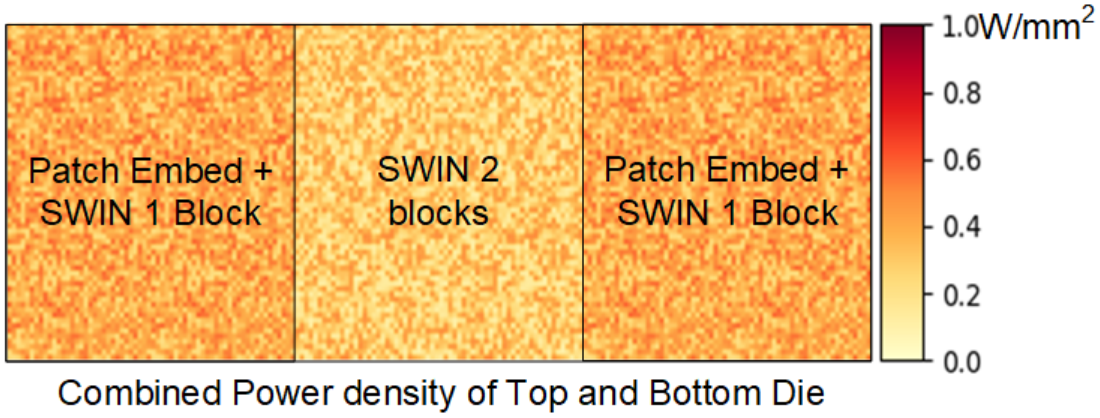

Figure 4.9: Illustrative power density distribution of the top and bottom stack combined. At the edges the power density is higher since there are two compute cores PEMbed and SWIN1 stacked on top, and the power density is lower in the middle since there are only SWIN2 cores. The power density is within the acceptable range of 1W/mm^2 for 3D designs to avoid thermal issues [2, 3].

the power distribution was calculated by superimposing the two dies as a single system. The left and right sides of the system have a higher power density since both PEmbed and SWIN 1 blocks are stacked on top of each other, whereas the middle portion with SWIN 2 blocks has a lower power density. The average power density is 0.24W/mm^2, which is much lower than the 1W/mm^2 threshold for 3D stacked designs for efficient thermal performance [2, 3].

Inter-Die data synchronization

Legacy 3D designs using large-pitch microbumps, have a large latency and low-bandwidth interface between dies, and hence are usually asynchronous [52]. In contrast, hybrid bonding with 5μm pitch and a relatively simple IO circuitry as explained in Section 4.4.4, the propagation delay will be minimum. We calculated the propagation delay of sending the clock through the Cu pillar using the same circuit and experiment explained in Section 4.4.4, but with a clock signal as input. With an 800 MHz clock at the input of the pad, the propagation delay is very low, on the order of tens of picoseconds, as shown

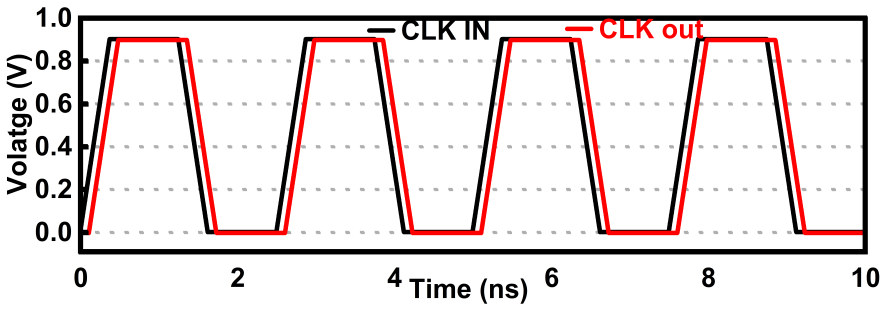

Figure 4.10: Spice simulation waveform of sending a clock signal across the die stack through Cu pillar hybrid bond. The propagation delay is very negligible, in the range of picoseconds thus resulting in negligible clock signal variations.

in Figure 4.10, resulting in very low inter-die skew, jitter, or phase difference. Thus, the 3D interface is almost synchronous and both the top and bottom dies run at the same clock frequency. Each die has its own clock tree as in a typical 3D stack design [47, 65] and the clock is forwarded from the top die to the bottom die. Since the launch and capture flip flop of the data are on the same clock but on different tiers, we use a 1-Flip Flop synchronizer at the receiving end to account for the slight phase difference that can be introduced due to the low propagation delay.

