5-1-2022

Efficient Networks-On-Chip Communication Support Solutions for Deep Neural Network Acceleration

Binayak Tiwari

EFFICIENT NETWORKS-ON-CHIP COMMUNICATION SUPPORT SOLUTIONS FOR DEEP NEURAL NETWORK ACCELERATION

By

Binayak Tiwari

Bachelor of Electronics and Communication Engineering
Tribhuvan University, Nepal
2013

A dissertation submitted in partial fulfillment
of the requirements for the

Doctor of Philosophy – Electrical Engineering

Department of Electrical and Computer Engineering
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2022

UNLV | GRADUATE COLLEGE

Dissertation Approval

The Graduate College
The University of Nevada, Las Vegas

April 8, 2022

This dissertation prepared by

Binayak Tiwari

entitled

Efficient Networks-On-Chip Communication Support Solutions for Deep Neural Network Acceleration

is approved in partial fulfillment of the requirements for the degree of

Doctor of Philosophy – Electrical Engineering
Department of Electrical and Computer Engineering

Mei Yang, Ph.D.
Examination Committee Chair

Yingtao Jiang, Ph.D.
Examination Committee Member

Henry Selvaraj, Ph.D.
Examination Committee Member

Mingon Kang, Ph.D.
Graduate College Faculty Representative

Kathryn Hausbeck Korgan, Ph.D.
Vice Provost for Graduate Education & Dean of the Graduate College

ABSTRACT

The increasing popularity of deep neural network (DNN) applications demands high computing power and efficient hardware accelerator architectures. DNN accelerators use a large number of processing elements (PEs) and on-chip memory for storing weights and other parameters. A significant challenge is faced when designing a many-core DNN accelerator to handle the data movement between the processing elements. As the communication backbone of a DNN accelerator, networks-on-chip (NoC) plays an important role in supporting various dataflow patterns and enabling processing with communication parallelism in a DNN accelerator. However, the widely used mesh-based NoC architectures inherently cannot efficiently support many-to-one (gather) and one-to-many (multicast) traffic largely existing in DNN workloads. This dissertation is focused on efficient communication support solutions for these traffic in DNN accelerators.

In NoCs, many-to-one traffic is typically handled by repetitive unicast packets which is inefficient. The dissertation first proposes to use the gather supported routing on mesh-based NoCs employing the Output Stationary (OS) systolic array in support of many-to-one traffic. Initiated from the left-most node, the gather packet will collect data generated from the intermediate nodes along its way to the global memory on the right side of the mesh. Without changing the router pipeline, the gather supported routing significantly reduces the network latency and power consumption than the repetitive unicast method evaluated under the traffic traces generated from the DNN workloads.

Further, the study is extended by proposing a modified mesh architecture with a one-way/two-way streaming bus to speed up multicast traffic and support multiple PEs per router using gather supported routing. The analysis of the runtime latency of a convolutional layer shows that the two-way streaming architecture achieves better improvement than the one-way streaming architecture for an OS dataflow architecture. Simulation results confirm the effectiveness of the proposed method which achieves up to 1.8× improvement in the runtime latency and up to 1.7× improvement in the network power consumption. The hardware overhead of the proposed method is justifiable for the performance improvements achieved over the repetitive unicast method.

Finally, In-Network Accumulation (INA) is proposed to further accelerate the DNN workload execution on a many-core spatial DNN accelerator for Weight Stationary (WS) dataflow model. The INA unit further improves the latency and power consumption by allowing the router to support the partial sum accumulation which avoids the overhead of injecting and ejecting the partial sum from and to the PE. Compared with OS dataflow model, the INA-enabled WS dataflow model achieves up to 1.19× latency improvement and 2.16× power improvement across different DNN workloads.

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Dr. Mei Yang, for her support and guidance during my journey. Her dedication to the research task and my success leads to the completion of this dissertation. I would also like to thank my committee members Dr. Yingtao Jiang, Dr. Henry Selvaraj, and Dr. Mingon Kang for their advice on improving the dissertation. I would like to thank Dr. Xiaohang Wang for his support and help while I was stuck setting up experiments. I would like to thank Dr. Grzegorz Chmaj for all the help and support that he provided me during my graduate studies.

I would like to thank all my labmates who shared their knowledge and learning with me. I would like to thank all my friends at UNLV/Las Vegas who made my journey enjoyable and memorable. I would like to thank my family for their unconditional support and love. Finally, to my better half, Sonia, without whom this journey would have been incredibly difficult.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this dissertation, the limitation of the existing communication backbone i.e., Network-on-Chips (NoC) on executing the Deep Neural Network (DNN) workloads is presented. The dissertation presents the improved communication architecture which helps in accelerating the traffic that exists in a DNN workload. For the rest of this chapter, Section 1.1 briefly introduces the concepts and ideas on DNN. Section 1.2 describes the challenges and opportunities in the field of DNN hardware execution. Section 1.3 presents the motivation of this disseration. Section 1.4 outlines the contribution of this dissertation and Section 1.5 summarizes the outline for rest of the dissertation.

1.1 Deep Neural Networks

Deep Learning is a subfield of machine learning which is an artificial intelligence (AI) field study that gives the computers the ability to learn without explicitly being programmed. Deep learning is framed by the development of deep neural networks which are widely adopted in a variety of applications ranging from speech recognition, object detection, self-driving cars to cancer detection, drug discovery and genomics [1] [2] [3].

Dated back to 1940s, the idea of neural networks is inspired by how biological neural systems work. Fig. 1.1 (a) shows the biological model of the neuron and its mathematical equivalent is shown in Fig. 1.1 (b). A neuron takes in the input/signal from dendrites,

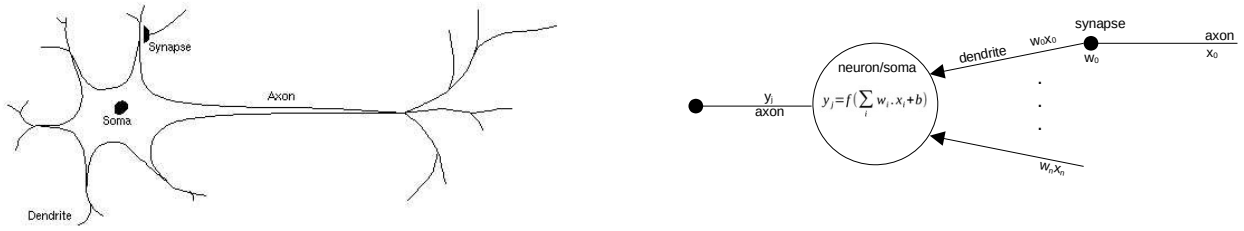


Figure 1.1: Neuron model (a) biological model, (b) mathematical model, weighted sum in a neuron x,w,f,b are input activations, weights, activation function, and bias, respectively (figures adopted from [6])

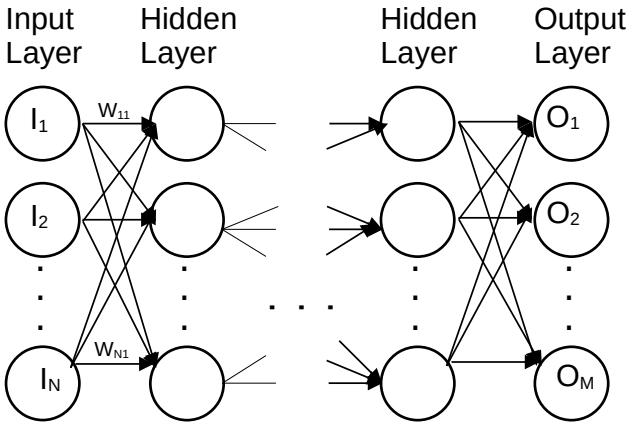


Figure 1.2: Example DNN model.

performs the operation on those inputs and provides the output to an axon. The axon of one neuron is connected to the dendrites of other neurons. As in improved version of the conventional artificial neural network, a DNN consists of multiple layers (more than three inclusive of input and output layers).

A DNN model may include tens of layers (such as convolutional layers, pooling layers, and fully connected layers) and millions of parameters. The neurons (activations) in each layer are connected to neurons (activations) in another layer in full or in part via synapses (weights) as shown in Fig. 1.2. The output of each neuron in Fig. 1.2 can be expressed as

the operation shown in Equation (1.1).

$$Output = \mathcal{F}\left(\sum_{i=0}^{N-1} I_i \cdot W_{i,1} + b\right) \tag{1.1}$$

where, $W_{i,1}$ represents the weights and I_i represents the input activation for the neurons in a particular layer containing N neurons. *Output* represents the output activation which will be fed as an input to another layer, and $\mathcal{F}(\cdot)$ is the activation function like ReLU, sigmoid, etc.

DNNs vary in a number of layers, the operation these layers perform, the size of inputs and weights on these layers, etc. Most of the DNNs have a convolutional (CONV) layer and a fully-connected (FC) layer. DNNs with only FC layers are called multi-layer perceptron (MLP) and the ones with CONV layers are called convolutional neural networks (CNNs). Apart from MLP and CNNs, there are other kinds of DNNs i.e., Recurrent Neural Network (RNN) [9], Transformers [10], General Adversarial Networks (GANs) [11], etc. Although the types of application for other DNNs vary from CNNs, most of the building blocks and basic underlying operations remain the same. For example, RNNs and transformers are dominantly memory bound and these are similar to the FC network in terms of network traffic. In this dissertation, the focus is on the traffic optimization methods inside DNN accelerators which are equally valid for all types of DNNs though the evaluation done in CNNs like AlexNet [7], VGG-16 [8], ResNet50 [12].

The CONV layer performs an element-wise multiplication and accumulation operation as shown in Fig. 1.3 (a). Each element in a filter (F) is multiplied with a corresponding input feature map (I) to produce a partial sum. These partial sums are accumulated to get an

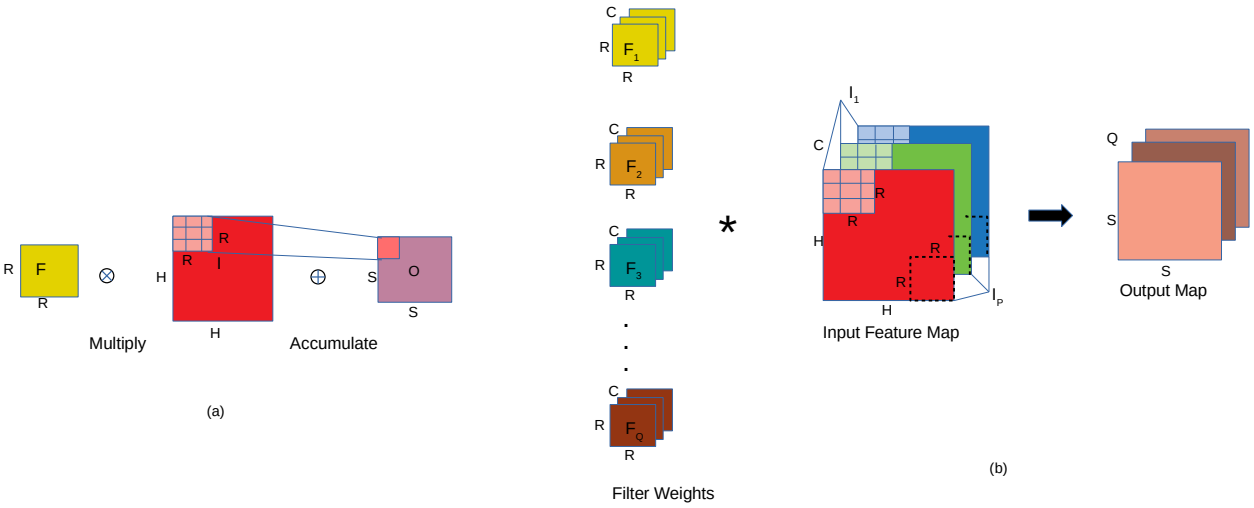


Figure 1.3: Convolution operation (a) 2D convolution, (b) higher dimensional convolution

element on an output feature map (O). Multidimensional CONV layers shown in Fig. 1.3 (b) are usually composed of Q filter weights each with dimension $C \cdot R \cdot R$, and input feature maps with dimension $C \cdot H \cdot H$. The output map can be parallelized in hardware since the multiply and accumulate operation can be performed simultaneously. This also provides an opportunity in reusing the weights or inputs that are already loaded from the memory to reduce the memory transactions in the hardware.