4.5 Results

Experimental Setup

The hardware accelerator architecture shown in Fig. 4.4 is implemented in RTL and is synthesized using TSMC 28nm technology for 800MHz. Three systolic array configurations were used: 32x24 in PEmbed, 64x24 in SWIN1, and 16x48 in SWIN2. The column dimension is set to either the maximum number of output channels in convolution or to the column dimension of the matrix multiplication of attention calculation. The row dimension is set to support the maximum parallel computation for a given column dimension. A total of 22MB of memory is split between the compute cores to support the input activation tile and the corresponding weights. For comparison, following the same methodology as [44], we developed a 2D accelerator with 4 PEMbed cores and 4 SWIN blocks. The PEmbed cores' configuration is kept the same as our 3D design, where as for the SWIN blocks the size of SA is set to 64x48. For the 2D system design, the compute

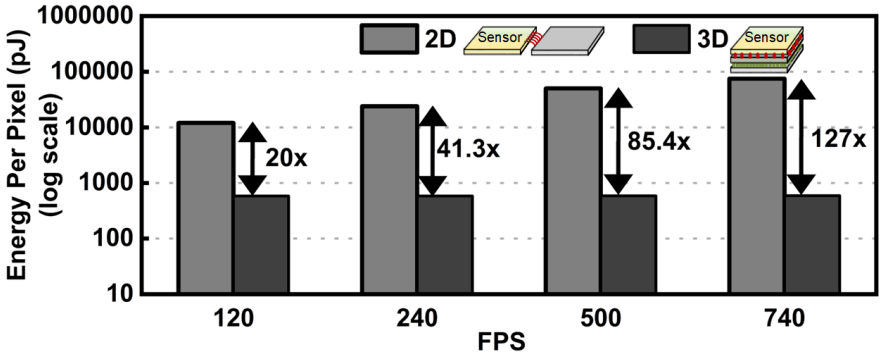

Figure 4.11: Energy per pixel (transmission + compute) for 2D and 3D accelerators at different FPS. The 2D accelerator uses MIPI-CSI2 to transfer the entire image frame, resulting in high transmission energy. The 3D stack with TSVs and F2F bond has lower transmission energy. The lower compute density of the 2D accelerator increases compute latency per frame and hence, compute energy compared to the 3D accelerator.

Table 4.2: Comparison of 2D vs 3D stacked accelerator system.

	Latency (uS)	Power (W)	Power Density (W/mm^2)	Compute Density (/mm^2)
2D	64.41	4.81	0.19	368.64
This work	35.66	6.01	0.24	491.52
Over 2D	18.06x	0x	0x	1.3x

die will receive the image input from the sensor through MIPI CSI-2 and for 3D we use the same setup as [44], TSVs to transfer data from sensor to compute stack. All results are calculated for the Prophesee dataset [58]. Place and Route is not implemented in this work. The methodology for implementing the place and route is discussed in [47].

Comparison with 2D design

Table 4.2 shows the comparison between the 2D and 3D systems. Our work achieves 18.06x improvement in end-to-end latency for the ViT model and provides 1.3x more compute density than a 2D system. As expected, the 3D system consumes more power and have a higher power density compared to the 2D system. Power density is a critical measurement in 3D design. Higher power density results in thermal hotspots, which is detrimental for 3D designs. Fig. 4.12 shows the comparison of the power density between the 2D and 3D designs at different frequencies. Since two dies are stacked together, the

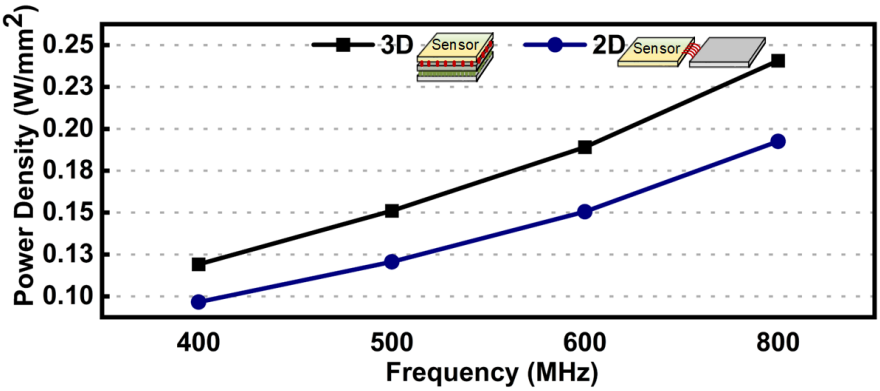

Figure 4.12: Power density of the proposed 3D stacked accelerator vs 2D accelerator of same area at different frequencies. The power density is within the acceptable range of 1W/mm^2 for 3D designs to avoid thermal issues.

3D design has 1.2x higher power density than the 2D design. However, our design has an average power density of 0.24W/mm^2, well within the limit of 1W/mm^2 required for 3D designs [2, 3]. We used [66] tool to perform a thermal analysis of our design. Our design has a uniform temperature distribution of 300K, a marginal increase of 7K with respect to room temperature (293K). Hence, there are no major hotspots in our design. Fig. 4.11 shows the energy per pixel required for 2D and 3D designs at different FPS. Our design achieves upto 127x improvement over the 2D design. We compute the total energy by adding the data transmission energy from the sensor to compute die and the compute energy. To calculate transmission energy, we use 1pJ/byte for TSV and 100pJ/byte for MIPI-CSI2 [44]. The energy improvement is mainly due to the reduction in data transmission energy enabled by 3D stacking of sensor and compute units.

Comparison with Prior works

We compare our work against [44], a 3D stacked In-Sensor accelerator for image compression, as shown in Table 4.3. [44] has a sensor layer stacked with a compute layer using TSVs and uses a two-stage encoder to compress and send the image. The design is implemented using 65nm and to ensure fair comparison, our design is scaled to 65nm using the DeepScaleTool [67]. [44] has much lower power consumption than our design and is mainly due to: (1) compared to our two die architecture, it has only one compute

Table 4.3: Comparison with prior work.

	Tech nm	Design	Frequency (Mhz)	Power (W)	NN performance[b] (DVS)
[44]	65	Sensor+ one stack	10	0.001	1.05 ms 3.23μj/fr
Our Work	28	Sensor+ two stack	800	0.239[a]	0.09 ms 22.3μj/fr

[a] @10 Mhz Scaled to 65nm using [67]. [b] Used same network as [44].

die (2) it uses In Memory-Compute (IMC), which performs energy-efficient MAC operations compared to systolic arrays [27, 30, 68]. In [44] the power to send the compressed image to a compute die using MIPI CSI-2 for decoding and for subsequent object detection is not accounted. Our design performs entire end-to-end detection and only needs to send the selected image patch, and thus is more efficient. Also IMCs are well suited for weight stationary architectures, thus making it inefficient for attention calculation of transformer models where the query, key and value matrices are dynamically generated. We achieve 11x improvement over [44] at 10MHz frequency, but have higher energy per frame due to higher power consumption.