DNNs include the training phase and the inference phase. In the training phase, learning is involved in determining the network weights and the biases. The inference phase is actually taking the inputs from the user or sensor and making use of the weights and biases obtained during the training phase to get the estimated result. Training DNNs often requires the use of a large dataset and is more computation-intensive than inference. Training is not performed frequently which is also a time-consuming process that may take up to several weeks at a cloud/data center. On the other hand, inference usually happens at edge devices like mobile phones which are directly performed by the users. This dissertation is focused

on the inference phase of the DNN model.

As DNNs are continuously evolving, the DNN executing hardware infrastructure should also be able to support the diversity and non-uniformity in DNN models. Unlike some other standardized protocols in telecommunications, MP3 encoding/decoding, etc., there is no one standard on how hardware can implement DNN models. This leads to the problem of non-uniformity in the computing and communication requirement for DNN hardware. Depending on the application, users can employ the techniques like pruning, quantization, etc. to further reduce the DNN complexity. These complexity-reducing techniques also contribute to the non-uniformity in DNN execution. Hence, the DNN executing hardware infrastructure should also be able to support this diversity in DNN models and not necessarily designed to execute certain DNN models.

1.2 Challenges and Opportunities in DNN Execution

The availability of training data and advancement in high-performance computing leads to the pervasiveness of DNNs. However, there is also a disparity between the rate at which DNN architecture is evolving and the underlying hardware that executes DNN to satisfy the need of real-world applications. DNN execution demands a high computing and power budget, however, many AI applications demand the execution on a hardware with limited computation and power budget. Hence, traditional general-purpose processors are no longer able to cater to this need, which drives the research on domain-specific processors i.e., accelerators [4].

DNNs can extract the high-level features from the input data as in statistical learning

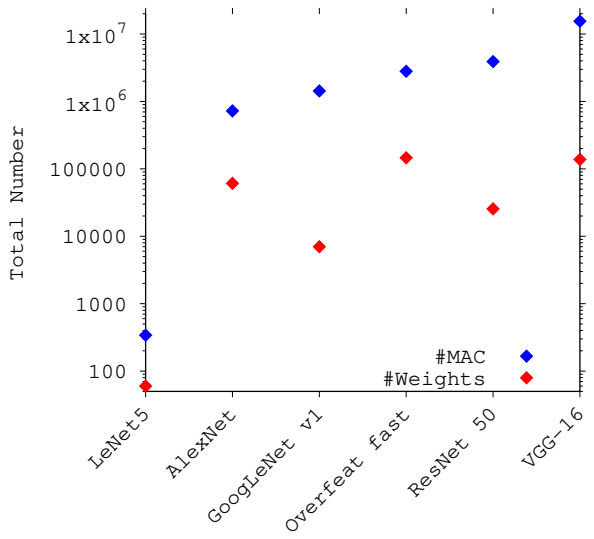


Figure 1.4: Some popular DNN models with number of weights and MAC operations.

compared to hand-picked features from classic machine learning. This has enabled DNNs to achieve human-level accuracy which comes at the cost of high communication and computation complexity. High complexity in DNNs is due to the involvement of a huge number of parameters and multiply-and-accumulate (MAC) operations. Fig. 1.4 shows the number of weights and MAC operations used in some of the popular DNN models. AlexNet [7] consists of 61M weights and 724M MACs, while VGG-16 [8] consists of 138M weights and 15.5G MACs.

Authors in [13] demonstrated that more than 90% of a certain class of DNN workload execution time on a single-threaded CPU was spent on a convolution layer. Similarly, in most commonly used DNNs the multiply and accumulate operation in CONV and FC layers consume more than 90% of the total operations involved. Fig. 1.4 shows the amount of MAC operations involved in some of the popular DNNs. The same figure also shows the total weights involved in these DNNs. Hence, due to the volume of MAC operation and data (weights, inputs) involved, the DNN execution bottleneck in hardware can be broadly

Table 1.1: Convolution layers for AlexNet [7] & VGG-16 [8]

Model	Layers	Kernels (C × #kernels @ R × R)	Layer Size (C @ H × H)
AlexNet [7]	Input		3@224x224
	Conv1	3x64@11x11	64@55x55
	Conv2	64x192@5x5	192@27x27
	Conv3	192x384@3x3	384@13x13
	Conv4	384x256@3x3	256@13x13
	Conv5	256x256@3x3	256@13x13
VGG-16 [8]	Input		3@224x224
	Conv1	64x64@3x3	64@224x224
	Conv2	128x128@3x3	128@112x112
	Conv3	256x256@3x3	256@56x56
	Conv4	512x512@3x3	512@14x14

categorized into communication and computation.

The computation bottleneck is fairly straightforward to resolve. This involves adding or increasing the number of processing elements (PEs) or computation nodes. However, it is also to note that increasing computation resources will increase a fair amount of complexity in the communication infrastructure. Additionally, DNNs comes in a variety of shape and sizes which include the diversity in layers, kernels, etc. Computation needs should be reconfigurable to be able to handle these diverse workloads. Table 1.1 shows the diversity in different convolution layers in Alexnet [7] and some representative layers from VGG-16 [8].

In a DNN accelerator, PEs perform MAC operations while the involved parameters are usually stored in the global memory. There is a need of transferring data from global memory to PEs and vice versa. PEs and the memory elements are often interconnected by a Networks-on-Chip (NoC) [14] [15] [16] for realizing high throughput. These PEs operate in parallel and reduce the memory access as much as possible by sharing and reusing the parameters

with each other. Also, while sharing and resuing of these data happens, different kinds of traffic patterns exist in the DNN execution. In addition to one-to-one traffic, there are a large amount of one-to-many (input/weight distribution) and many-to-one traffic (result collection) in a DNN workload. The disparity of traffic patterns imposes great challenges in the communication supporting schemes of a DNN accelerator.

1.3 Motivation

As the communication backbone [17] [18] [19] of a DNN accelerator, NoC plays an important role in supporting various traffic patterns and dataflow models, enabling processing with communication parallelism, and enhancing scalability. Most of the recently proposed DNN accelerators adopt the mesh topology, which is simple in design but also has some drawbacks. Some common problems that can be addressed to improve the execution of a DNN workload in mesh-based accelerators, as listed below:

- Lack of support for many-to-one traffic: Existing mesh-based accelerator systems focus more on improving scalability and data reuse, and little attention is given to enhancing communication support that is abundant in the DNN workloads i.e., many-to-one (gather) traffic.

- Incompatible exisiting multicast algorithm: In the literature, different approaches have been proposed to support multicast traffic in NoCs [19] [20]. Noticeably, in a DNN workload, multicast traffic tends to have a fixed communication pattern. Thus, existing multicast algorithms is not suitable for DNN workloads.

- Increasing the computation parallelism with limited communication overhead: As

pointed out in Section 1.2, the computation throughput of a DNN accelerator can be improved by increasing the number of PEs. There is a need of an architecure that will scale up the computing bandwidth of the DNN layers at the same time not requiring too much of the overhead in communication and memory requirements.

- As the DNN models are evolving, dataflow models are also improving. There are certain dataflow models which may not perform well in other kinds of hardware tuned to execute on different dataflow models. There is a need to identify the common computing constraints that would work for different dataflow models.

1.4 Objectives

This dissertation aims to address the aforementioned problems and provide solutions to enhance the communication support and computation throughput in DNN accelerators. The following research tasks are conducted:

- Support for many-to-one traffic: This dissertation proposes the gather supported routing scheme to support many-to-one type of traffic on the Output Stationary (OS) data flow model on mesh-based DNN accelerators. The use of gather packets helps in reducing the network congestion and power consumption as well.

- Support for one-to-many traffic: This dissertation proposes to use streaming buses to stream input activations and weights from the memory elements to the PEs in the same row and column, respectively. The streaming buses overcome the additional routing overhead and thus improve the runtime latency and power consumption of a DNN workload.

- Support for multiple PEs per router: While improving the computation bottleneck is fairly straightforward, this work proposes the support for multiple PEs per node which helps in distributing the weights/inputs effectively and maintaining compute level parallelism to accelerate the DNN workload.

- In-Network Accumulation (INA) for different dataflow models: This dissertation further proposes a modified router architecture i.e., the INA that is capable of accelerating different types of dataflow models which exist in modern DNN accelerators.

1.5 Outline

The rest of the dissertation is organized as follows. Chapter 2 of this dissertation provides the background and related works in NoC, dataflow models and existing DNN acclerators. Chapter 3 presents the gather supported routing method to support many-to-one traffic. Chapter 4 presents the streaming architecture to address the DNN suitable multicast solution. Chapter 5 presents the solution on scaling the computation throughput for DNN accelerators. Chapter 6 presents the in-network accumulation architecture to support other dataflow models and finally, Chapter 7 provides the conclusion and future work of this dissertation.

CHAPTER 2

BACKGROUND AND RELATED WORKS

In this chapter, all the necessary background and related works to understand the dissertation is presented. Section 2.1 briefly introduces the traffic patterns in DNN workloads and dataflow models and Section 2.2 describes the existing solutions that have been proposed to address the need and requirement for DNN execution in hardware accelerators.

2.1 Background

2.1.1 Traffic in DNN Workloads

A typical DNN model is shown in Fig. 1.2 where multiple layers are interconnected to each other. While implementing these layers in hardware, typically neurons are mapped to PEs inside a DNN accelerator. These neurons share the weights stored in the memory element, similarly, the outputs of the neurons in one layer are the input to the neurons in the adjacent layer. This sharing of data between adjacent PEs (neurons) creates traffic inside accelerators which can mainly be classified as one-to-one (unicast), one-to-many (multicast), and many-to-one (gather) as shown in Fig. 2.1.

Different from conventional parallel workloads like PARSEC [23], DNNs involve a significant amount of many-to-one traffic in addition to one-to-one and one-to-many types of traffic. Unicast traffic usually occurs when sending an input activation or weight from a

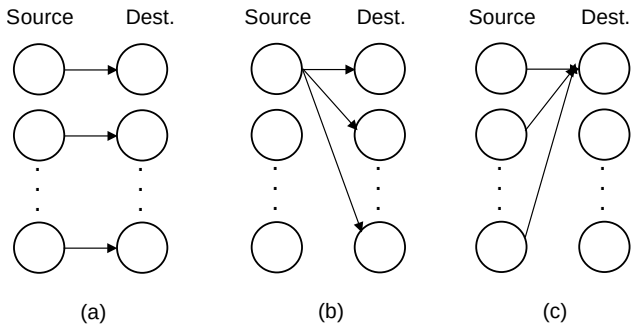


Figure 2.1: Traffic pattern inside a DNN hardware (a) unicast (b) multicast (c) gather

memory element to a PE or any other inter-PE traffic. Multicast traffic mainly covers the distribution of weights from the memory element to multiple PEs. Different dataflow mechanisms can be used to support multicast traffic for weight distributions. Gather traffic is used to collect the output from multiple PEs to the memory element. Due to the limitation of computing resources, the inference operation of a DNN workload is performed in multiple rounds. When one round of MAC operations is completed, the intermediate results will be gathered back to the memory element before initiating a new round.

The output of each neuron in Fig. 1.2 can be expressed as the operation shown in Equation (1.1). Many DNNs consist of multiple layers where both convolution and fully connected layers perform MAC operation as shown in (Equation (1.1)). These layers are computationally intensive and hence performed in multiple PEs in a distributed way. Moreover, the weights and inputs are stored in the memory element and these steps require frequent access to the memory element which is an expensive task in terms of latency and energy [25]. Data communication in a DNN accelerator consumes around 30% of the total energy and increases with the system scaling [24]. Hence, traffic in a DNN workloads and communication support is an important aspect of an efficient DNN execution.

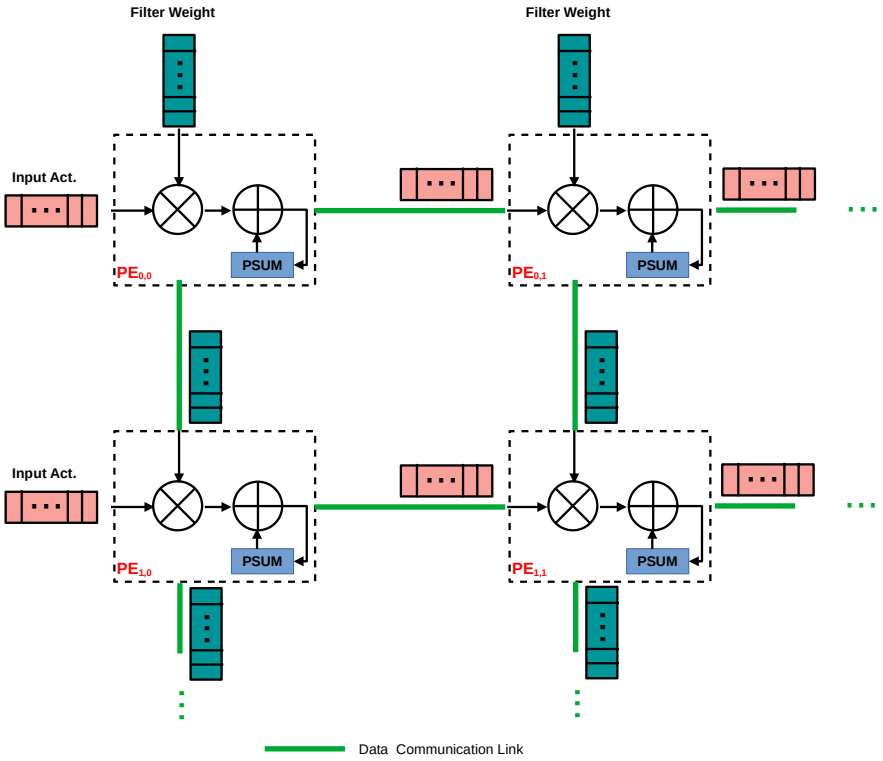


Figure 2.2: OS dataflow model

2.1.2 Dataflow Models

The number of computations that should be performed in a DNN workload is very high compared to the computing resources available in a DNN accelerator. Therefore, the operation needs to be broken down into chunks and executed on the limited number of computing resources available in the accelerator. This presents the opportunity for the data resue i.e., the tensor used by one operation is also used by another operation in time by the same PE or in space by another PE. This temporal and spatial data resue depends on the dataflow technique employed in an accelerator.

The dataflow determines the processing order and where data is stored and reused, i.e., the way data (i.e., inputs, weights, and partial sums) communication happens between the

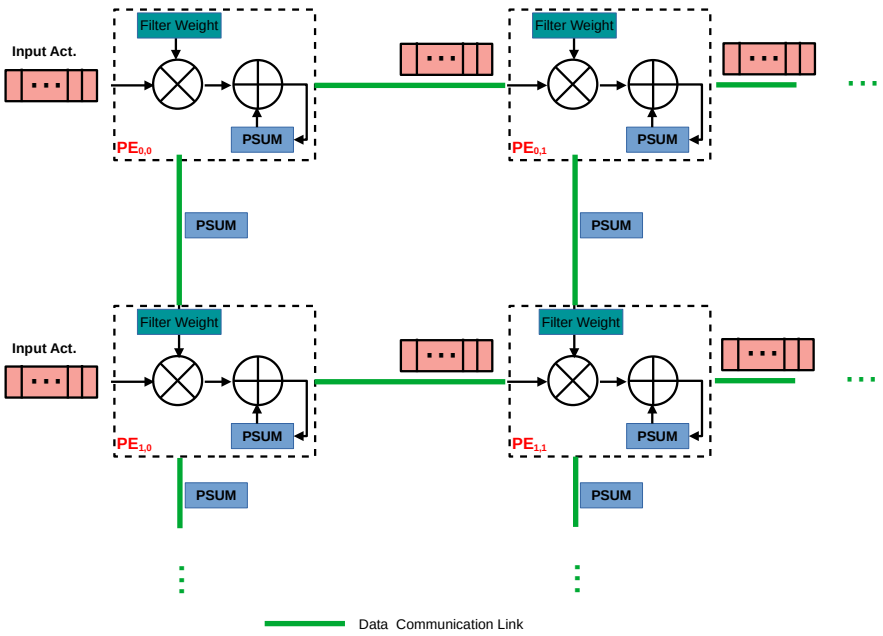


Figure 2.3: WS dataflow model

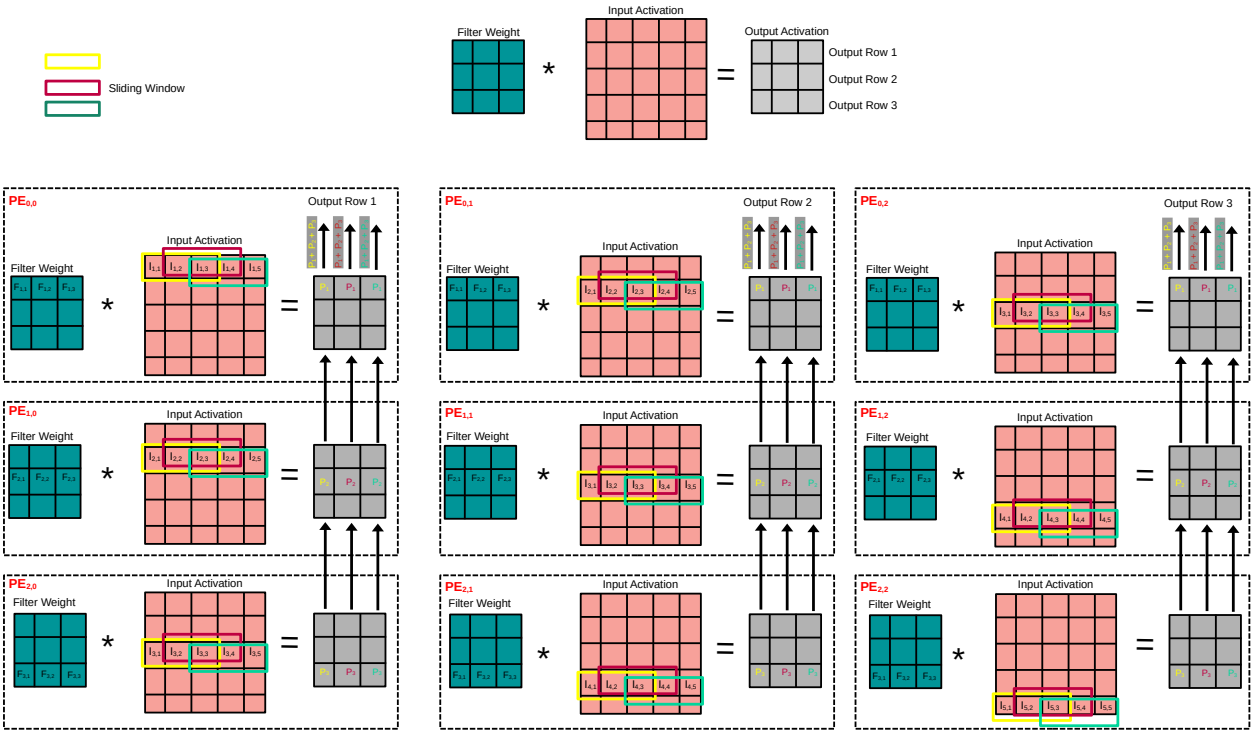


Figure 2.4: RS dataflow model

PE and memory element. In the literature, various dataflow models have been proposed [25] including Weight Stationary (WS), Output Stationary (OS), Row Stationary (RS), and No Local Reuse (NLR), each of them has its own memory usage and energy advantages.

The examples of the OS, WS, and RS dataflow models are shown in Fig. 2.2 - 2.4, respectively. Fig. 2.2 shows the OS dataflow where input activations and weights are streamed in a row-wise and column-wise manner respectively while the partial sums are accumulated on a PE. ShiDianNao [28] is one example of a design that uses an OS dataflow model. The choice of dataflow depends on many factors such as the input size, the number of kernels, stride, etc., mapping scheme of the DNN workload onto the architecture, and DNN optimizations like pruning, sparsity [15]. An inefficient dataflow model will cause stalls as appropriate data may not be available at the PE when needed and low data reuse so that the same data must be fetched multiple times from the memory, thus resulting in higher latency and energy inefficiency. Compared with other dataflow models, the OS dataflow model achieves good performance with less complexity. This dissertation will implement the proposed solutions on the OS dataflow model first and then extend to WS dataflow model.

Fig. 2.3 shows an example of the Weight Stationary (WS) dataflow model where each PE keeps the filter weight stationary in a PE while the input activations and partial sums move across the PE array. When these weights are no longer needed, they will be replaced with other weights. This setting reuses the weights and minimizes the energy consumption in reading the weights. Google's TPU [29] is one example of a design that uses the WS dataflow model. Opposite to the WS dataflow is the Input Stationary (IS) which focuses on reusing the input activation and minimizing the energy consumption in reading the input activations. IS model keeps the input stationary in a PE for further access while weights

and the partial sum move along the PE array.

The Row Stationary (RS) as shown in Fig. 2.4 on the other hand aims in maximizing the reuse of all types of data i.e., weights, input activation, and partial sums. In an RS dataflow, each row of the convolution processing (output row in Fig. 2.4) is mapped into each PE so that eventually the partial sum for a row of a convolution are spread across multiple PEs (in a column) which can be accumulated to get the final output convolution. In this model, filter weights are kept stationary inside a PE, and inputs are streamed. The MAC operation for each sliding window is performed at a time and a partial sum is generated. The generated partial sums are finally accumulated columnwise to get the output activations. Due to the nature of the dataflow, the input will also get reused between different sliding windows. Eyeriss DNN accelerator [30] proposed the RS dataflow model.

2.2 Related Works

2.2.1 DNN Accelerators

DNN processing involves tens of layers and a large number of multiply-accumulation (MAC) operations using millions of parameters, which imposes tremendous throughput and energy-efficiency challenges to the computing platforms. To cope with this challenge, temporal architectures like Single-Instruction, Multiple-Data (SIMD)/Single Instruction, Multiple-Thread (SIMT) are used in CPU/GPU platforms where computing elements share the control unit and memory element to perform the MAC operation in parallel. The operations in convolution and fully-connected layers are often mapped to a matrix multiplication for efficient processing [31] [32]. However, the general-purpose CPUs provide the least performance ben-

efit because of the limit in parallelism. Though GPUs solve the problem of parallelism, they tend to consume high power.

On the other hand, spatial architectures use a distributed approach where the processing elements (PEs) may have their own control logic and limited local memory. Recently, the spatial DNN accelerators like [29] [30] [28] are gaining attention as they are optimized to handle DNN processing effectively. Commonly used in FPGA and ASIC based designs [25] [29] [33] [34], spatial architectures use a distributed approach which adopts a large number of PEs each having its own control logic & limited local memory and shared global memory. Communication between PEs is allowed which enables the data movement between each other. In the design of a DNN accelerator, a major consideration is optimizing data movement which aims to minimize the global memory access and thereby reduce the power consumption during the DNN processing.