4.6 Conclusion

With Moore's law slowing down and with the rapid evolution of AI models along with the huge volume of data and model size of the models, it is necessary to think beyond the traditional monolithic hardware design space. The next generation of AI accelerators must be easily scalable to add more performance and should support the huge bandwidth requirements of workloads but with the same or less area and power budget. In this work, we address the problem of edge computing for computer vision tasks using Vision Transformers. Traditional sensor interconnects cannot support the latency or bandwidth requirements of advanced sensors and real-time processing. In this work, we present the use of advanced packaging technology, F2F bonding with Cu pillars, to develop a two-stack AI accelerator that can be stacked with sensors using TSVs to achieve real-time

data processing. We develop a two-stage ViT algorithm for small object detection and efficiently map the algorithm to hardware, addressing the design concerns associated with 3D stacking such as IO circuitry, clocking, and thermal management. Our design achieves an 18x improvement in latency and 127x reduction in energy consumption compared to conventional 2D designs and an 11x improvement in latency compared to similar 3D architecture.

Chapter 5

Conclusion and Discussion

Artificial intelligence (AI) has progressed from dense Deep Neural Networks (DNNs) to a diverse range of models, including sparse Graph Convolutional Networks (GCNs) and Large Language Models (LLMs). To efficiently handle these evolving workloads, AI accelerators have become essential. However, designing AI accelerators remains a challenge due to variations in model size, processing flow, memory access patterns, and data/model sparsity. This thesis explores the critical design requirements for AI accelerators, namely reconfigurability, heterogeneity, and scalability, needed to optimize AI model acceleration. First, we propose FPGA Acceleration of GCN in Light of the Symmetry of Graph Adjacency Matrix, an accelerator design to handle heterogeneity within AI models such as GCNs. Different from deep neural networks (DNNs), GCNs are sparse, irregular, and unstructured, posing unique challenges to hardware acceleration with regular processing elements (PEs). In particular, the adjacency matrix of a GCN is extremely sparse, leading to frequent but irregular memory access, low spatial/temporal data locality and poor data reuse. Furthermore, a realistic graph usually consists of unstructured data (e.g., unbalanced distributions), creating significantly different processing times and imbalanced workload for each node in GCN acceleration. To address these issues we proposed a dynamically reconfigurable AI accelerator core to handle the alternating dense and sparse computation pattern of GCNs. In addition to this we also exploit the symmetry property of the graph which not speeds up computation but also increases data and reduces memory storage cost. We implement our accelerator design in Intel Stratix10 MX FPGA board with HBM2, and demonstrate 1.3×-110.5×

improvement in end-to-end GCN latency as compared to the state-of the-art FPGA implementations, on the graph datasets of Cora, Pubmed, Citeseer and Reddit.

The next work extend on this to build a heterogeneous RISCV SOC based heterogeneous AI accelerator to handle heterogeneity across AI workloads. In this work, we establish that homogeneous accelerators are not efficient in accelerating the diverse types of AI models, such as GCNs. In-memory computing (IMC) struggles to efficiently accelerate extremely sparse workloads due to its low operational intensity. To address this, we propose a scalable heterogeneous accelerator that integrates LIMC and SIMD cores, balancing computational demands with the efficiency required for CNNs, GCNs, and SpMVs. Our architecture consists of a RISCV CPU core and a ping-pong-based LIMC for MAC operations and SIMD cores optimized for sparse computations. We provide a workload mapping strategy to select the appropriate core based on workload sparsity. To ensure adaptability, custom instructions are employed for workload mapping, making the hardware flexible across a diverse range of inputs. Our design, implemented and taped out using Intel's 16nm PDK at 200 MHz. For CNN, GCN, and SpMV workloads we achieve up to 11× performance improvement over homogeneous accelerators, 2.1× higher TOPS/mm^2 over [36] at 16nm, and 20× better TOPS/W than state-of-the-art accelerators. The design strategy and analysis in this work can be extended to accelerate more diverse AI workloads.