In [29], authors presents a WS model where weights are stationary at the PEs while the inputs and partial sums will move through the PEs and the memory element. On the contrary, the OS model has the output stationary at the PEs while the inputs and weights move which is implemented in [28]. The NLR model [24] focuses on increasing the size of the global buffer at the expense of a register file and thus decreasing the DRAM accesses. The RS model increases the reuse of all data types rather than focusing on the reuse of one type. Systolic architecture [35] is another efficient way of implementing the convolution operation in hardware. In [30], various systolic CNN architectures are described which can be applied to mesh-based NoCs.

DNN workloads contain different types of communication traffic which handle the data movement like partial sums, weights, and inputs streams to and from the memory. In a DNN

accelerator [36], data movement is expensive in terms of energy which consumes around 50% of the total energy. In some cases, data movement can even increase the latency [30] due to the communication bottleneck. Although a bus-only based system is proposed in some prior work, this kind of system will quickly become the bottleneck when the DNN size increases [37]. This observation leads to the works which are focused on the NoC architecture and communication support of the DNN accelerators [15] [38] [39] [40] [41] [42].

2.2.2 NoC in Accelerators

Network-on-Chip (NoC) has emerged as the de-facto standard for on-chip communication in multi/many-core systems [21]. The need for NoC in a DNN accelerator is ever increasing and becoming important because of the inherent nature of DNN computation and the on-chip traffic involved. Another important feature of NoC is scalability where nodes or processing elements (PE) can be added with minimal changes. Modular design property helps in gating and thus saving power by turning off the unused modules. This also provides flexibility in running different kinds of workloads on the same NoC-based system [22].

Various study has been done in NoC topology to accelerate a DNN workload [36] [37] [42] [43]. In [42], a hierarchical Neu-NoC architecture is proposed which adopts a hybrid ring-mesh topology. Multiple PEs are connected in a group of rings that are connected via a mesh topology. This structure reduces the communication distance and shows better performance against the bus and tree structures. Authors in [43] propose a reconfigurable topology for a 3D neural network accelerator that can be reconfigured as a tree to handle the multicast traffic. A many-core system SpiNNaker is proposed to simulate the spiking neural networks with torus network topology. In [37], the authors study different topologies and conclude

that the mesh NoC is better for realizing spiking neural networks, compared with the tree, point-to-point, and bus-based structures. In [36], a fat tree and a mesh are used for intrachip communication and data movement among chips, respectively. The separation of intrachip and interchip communication may create a bottleneck for the gather traffic abundant in a DNN workload.

Changes in the NoC topology also cause the change in the communication cost and support of different traffic patterns. Authors in [30] propose a mesh-based interconnection network called hierarchical mesh network for DNN processing. The PEs and memory elements are grouped into a cluster which is then connected via the hierarchical mesh network. The NoC is capable of configuring the network topology based on the needs. The NeuronLink [39] is a chip-to-chip interconnection network for large neural networks that support both interchip and intrachip communication. Each chip consists of 16 PEs in a mesh topology and 4 such chips are connected in a star topology to handle a large amount of unicast and multicast traffic. The artificial intelligence (AI) computing system from Cerebras [26] uses 2D mesh as a communication topology to connect thousands of AI cores. Groq [27] reorganizes 2D mesh into functional slices to optimize the microarchitecture.

Various routing methods are adopted to fulfill the communication needs especially multicast and gather traffic in a DNN accelerator. Authors in [39] adopt XY routing for unicast traffic and a table-based routing for multicast traffic. Authors in [30] propose different NoC configurations for each datatype i.e., input activation, weights and partial sums. Further, this method is suitable for RS dataflow architecture where partial sums are accumulated across multiple PEs and hence not suitable to perform gather for unique partial sums across the PEs which exist in OS dataflow architecture. In [44], authors have proposed an array

of microswitches that are configured to handle different kinds of DNN traffic by creating a tree. In ClosNN [40], one or more layers can be conducted on the network by mapping the neurons (PEs) on the input/output ports. Various stages of switching are done to connect the input and output ports in ClosNN depending on the type of data traffic.

Since the field of DNNs is evolving rapidly, hardware design should also be able to cope with this pace. As a widely adopted NoC topology, mesh is used in most of the recently proposed DNN accelerators [26] [27] [39] [37] [45]. As many-to-one and one-to-many traffic are not inherently supported in a mesh topology, they are typically modeled as repetitive unicast (RU) traffic in existing work [45] [30] which turn out to be inefficient. Recent work modifies the topology to simulate a tree or Clos network [40] [44] to support many-to-one traffic. Hence, this dissertation proposes an effective communication solution to many-to-one traffic for a mesh-based NoC along with other improvements on the communication aspect of the DNN accelerators.

CHAPTER 3

SUPPORTING MANY-TO-ONE TRAFFIC

This chapter introduces many-to-one communication support solutions for a DNN accelerator. Section 3.1 provides the motivation behind this study. Section 3.2 describes the proposed routing algorithm and necessary support to enable gather traffic. Section 3.3 shows the proposed changes in the router microarchitecture to enable gather supported routing. Section 3.4 provide the implementation results and finally, Section 3.5 summarizes this chapter.

3.1 Motivation

It is clear that in order to utilize the best potential of the DNN accelerators we need to parallelize the MAC operation. In a DNN accelerator, PEs perform MAC operations while the involved parameters are usually stored in the global memory. There is a need of transferring data from global memory to PEs and vice versa. PEs and the memory elements are often interconnected by a Networks-on-Chip (NoC) [14] [15] [16] for realizing high throughput. These PEs operate in parallel and reduce the memory access as much as possible by sharing and reusing the parameters with each other, especially in the spatial architectures. Effective utilization of the local PE memory requires the existing data to be moved to the global memory which creates an abundant amount of many-to-one traffic.

Fig. 3.1 shows a 6x6 mesh with six PEs ready to send data to the global buffer, Fig.

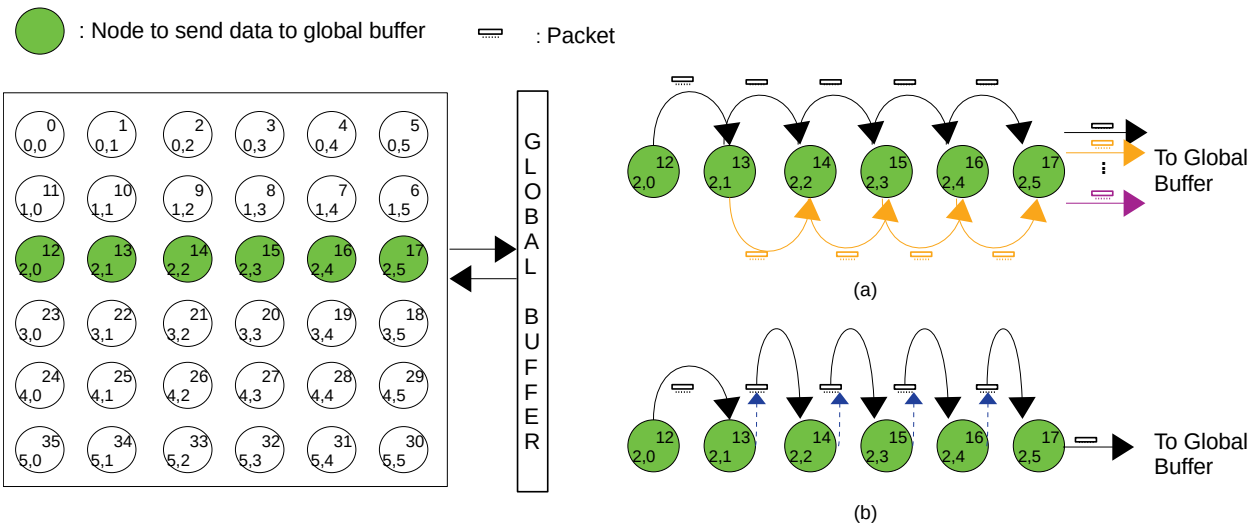


Figure 3.1: 6x6 mesh example (a) without gather support (b) with gather support

3.1(a) shows the operation without the gather support where each PE sends a unicast packet to the global buffer which requires 15 hops to get the data to the buffer. Is it possible to collect the data to be sent to the buffer into one gather packet? Fig. 3.1(b) illustrates the gather packet initiated from node 12 and also provides an answer to the above question. As the gather packet progresses along the route, it will enclose the data payload from each intermediate node. The hop count for the gather packet is only 5. It is clear that the gather packet is able to deliver the data to the global buffer with less resource utilization by removing the redundant use of the resources compared to the repetitive unicast method.

3.2 Gather Supported Routing

In this section, the NoC architecture and data flow on a systolic array will be introduced followed by the description of the gather supported routing scheme and analysis of the performance gain.

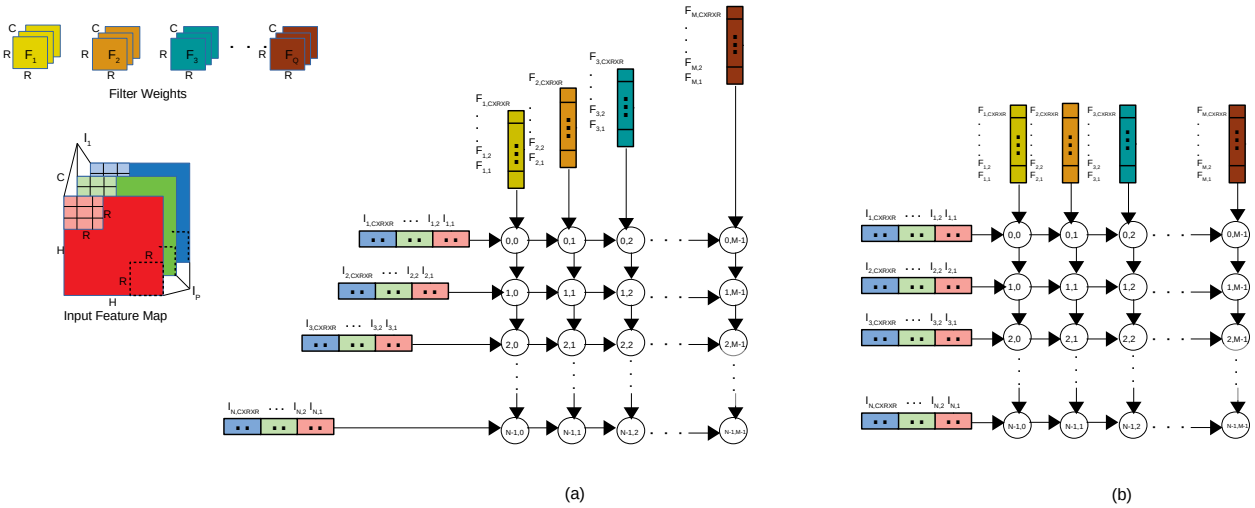


Figure 3.2: OS Dataflow in NxM Mesh NoC (a) systolic array based (b) streaming bus based

3.2.1 Data Flow Model

The mesh-based NoC adopting the Output Stationary (OS) [44] [30] systolic architecture is shown in Fig. 3.2(a). Each PE can perform the simple multiply and accumulate (MAC) operation. The input data and filter weights are fed from the left and top edges to the left-most column PEs and top-most row PEs, respectively. The output data will be stored to the global buffer located on the right side of the NoC (similar to Fig. 3.1) which will be accessed by the subsequent layers. These PEs will propagate the data to the PEs on the same column and row until all the PEs receive the data. One of the advantages of systolic architecture is that the whole operation is repeated in a pipelined manner making it easier to perform input and output operations. In each pipeline stage, each PE stores the data received from its left and top neighboring nodes, perform the MAC operation, and forwards the same data to the right and bottom neighbor nodes in parallel. This helps in reducing a significant amount of memory operations. Similarly, Fig. 3.2(b) shows another way of performing the same operation using streaming bus architecture, more on this kind of architectures are explained

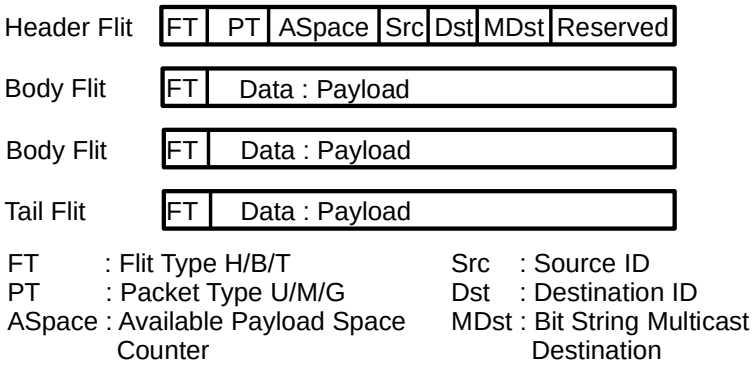


Figure 3.3: Packet format

in Chapter 4.

Without the loss of generality, assume that there are $N \times M$ PEs in the mesh-based NoC. The P inputs or feature maps of size $H \cdot H$ with C channels are streamed from the left edge of the NoC and Q filters each of size $R \cdot R$ with C channel are streamed from the top edge. Each PE will receive a total of $C \cdot R \cdot R$ input and weight data then perform $C \cdot R \cdot R$ MAC operations. On this setting $PE_{0,0}$ will be the first to finish the operation and then $PE_{1,0}$, $PE_{0,1}$ are the second in line, and so on. When all P inputs are convolved with Q filters the convolution operation is complete. Noticeably $P \cdot Q$ is not a trivial number and the convolution will be completed by $N \times M$ PEs in multiple rounds. Thus, once the $C \cdot R \cdot R$ data is streamed, partial sum (PS) output as shown in (3.1) is sent to the global buffer before the next set of data is streamed.

$$\underset{i\in\{1\ldots P\};k\in\{1\ldots Q\}}{PS_{i,k}} = \sum_{j=1}^{C\cdot R\cdot R} (I_{i,j} \cdot F_{k,j}) \tag{3.1}$$

3.2.2 Routing Scheme

The process of moving the data from the PE to memory is gather traffic or many-to-one traffic. This section proposes a gather traffic support routing scheme. The PE which has finished the operation first will initiate the gather packet towards the memory or a global buffer and on its way, it will collect all the available results from the intermediate nodes on the same row until its capacity is reached. Fig. 3.3 shows the packet format for the proposed gather support. *FT* is the flit type that includes the head, body, and tail identifier. *PT* is the packet type that includes unicast, multicast, and gather type identifier. *ASpace* in the header flit is used to identify available space in a packet for a node to upload its payload. *Src* and *Dst* are the identifiers for source and destination node and finally, *MDst* is the bit string multicast destination representation. Both body and tail flits include the payload field.

Algorithm 1: Flow for the Gather Support

Input : Arriving flit (F), Gather Payload (P)
Output: Updated flit (F) or initiate a gather packet

```
1 if ((F.FT = H) and (F.ASpace >= sizeof(P)) and (F.PT = G)) then
      // generate a load signal
2     if (F.Dst = P.Dst) then Load ← 1
      // update F.ASpace before switch traversal
3     if (Load = 1) then F.ASpace ← F.ASpace - sizeof(P)
4 end
5 if ((F.FT = B or F.FT = T) and (Load = 1)) then F.Data ← P.Data
6 else Can initiate a gather packet after δ clock cycles
7 if (F.FT = T) then Load ← 0
```

Algorithm 1 shows the flow for the gather supported routing implemented at each router. The incoming header flit and the information from a gather payload are used to generate

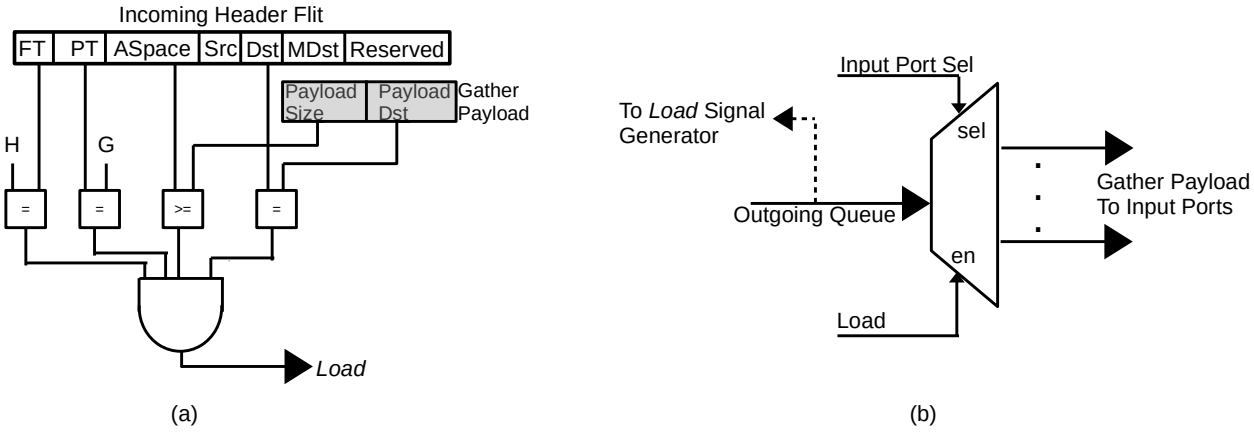


Figure 3.4: (a) *Load* signal generator, (b) payload generator

a *Load* signal, which indicates the router to fill a gather payload in an incoming body or a tail flit by appending the payload. Ideally, the size of a gather payload is considered to be less than a flit size. Fig. 3.4(a) shows the logic to generate a *Load* signal; the space counter *ASpace* is decremented using the *Load* signal so that other PEs can estimate the space for filling their gather payloads. If the *ASpace* is less than a gather payload size, the router can initiate its own gather packet. However, to avoid the flooding of gather packets, each router must wait for the timeout period of δ cycle so that any other previously generated gather packet can go through.

The value of δ can be determined based on the router pipeline stages. Additionally, δ can be fine-tuned further for an individual router, if required. A too low value of δ will result in an increased amount of packets in the network, leading to congestion and increased latency, while a too high value of δ will cause nodes to wait too long for an incoming gather packet, which may increase the latency of the packets. Noticeably, δ also serves as a fault tolerance mechanism. If a link is faulty, then the node can initiate its own packet without having to wait indefinitely for a previously generated gather packet. In such a scenario, a large value

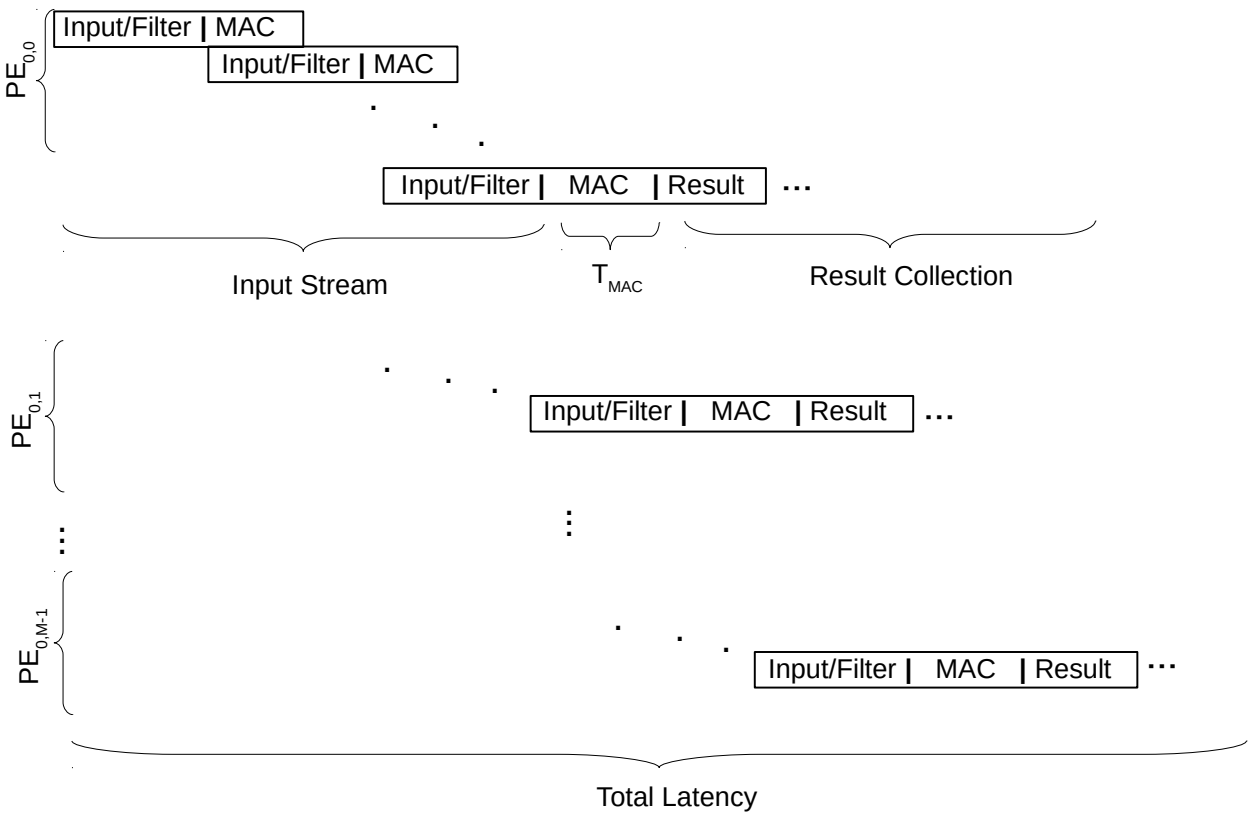


Figure 3.5: Pipelined operation of convolution on a row of PEs

of δ can lead to higher packet latency. In our experiments, all links are assumed to be fault free and reliable.

It is important to restrict a circular path in the routing algorithm to avoid a potential deadlock. The proposed gather packet still follows XY routing, which is deadlock-free.

3.2.3 Analysis of Performance Improvement

The advantage of the gather supported routing is the savings in network latency and power consumption that are obtained using the gather packet to collect data that were otherwise going to be sent using unicast traffic. The gather packet reduces the hop count needed to deliver the data to the memory element by allowing the data from the intermediate nodes

to piggyback its payload into the incoming gather packet.

The latency to finish one round of convolution is attributed to the input streaming time, the MAC computation time, and the result collection time. Assume that wormhole switching is used and the operations of input streaming/MAC/result collection are performed in a fully pipelined fashion as shown in Fig. 3.5. Equations (3.2) and (3.3) estimate the latency in clock cycles using the repetitive unicast (RU) method and the proposed gather method (G) respectively. Equation (3.4) shows the expected performance improvement using the proposed gather method over the repetitive unicast.