The final part of the thesis addresses the design considerations of the next-generation AI accelerators. To keep up with the fast evolution of AI workloads and the proliferation of data volume, it is critical that accelerators can be easily scaled up to meet performance and data bandwidth requirements at the same or lower area and power costs. For this we consider a use case scenario for computer vision tasks with Vision Transformer as the AI model. By using advanced packaging technology namely F2F bonding with Cu pillars, we develop a two-stack AI accelerator that can be stacked with sensors using TSVs to achieve real-time data processing. We develop and co-design a two-stage ViT algorithm for small object detection and efficiently map the algorithm to hardware, while addressing the design concerns associated with 3D stacking such as IO circuitry, clocking, and thermal management. Our design achieves an 18x improvement in latency and 127x reduction in energy consumption compared to conventional 2D designs and an 11x improvement in latency compared to similar 3D architecture.

References

[1] JEDEC JEP157A. *Recommended ESD-CDM Target Levels.* Apr 2022.

[2] 3d microelectronic packaging 2021, volume 64. *3D Microelectronic Packaging 2021, Volume 64, year =.*

[3] Zhihao Zhang, Xuehui Wang, and Yuying Yan. A review of the state-of-the-art in electronic cooling. *e-Prime*, 1:100009, 10 2021.

[4] https://ourworldindata.org/scaling-up-ai.

[5] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[6] Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, Tianqi Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steve Reinhardt, and Martin C. Herbordt. Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 922–936, 2020.

[7] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 2224–2232, Cambridge, MA, USA, 2015. MIT Press.

[8] Hanjun Dai, Zornitsa Kozareva, Bo Dai, Alex Smola, and Le Song. Learning steady-states of iterative algorithms over graphs. In Jennifer Dy and Andreas Krause,

editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1106–1114. PMLR, 10–15 Jul 2018.

[9] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 4805–4815, Red Hook, NY, USA, 2018. Curran Associates Inc.

[10] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 974–983, New York, NY, USA, 2018. Association for Computing Machinery.

[11] Jianguo Jiang, Jiuming Chen, Tianbo Gu, Kim-Kwang Raymond Choo, Chao Liu, Min Yu, Weiqing Huang, and Prasant Mohapatra. Anomaly detection with graph convolutional networks for insider threat and fraud detection. In *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*, pages 109–114, 2019.

[12] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[13] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[14] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. Hygcn: A gcn accelerator with hybrid architecture. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 15–29, 2020.

[15] Bingyi Zhang, Rajgopal Kannan, and Viktor Prasanna. Boostgcn: A framework for optimizing gcn inference on fpga. In *2021 IEEE 29th Annual International*

Symposium on Field-Programmable Custom Computing Machines (FCCM), pages 29–39, 2021.

[16] Bingyi Zhang, Hanqing Zeng, and Viktor Prasanna. Hardware acceleration of large scale gcn inference. In *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 61–68, 2020.

[17] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. Nvidia tesla: A unified graphics and computing architecture. *IEEE Micro*, 28(2):39–55, 2008.

[18] Fazle Sadi, Joe Sweeney, Tze Meng Low, James C. Hoe, Larry Pileggi, and Franz Franchetti. Efficient spmv operation for large and highly sparse matrices using scalable multi-way merge parallelization. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, page 347–358, New York, NY, USA, 2019. Association for Computing Machinery.

[19] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[20] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[21] Shengwen Liang, Ying Wang, Cheng Liu, Lei He, Huawei LI, Dawen Xu, and Xiaowei Li. Engn: A high-throughput and energy-efficient accelerator for large graph neural networks. *IEEE Transactions on Computers*, 70(9):1511–1525, 2021.

[22] Teng Tian, Letian Zhao, Xiaotian Wang, Qizhe Wu, Wei Yuan, and Xi Jin. Fp-gnn: Adaptive fpga accelerator for graph neural networks. *Future Generation Computer Systems*, 2022.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[24] En-Yu Yang, Tianyu Jia, David Brooks, and Gu-Yeon Wei. Flexacc: A programmable accelerator with application-specific isa for flexible deep neural network inference. In *ASAP*, 2021.

[25] Anupreetham Anupreetham, Mohamed Ibrahim, Mathew Hall, Andrew Boutros, Ajay Kuzhively, Abinash Mohanty, Eriko Nurvitadhi, Vaughn Betz, Yu Cao, and Jae-sun Seo. End-to-end fpga-based object detection using pipelined cnn and non-maximum suppression. In *2021 FPL*, 2021.