In these equations, $C \cdot R \cdot R$ represents the time to stream the inputs to the PE, T_{MAC} represents the computation time for the MAC operation, κ represents the number of pipeline stages at each router (each stage occupying one cycle), $\frac{P}{N} \cdot \frac{Q}{M}$ represents the multiple rounds for all the P inputs and Q filters. Assume that each unicast packet size is L, each gather packet size is L', and the flit size is W. The gather packet is initiated from the leftmost node of each row.

$$Latency_{RU} =$$

$$\left(C \cdot R \cdot R + T_{MAC} + M \cdot \kappa + \left\lceil \frac{L}{W} \right\rceil - 1 + (M-1) \cdot \left\lceil \frac{L}{W} \right\rceil + \Delta_R \right) \frac{P}{N} \cdot \frac{Q}{M}$$

$$=$$

$$\left(C \cdot R \cdot R + T_{MAC} + M\left(\kappa + \left\lceil \frac{L}{W} \right\rceil\right) - 1 + \Delta_R \right) \frac{P}{N} \cdot \frac{Q}{M} \tag{3.2}$$

where $M \cdot \kappa$ represents the latency for the header flit from $PE_{0,0}$ to reach to the global buffer,

$\left\lceil \frac{L}{W} \right\rceil - 1$ represents the remaining flits from $PE_{0,0}$ to reach the global buffer, $(M-1) \cdot \left\lceil \frac{L}{W} \right\rceil$ represents the latency for the remaining packets to reach to the global buffer, and Δ_R is the latency added due to the congestion.

$$Latency_G = \left(C \cdot R \cdot R + T_{MAC} + \sum_{i=0}^{\left\lceil \frac{M}{\eta} \right\rceil - 1} \left((M - i \cdot \eta) \cdot \kappa + \left\lceil \frac{L'}{W} \right\rceil - 1 + t_\delta + \Delta_G\right)\right) \frac{P}{N} \cdot \frac{Q}{M} \tag{3.3}$$

where η is the number of payloads that can be collected by one gather packet, $\left\lceil \frac{M}{\eta} \right\rceil$ represents the number of gather packet, $(M - i \cdot \eta) \cdot \kappa$ represents the latency for the header flit in the gather packet, $\left\lceil \frac{L'}{W} \right\rceil - 1$ represents the latency for the rest of the flit in the gather packet, $t_\delta \in \{0, 1, .., \delta\}$ is the latency added due to the $delta(\delta)$ cycle which will vary depending on the availability of gather packet when the gather payload is ready, and Δ_G is the latency added due to the congestion.

$$Improvement = \frac{\left(M\left(\kappa + \left\lceil \frac{L}{W} \right\rceil\right) - 1 + \Delta_R\right) - \left(\sum_{i=0}^{\left\lceil \frac{M}{\eta} \right\rceil - 1} \left((M - i \cdot \eta) \cdot \kappa + \left\lceil \frac{L'}{W} \right\rceil - 1 + t_\delta + \Delta_G\right)\right)}{C \cdot R \cdot R + T_{MAC} + \sum_{i=0}^{\left\lceil \frac{M}{\eta} \right\rceil - 1} \left((M - i \cdot \eta) \cdot \kappa + \left\lceil \frac{L'}{W} \right\rceil - 1 + t_\delta + \Delta_G\right)} \tag{3.4}$$

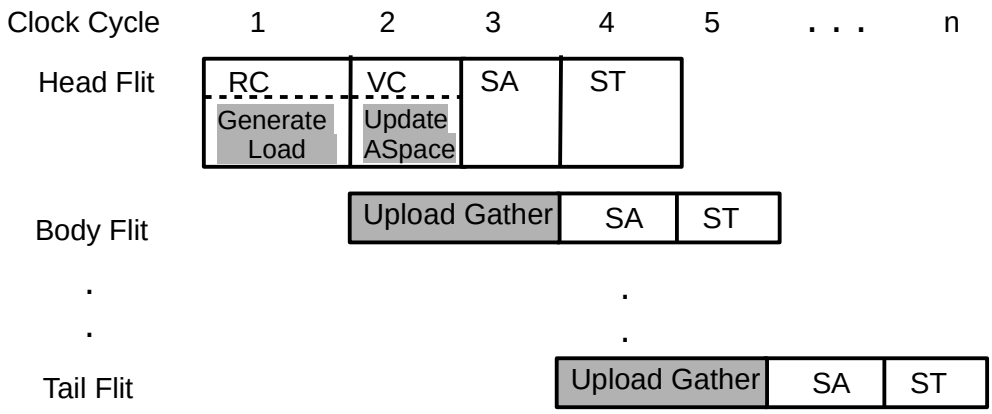


Figure 3.6: Modified router pipeline

3.3 Router Architecture

An NoC router typically consists of multiple pipeline stages. Fig. 3.6 shows a four-stage router pipeline including stages of route computation (RC), virtual channel allocation (VC), switch allocation (SA), and switch traversal (ST) [22]. For an incoming packet, only the header flit undergoes the RC stage to determine the output port for the packet. Similarly, only the header flit moves to the VC stage, where the flit arbitrates for a virtual channel corresponding to its output port. In the SA stage, each flit arbitrates for the switch input and output ports. Finally, in the ST stage, each flit traverses the crossbar. All flits of a packet undergo the SA and ST stages. The unused pipeline stages for the body/tail flit can be used to fill the gather payload into a gather packet.

Fig. 3.6 shows the modified router pipeline to incorporate the gather support. After the header flit of a gather packet arrives at the input buffer, the *Load* signal is generated during the RC stage and in the VC stage *ASpace* counter is updated. Upon the arrival of the body/tail flit, the gather payload is filled into the packet during the RC and VC stages. This modification does not require the packet to leave the router, nor add extra pipeline

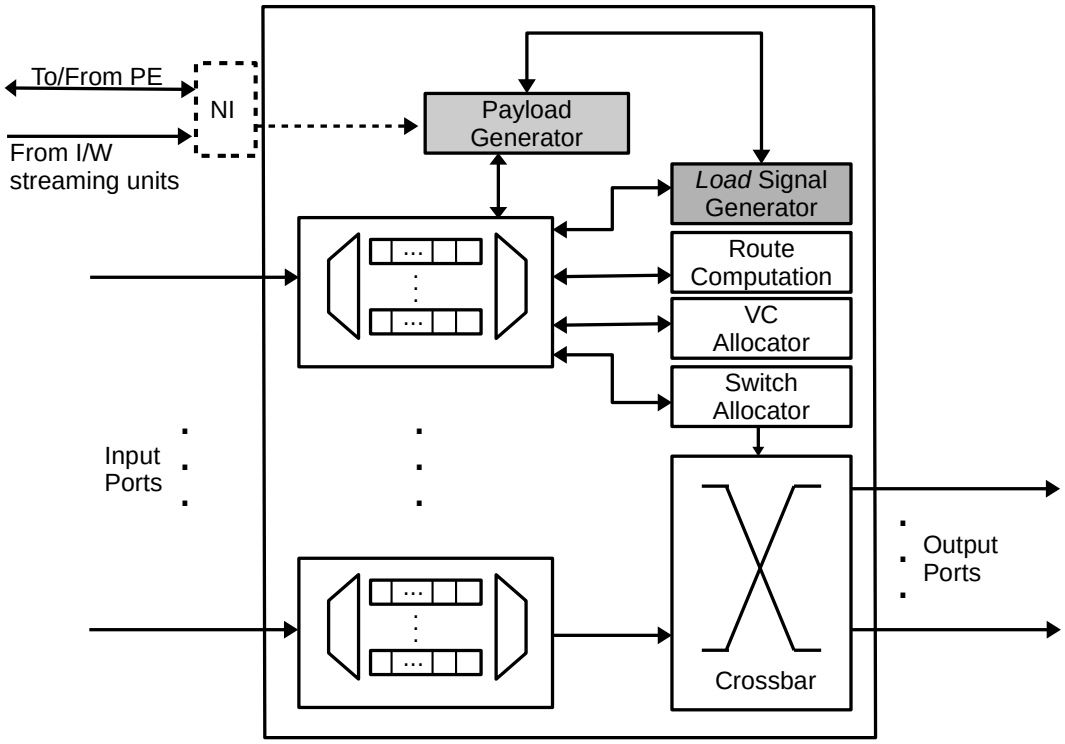


Figure 3.7: Router microarchitecture.

stages; thus, no additional latency is introduced. As the router pipeline does not change, there is no impact on the router performance.

The modified router microarchitecture is shown in Fig. 3.7. The *Load* Signal Generator block, shown in Fig. 3.4(a), has access to all the incoming input ports and the gather payload generated from the PEs. A matching logic is used to identify certain conditions to generate the load signal as described in Algorithm 1. The Payload Generator block as shown in Fig. 3.4(b) uses the load signal along with the decoded input port number to forward the payload to the respective port which then updates the *ASpace* and uploads the gather payload to the subsequent incoming body or tail flit. The Payload Generator block is also responsible for sending the ACK signal back to the PE. Depending on the ACK signal, the PE will either initiate its own gather packet if the incoming gather packet is full, or initiate its own unicast

Table 3.1: Network configuration for many-to-one simulation

Topology	8x8 Mesh, 16x16 Mesh
Virtual Channels	4
Router Pipeline Stage	5
Buffer Depth	4 flits
Packet Size	Gather : 4 flits/packet Other: 2 flits/packet
Flit Size	98 bits/flit
Gather Payload	32 bits
T_{MAC}	5 Clock Cycles

packet upon the expiration of δ cycles controlled by a counter set by the PE.

3.4 Performance Evaluation

This section evaluates the gather supported routing method and compare it with the repetitive unicast method using the convolution layers in AlexNet [7] and VGG-16 [8].

3.4.1 Simulation Settings

A cycle accurate C++ based NoC simulator [46] is used to simulate the proposed method implemented with the OS systolic array [30] on mesh-based NoCs as shown in Fig. 3.2(a). Table 3.1 lists the NoC settings. Pytorch [47] framework is used to generate the parameters for AlexNet [7] and VGG-16 [8] shown in Table 1.1. These parameters are used to generate the traces for the experiment. Orion 3.0 [48] is used to estimate the dynamic power consumption. *Delta*(δ) is set to the 5 *clock cycles* to ensure at least the head flit will arrive at the neighboring node. Assume that the global buffer is connected to the right edge of the mesh. As shown in Fig. 3.2(a), the input feature maps and filter weights are streamed from

Table 3.2: Estimated vs simulated performance improvement in total latency for Alexnet [7] in 8x8 mesh for gather supported routing

Result \ Layers	Conv1	Conv2	Conv3	Conv4	Conv5
Estimated	2.92	0.73	0.68	0.34	0.51
Simulated	5.93	1.37	1.27	0.63	0.95

the input and weight buffers from the left and top side of the network respectively. A row-based gather is simulated where the gather packet is initiated from the PEs in the leftmost column to the rightmost PEs and then to the global buffer.

3.4.2 Result

Fig. 3.8 shows the performance improvement in the total latency (including the input streaming time, MAC time, and result collection time) of all five convolution layers in AlexNet [7] using the proposed gather method over the repetitive unicast method. Table 3.2 shows that the changing trend of both results is consistent. For the estimated result in Table 3.2,the parameters Δ_G, Δ_R, and t_δ are all set to 0, which reflects the ideal case with no congestion and δ delay. Noticeably, the estimated result represents the least improvement that can be achieved. Nevertheless, in simulations, there exists congestion and delta delays, i.e., Δ_R, Δ_G, and t_δ are not 0. That's why the simulated improvement is higher than the estimated result from (3.4).

It's clear that 16×16 mesh offers better improvement than 8×8 mesh as the saving in hop count achieved by the gather method in 16×16 mesh is more significant than that in 8×8 mesh. In addition, the congestion will start to degrade the performance of the

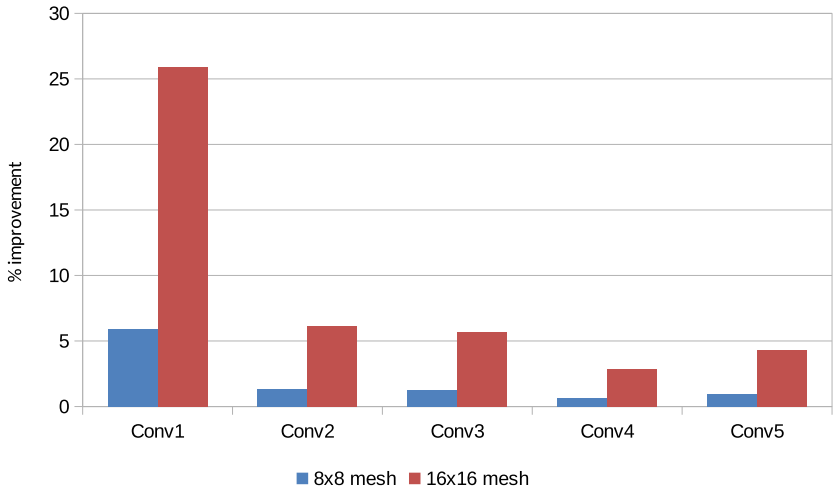


Figure 3.8: Improvement in total latency for AlexNet [7] on 8×8 and 16×16 mesh

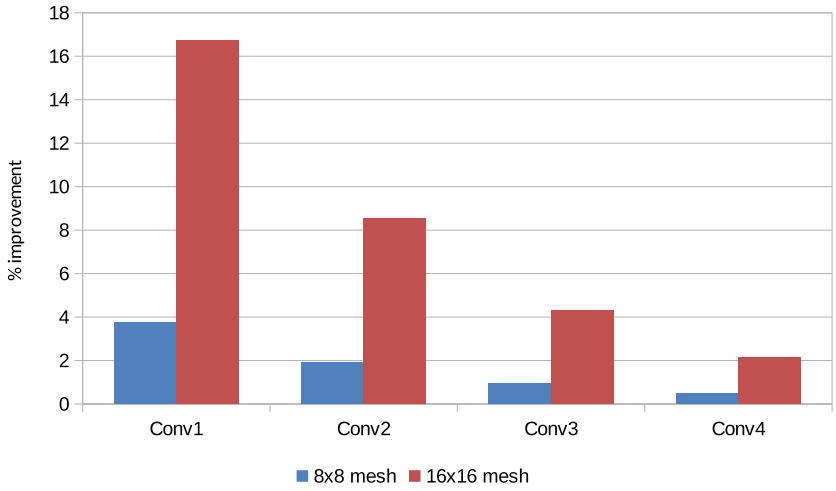


Figure 3.9: Improvement in total latency for VGG-16 [8] on 8×8 and 16×16 mesh

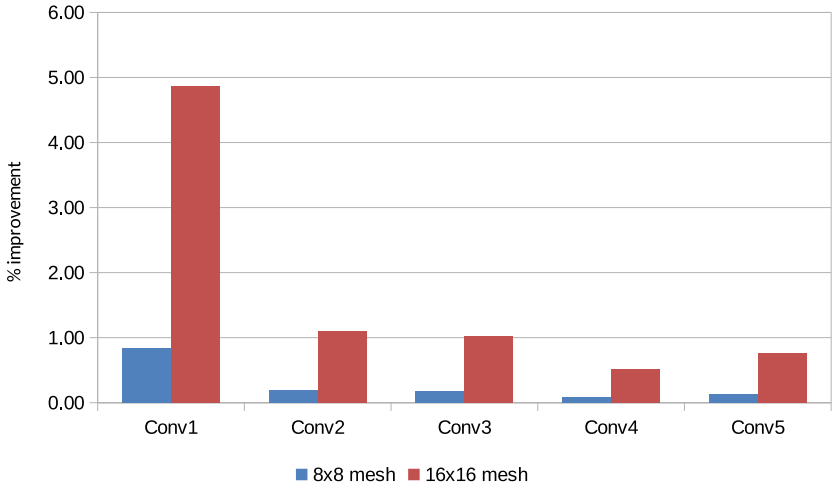


Figure 3.10: Improvement in power for AlexNet [7] on 8×8 and 16×16 mesh

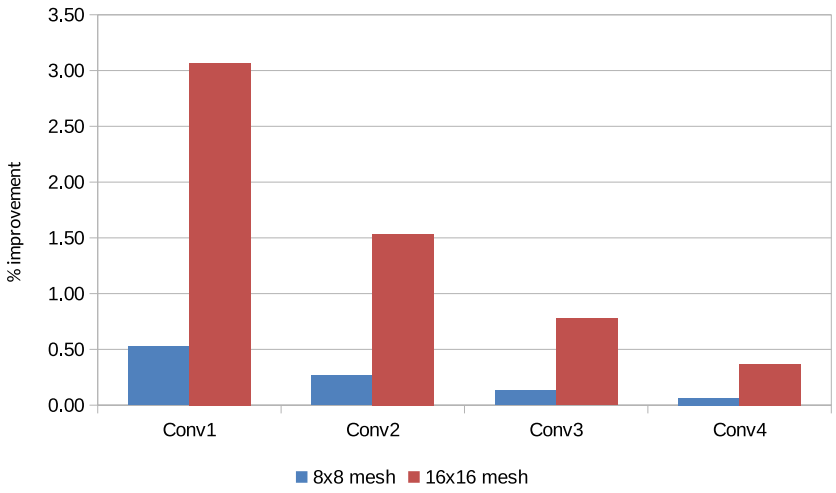


Figure 3.11: Improvement in power for VGG-16 [8] on 8×8 and 16×16 mesh

repetitive unicast with the increase in the network size. The first convolution layer (Conv1) shows the highest improvement in both meshes because it has the smallest $C \cdot R \cdot R$ value (refer to Table 1.1 and (3.4)). Similarly, Fig. 3.9 shows the performance improvement in the total network latency for the selected 4 out of 13 convolution layers 2, 4, 6, 13 with different parameters in VGG-16 [8]. Similarly, Conv1 (for layer 2) which has the smallest $C \cdot R \cdot R$ value has the best improvement compared with other CONV layers.

Fig. 3.10 shows the performance improvement in the total power consumption of all the convolution layers in AlexNet [7] using the proposed gather method over the repetitive unicast method. Although for 8×8 mesh less than 1% improvement is seen for all the convolution layers, this improvement is better with the increased size of the network. For 16×16 mesh, the total improvement in all the convolution layers is around 8%. Similarly, Fig. 3.11 shows the performance improvement of power in VGG-16 [8]. We can see a similar trend, with an increase in the network size the performance improvement is getting better.

3.5 Summary

This chapter presents the analysis of using the gather packet on mesh-based NoCs to support the significant amount of many-to-one traffic existing in a CNN workload. Particularly during the convolution layer, continuous streams of input and weights are fed to PEs and the MAC results are sent back from each PE to the output buffer before starting the new round of computation. The simulation results of the gather method using the traffic traces generated from the convolution layers of AlexNet and VGG-16 are compared against the repetitive unicast method on both 8×8 and 16×16 mesh-based NoCs. AlexNet [7] and VGG-16 [8] both show the improvement in the total runtime and power consumption for 8×8 and 16×16 mesh-based NoCs employing output stationary systolic array. This gather method can also be used to support gather type traffic in other DNN workloads.

CHAPTER 4

SUPPORTING ONE-TO-MANY TRAFFIC

This chapter presents the support to multicast traffic for DNN workload i.e., one-to-many traffic. Section 4.1 explains the motivation behind this study. Section 4.2 presents the modified streaming bus support for multicast traffic. Section 4.3 presents the result of the performance evaluation and finally, Section 4.4 summarizes this chapter.

4.1 Motivation

As shown in Fig. 3.2, in the OS model, a partial sum will be accumulated at a PE with the input activations and weights streaming from the memory element [28] [49]. Assume that one router can connect to up to N PEs so that these PEs will receive N sets of inputs and weights. Effective data re-utilization can only happen if the distribution is performed effectively.

Fig. 4.1 shows the traffic distribution in different layers in AlexNet [7] and VGG-16 [8]. The majority of the traffic is concentrated in the CONV and FC layers. Other traffic involved in the maxpool layer is less compared to the CONV and FC layers combined. In both DNNs for 1PE case, we can see the amount of traffic for CONV and FC layer is more than 90%. This traffic includes the distribution of weights and inputs from memory to PEs to perform MAC operations. Notice that, as the number of PEs per router increases, the CONV/FC layer traffic is distributed among those PEs with less overhead which reduces the

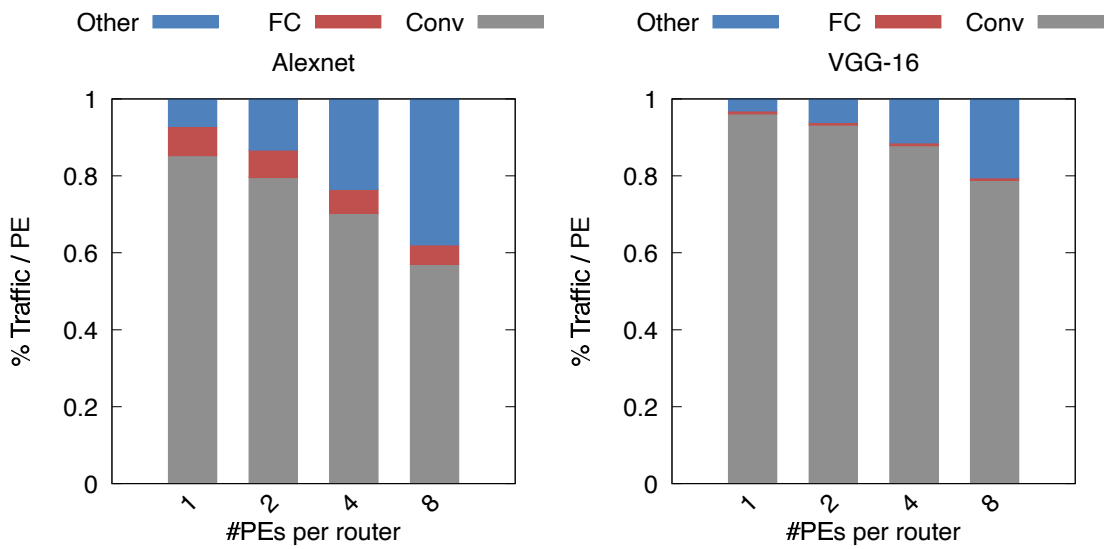


Figure 4.1: % Traffic/PE in AlexNet[7] and VGG-16[8]

traffic in these layers. However, other traffic which involves the inter PE sync and command from the task scheduler increases due to extra PEs. The major advantage of having a spatial neural network accelerator is to reuse the data and to avoid the duplication of the data. Multicast communication i.e., one-to-many is one of the major traffic in the CONV and FC layer which needs special attention and handling in the DNN accelerators.

Multicast routing algorithms are broadly classified into tree-based and path-based. In the literature, several tree-based and path-based multicast routing algorithms have been proposed for NoCs [19][50]-[53]. The multiple unicast routing can be considered as a simple tree-based algorithm that has an advantage when the multicast destination is sparsely located. In path-based multicast algorithms, intermediate branching is prohibited. Multiple paths at the source node are identified and the packet is delivered to the destination nodes along the paths. However, the multicast requirement for the DNN workloads vary from the assumption that traditional single/multithreaded workload like PARSEC [23] in

the following ways:

- Path-based multicast algorithm like DPM [19] follows a certain path from the source node to all the destination nodes, this way of data delivery adds extra latency to the packet delivered on each destination node. In the DNN accelerators, most of the PEs work on the same weights or input activation, hence existing multicast algorithm is inefficient in DNN accelerators.

- Data reuse is the key feature of DNN accelerators which rely on data reception and data delivery. Due to the variable data reception latency, data delivery will also vary. This variation results in the late initiation of the subsequent layers since the processing of the next layer in DNNs also depends on the completion of the previous layer.

4.2 Streaming Architecture

Based on this observation, a modification is proposed where the input activations and weights are streamed using a bus from the memory elements to the PEs in the same row and column, respectively. The streaming bus will help in overcoming the additional routing overhead and thus improve the runtime latency and power consumption of a DNN workload. When multiple PEs are supported by one router, the communication bandwidth can be further improved using the streaming bus.