[26] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IJSSCircuits*, 52(1), 2017.

[27] Zhenyu Wang, Gopikrishnan Raveendran Nair, Gokul Krishnan, Sumit K. Mandal, Ninoo Cherian, Jae-Sun Seo, Chaitali Chakrabarti, Umit Y. Ogras, and Yu Cao. Ai computing in light of 2.5d interconnect roadmap: Big-little chiplets for in-memory acceleration. In *2022 International Electron Devices Meeting (IEDM)*, pages 23.6.1–23.6.4, 2022.

[28] Naveen Verma, Hongyang Jia, Hossein Valavi, Yinqi Tang, Murat Ozatay, Lung-Yen Chen, Bonan Zhang, and Peter Deaville. In-memory computing: Advances and prospects. *IEEE Solid-State Circuits Magazine*, 11(3):43–55, 2019.

[29] Biresh Kumar Joardar, Aqeeb Iqbal Arka, Janardhan Rao Doppa, Partha Pratim Pande, Hai Li, and Krishnendu Chakrabarty. Heterogeneous manycore architectures enabled by processing-in-memory for deep learning: From cnns to gnns:. In *ICCAD*, 2021.

[30] Sumit K. Mandal, Gokul Krishnan, A. Alper Goksoy, Gopikrishnan Ravindran Nair, Yu Cao, and Umit Y. Ogras. Coin: Communication-aware in-memory acceleration for graph convolutional networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 12(2):472–485, 2022.

[31] Jinshan Yue, Xiaoyu Feng, Yifan He, Yuxuan Huang, Yipeng Wang, Zhe Yuan, Mingtao Zhan, Jiaxin Liu, Jian-Wei Su, Yen-Lin Chung, Ping-Chun Wu, Li-Yang Hung, Meng-Fan Chang, Nan Sun, Xueqing Li, Huazhong Yang, and Yongpan Liu.

A 2.75-to-75.9tops/w computing-in-memory nn processor supporting set-associate block-wise zero skipping and ping-pong cim with simultaneous computation and weight updating. *ISSCC*, 2021.

[32] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. Automated systolic array architecture synthesis for high throughput cnn inference on fpgas. In *DAC2017*, DAC '17, New York, NY, USA, 2017. Association for Computing Machinery.

[33] Jinshan Yue, Zhe Yuan, Xiaoyu Feng, Yifan He, Zhixiao Zhang, Xin Si, Ruhui Liu, Meng-Fan Chang, Xueqing Li, Huazhong Yang, and Yongpan Liu. 14.3 a 65nm computing-in-memory-based cnn processor with 2.9-to-35.8tops/w system energy efficiency using dynamic-sparsity performance-scaling architecture and energy-efficient inter/intra-macro data reuse. In *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 234–236, 2020.

[34] Tong Geng, Chunshu Wu, Yongan Zhang, Cheng Tan, Chenhao Xie, Haoran You, Martin Herbordt, Yingyan Lin, and Ang Li. I-gcn: A graph convolutional network accelerator with runtime locality enhancement through islandization. In *MICRO '21*, page 1051–1063, 2021.

[35] Gopikrishnan Raveendran Nair, Han-Sok Suh, Mahantesh Halappanavar, Frank Liu, Jae-sun Seo, and Yu Cao. FPGA Acceleration of GCN in Light of the Symmetry of Graph Adjacency Matrix. In *DATE*, 2023.

[36] Hongyang Jia, Murat Ozatay, Yinqi Tang, Hossein Valavi, Rakshit Pathak, Jinseok Lee, and Naveen Verma. 15.1 a programmable neural-network inference accelerator based on scalable in-memory computing. In *ISSCC*, 2021.

[37] Divya Govekar and Ameeta Amonkar. Design and implementation of high speed modified booth multiplier using hybrid adder. In *(ICCMC 2017)*, pages 138–143, 2017.

[38] Yu-Der Chih, Po-Hao Lee, Hidehiro Fujiwara, Yi-Chun Shih, Chia-Fu Lee, Rawan Naous, Yu-Lin Chen, Chieh-Pu Lo, Cheng-Han Lu, Haruki Mori, Wei-Chang Zhao, Dar Sun, Mahmut E. Sinangil, Yen-Huei Chen, Tan-Li Chou, Kerem Akarvardar,

Hung-Jen Liao, Yih Wang, Meng-Fan Chang, and Tsung-Yung Jonathan Chang. 16.4 an 89tops/w and 16.3tops/mm2 all-digital sram-based full-precision compute-in memory macro. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*.

[39] Gokul Krishnan, Gopikrishnan Raveendran Nair, Jonghyun Oh, Anupreetham Anupreetham, Pragnya Sudershan Nalla, Ahmed Hassan, Injune Yeo, Kishore Kasichainula, Jae-sun Seo, Mingoo Seok, and Yu Cao. 3d-isc: A 65nm 3d compatible in-sensor computing accelerator with reconfigurable tile architecture for real-time dvs data compression. In *2023 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, pages 1–3, 2023.

[40] Jiyue Yang, Tianmu Li, Wojciech Romaszkan, Puneet Gupta, and Sudhakar Pamarti. 65nm 8-bit all-digital stochastic-compute-in-memory deep learning processor. In *A-SSCC*, 2022.

[41] Wantong Li, Madison Manley, James Read, Ankit Kaul, Muhannad S. Bakir, and Shimeng Yu. H3datten: Heterogeneous 3-d integrated hybrid analog and digital compute-in-memory accelerator for vision transformer self-attention. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(10):1592–1602, 2023.

[42] https://www.mipi.org/download-mipi-whitepaper-introductory-guide-to-mass.

[43] Yu Feng, Tianrui Ma, Yuhao Zhu, and Xuan Zhang. Blisscam: Boosting eye tracking efficiency with learned in-sensor sparse sampling, 2024, 2404.15733.

[44] Gopikrishnan R. Nair, Pragnya S. Nalla, Gokul Krishnan, Anupreetham, Jonghyun Oh, Ahmed Hassan, Injune Yeo, Kishore Kasichainula, Mingoo Seok, Jae-Sun Seo, and Yu Cao. 3d in-sensor computing for real-time dvs data compression: 65nm hardware-algorithm co-design. pages 1–1, 2024.

[45] Minho Kwon, Seunghyun Lim, Hyeokjong Lee, Il-Seon Ha, Moo-Young Kim, Il-Jin Seo, Suho Lee, Yongsuk Choi, Kyunghoon Kim, Hansoo Lee, Won-Woong Kim, Seonghye Park, Kyongmin Koh, Jesuk Lee, and Yongin Park. A low-power 65/14nm stacked cmos image sensor. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2020.

[46] https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/product-briefs/3d-vcache.pdf.

[47] Jinwoo Kim, Lingjun Zhu, Hakki Mert Torun, Madhavan Swaminathan, and Sung Kyu Lim. Micro-bumping, hybrid bonding, or monolithic? a ppa study for heterogeneous 3d ic options. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1189–1194, 2021.

[48] https://www.synopsys.com/glossary/what-is-3dic.htmla.

[49] Eric Beyne. The 3-d interconnect technology landscape. *IEEE Design Test*, 33(3):8–20, 2016.

[50] Kyungwook Chang, Deepak Kadetotad, Yu Cao, Jae-Sun Seo, and Sung Kyu Lim. Power, performance, and area benefit of monolithic 3d ics for on-chip deep neural networks targeting speech recognition. *J. Emerg. Technol. Comput. Syst.*, 14(4), nov 2018.

[51] Sarah Eunkyung Kim and Sungdong Kim. Wafer level cu-cu direct bonding for 3d integration. *Microelectron. Eng.*, 137(C):158–163, apr 2015.

[52] Supreet Jeloka, Brian Cline, Shidhartha Das, Benoit Labbe, Alejandro Rico, Rainer Herberholz, Javier DeLaCruz, Rahul Mathur, and Shawn Hung. System technology co-optimization and design challenges for 3d ic. In *2022 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–6, 2022.

[53] P. Batude, B. Sklenard, C. Fenouillet-Beranger, B. Previtali, C. Tabone, O. Rozeau, O. Billoint, O. Turkyilmaz, H. Sarhan, S. Thuries, G. Cibrario, L. Brunet, F. Deprat, J-E. Michallet, F. Clermidy, and M. Vinet. 3d sequential integration opportunities and technology optimization. In *IEEE International Interconnect Technology Conference*, pages 373–376, 2014.

[54] https://www.appliedmaterials.com/us/en/semiconductor/markets-and-inflections/heterogeneous-integration/hybrid-bonding.html: :text=compared

[55] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted

windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[56] Tianyi Zhang, Kishore Kasichainula, Yaoxin Zhuo, Baoxin Li, Jae-Sun Seo, and Yu Cao. Patch-based selection and refinement for early object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 729–738, 2024.

[57] Tianyi Zhang, Kishore Kasichainula, Yaoxin Zhuo, Baoxin Li, Jae-Sun Seo, and Yu Cao. Transformer-based selective super-resolution for efficient image refinement. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 7305–7313, 2024.

[58] Pierre De Tournemire, Davide Nitti, Etienne Perot, Davide Migliore, and Amos Sironi. A large scale event-based detection dataset for automotive. *arXiv preprint arXiv:2001.08499*, 2020.

[59] Tianyi Zhang, Kishore Kasichainula, Dong-Woo Jee, Injune Yeo, Yaoxin Zhuo, Baoxin Li, Jae-sun Seo, and Yu Cao. Improving the efficiency of cmos image sensors through in-sensor selective attention. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2023.

[60] Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. A systematic methodology for characterizing scalability of dnn accelerators using scale-sim. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 58–68, 2020.

[61] Hyung Joon Byun, Udit Gupta, and Jae-sun Seo. 3d ic architecture evaluation and optimization with digital compute-in-memory designs. In *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '24, page 1–6, New York, NY, USA, 2024. Association for Computing Machinery.

[62] Alexander Graening, Saptadeep Pal, and Puneet Gupta. Chiplets: How small is too small? In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2023.

[63] https://www.intel.com/content/www/us/en/docs/programmable/683768/24-1/elements-of-an-ibis-model.htmls.

[64] Bryan Black, Murali Annavaram, Ned Brekelbaum, John DeVale, Lei Jiang, Gabriel H. Loh, Don McCaule, Pat Morrow, Donald W. Nelson, Daniel Pantuso, Paul Reed, Jeff Rupley, Sadasivan Shankar, John Shen, and Clair Webb. Die stacking (3d) microarchitecture. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, pages 469–479, 2006.

[65] Lennart Bamberg, Alberto García-Ortiz, Lingjun Zhu, Sai Pentapati, Da Eun Shim, and Sung Kyu Lim. Macro-3d: A physical design methodology for face-to-face-stacked heterogeneous 3d ics. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 37–42, 2020.

[66] Zhenyu Wang, Jingbo Sun, Alper Goksoy, Sumit K. Mandal, Yaotian Liu, Jae-Sun Seo, Chaitali Chakrabarti, Umit Y. Ogras, Vidya Chhabria, Jeff Zhang, and Yu Cao. Exploiting 2.5d/3d heterogeneous integration for ai computing. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 758–764, 2024.

[67] Satyabrata Sarangi and Bevan Baas. Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[68] Gopikrishnan Raveendran Nair, Fengyang Jiang, Jeff Zhang, and Yu Cao. A 16nm heterogeneous accelerator for energy-efficient sparse and dense ai computing. In *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '24, page 1–6, New York, NY, USA, 2024. Association for Computing Machinery.