Fig. 4.2(a) shows the two-way streaming architecture, where two different stream units will handle the streaming of input activations and filter weights. Each input activation streaming unit handles the streaming of the corresponding input activation from the memory element to a respective PE row. Each router in the same row will receive the same input

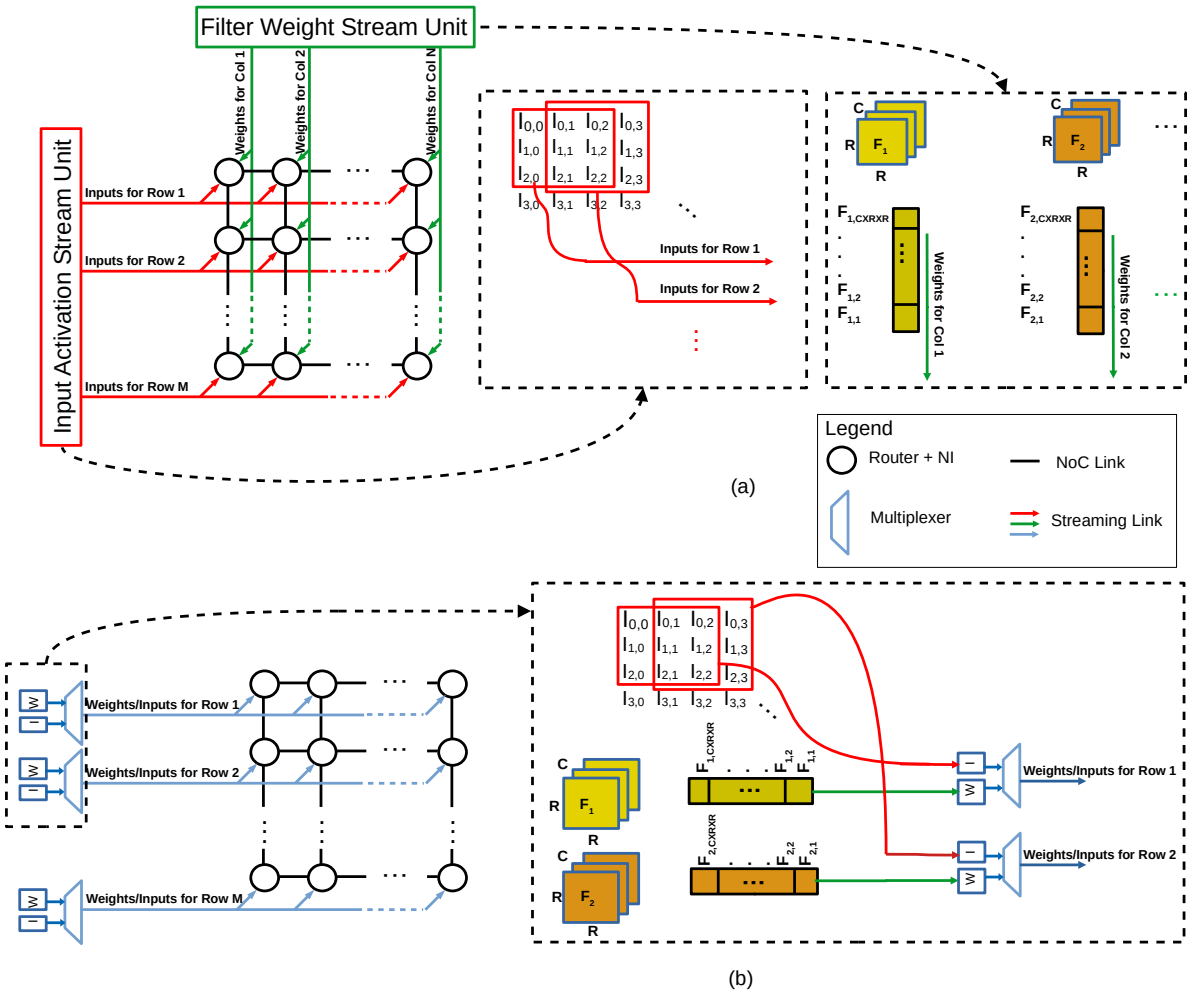


Figure 4.2: Modified architecture with direct streams (a) two-way streaming, (b) one-way streaming

activations, which are then buffered for MAC operation on a PE's internal register file. Similarly, the weight streaming unit streams the filter weights from the memory element to a respective PE column. These streaming units will stream all the row and column data to the respective PEs, similar to the pattern shown in Fig. 3.2. The partial sums or the output activations are calculated at all PEs. Results in the same row are then collected using a gather packet as it proceeds towards the memory element. Other dataflow models can also perform a similar data orchestration.

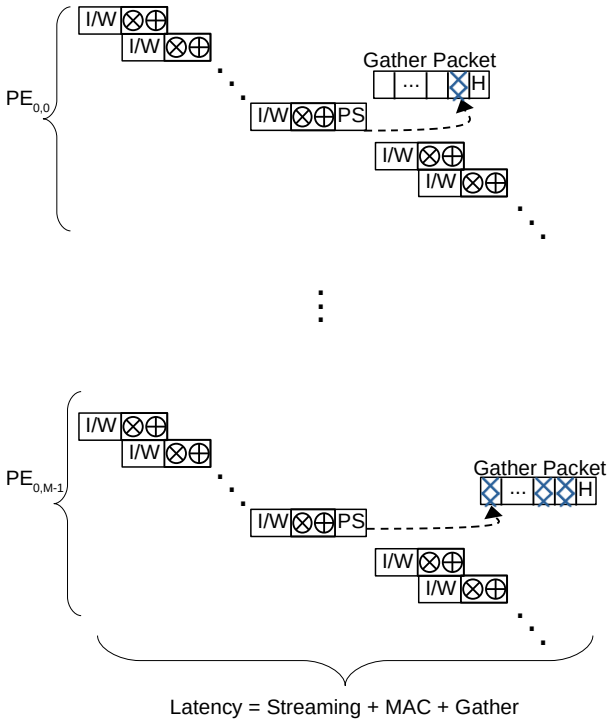


Figure 4.3: Pipelined operation of a partial sum(PS) generation/gather in a row of PEs

Fig. 4.2(b) shows the one-way streaming architecture, where both the input activations and the filter weights share the same streaming link to PEs in the same row. As either weight or the activation streams share the link at a given clock cycle, there is a latency added before the PEs can move ahead with the MAC operation. Fig. 4.2(b) also shows the streaming unit, which streams the input/weight activation in an interleaved manner through a multiplexor on a shared link. A gather packet will accumulate the partial sums before sending them back to the memory element. This architecture will use less silicon area compared with the two-way streaming architecture. This architecture may be beneficial for other types of dataflow models like WS, RS, where the weights are streamed first to the PEs before input streaming begins or where there can be a tradeoff between the area and power.

4.2.1 Analysis of Streaming Bus

The total clock cycles required to finish one round of convolution operation can be attributed to the time required for: input activation streaming and weight streaming; the MAC operation; and finally, the generation and collection of the result. Fig. 4.3 illustrates the pipeline execution of multiple rounds of convolution operations in one row of PEs with one PE/router, where the streaming of input activation and weights (I/W), followed by the MAC operation and activation function, happen in parallel at all the PEs. After $C \cdot R \cdot R$ MAC operations, the partial sum (PS) is generated at each PE, which is then collected by the gather packet. While the gather packet collects the PS results along its way to the global memory, the next round of convolution operation occurs concurrently. With multiple PEs/router, in each round the streaming time will be extended while other parts stay the same.

Equations (4.1)-(4.2) analyze the improvement of the runtime latency of a convolution layer using gather support over repetitive unicast (baseline) for the OS dataflow model on the proposed streaming architecture. In these equations, $C \cdot R \cdot R$ represents the time to stream the inputs to the PE, as shown in Fig. 3.2; n represents the multiplying factor, which depends on the number of PEs per router; f_l represents the factor that reduces the input streaming with the streaming bus in the proposed method; T_{MAC} represents the computation time for the MAC operation; κ represents the number of pipeline stages at each router (with each stage occupying one cycle); and $\frac{P}{N} \cdot \frac{Q}{M} \cdot \frac{1}{n}$ represents the number of rounds needed to finish the convolution of all P inputs and Q filters using the OS dataflow model. Each unicast packet size is L, each gather packet size is L', and the flit size is W. The gather packet is initiated from the leftmost node of each row in Fig. 3.2.

$$Latency_{_{RU}} =$$

$$\left(\frac{C{\cdot}R{\cdot}R{\cdot}n}{f_l} + T_{MAC}\right)\frac{P}{N} \cdot \frac{Q}{M} \cdot \frac{1}{n} + M \cdot \kappa + \left\lceil \frac{L}{W} \right\rceil - 1 + \Delta_R \tag{4.1}$$

Equation (4.1) derives the runtime latency of a convolution layer using repetitive unicast, where $M \cdot \kappa$ represents the latency for the header flit of the result packet (partial sum) from $PE_{0,0}$ to reach the global buffer, $\left\lceil \frac{L}{W} \right\rceil - 1$ represents the time needed for the remaining flits to arrive at the global memory, and Δ_R is the latency added due to the congestion. When a data streaming bus is used, as the transmission of unicast packets, all nodes are parallelized, the packet from the leftmost node will take the longest time to arrive at the global memory.

$$Latency_{_{G}} =$$

$$\left(\frac{C{\cdot}R{\cdot}R{\cdot}n}{f_l} + T_{MAC}\right)\frac{P}{N} \cdot \frac{Q}{M} \cdot \frac{1}{n} + \sum_{i=0}^{\left\lceil \frac{M.n}{\eta} \right\rceil - 1} \left((M - i \cdot \frac{\eta}{n}) \cdot \kappa + \left\lceil \frac{L'}{W} \right\rceil - 1\right) + \Delta_G \tag{4.2}$$

Equation (4.2) derives the runtime latency of a convolution layer using gather support, where η is the number of payloads that can be collected by one gather packet,, $\left\lceil \frac{M.n}{\eta} \right\rceil$ represents the number of gather packets, $(M - i \cdot \frac{\eta}{n}) \cdot \kappa$ represents the latency for the header flit in the gather packet, $\left\lceil \frac{L'}{W} \right\rceil - 1$ represents the latency for the rest flits in the gather packet, and Δ_G is the latency added due to the congestion.

Note that in Eqs. 4.1 and 4.2, the data streaming and MAC operation time is same; the difference lies in the time taken to transmit the results to the global memory. When n=1, the time taken to transmit the unicast packet from the leftmost node is nearly the same as

the time taken to transmit the gather packet. However, when n increases, the delay due to network congestion will increase significantly for RU (reflected by Δ_R). In comparison, the network congestion for gather packets (reflected by Δ_G) will be much less.

4.3 Performance Evaluation

Fig. 4.4 shows the simulated runtime latency improvement of the proposed gather support with two-way streaming (Fig. 4.2(a)) and one-way streaming(Fig. 4.2(b)) architectures vs. gather-only using the NoC parameter from Table 3.1. On average, the gather support with two-way streaming architecture achieves 1.71 times improvement, and the gather support with one-way streaming obtains 1.48 times improvement compared against the gather-only method [41]. Both the one-way and two-way streaming shows similar performance trends on an individual CONV layers. Also we can see the two-way streaming outperforms the one-way streaming for the OS dataflow model on which the experiments were conducted. This improvement of two-way streaming is due to the availablity of the input and weight data at the PE when required. Both of them are streamed from the streaming unit making these data available sooner than in one-way streaming where the streaming bus is multiplexed.

4.4 Summary

The traditional multicast algorithm performs well in multi/single-threaded many-core architectures because the succeeding threads may not be dependent on the preceding ones as frequently and we can expect a lot of out-of-orderness to exploit. This is not in the case of DNN workloads and hence these algorithms will not serve the need of DNN accelerators.

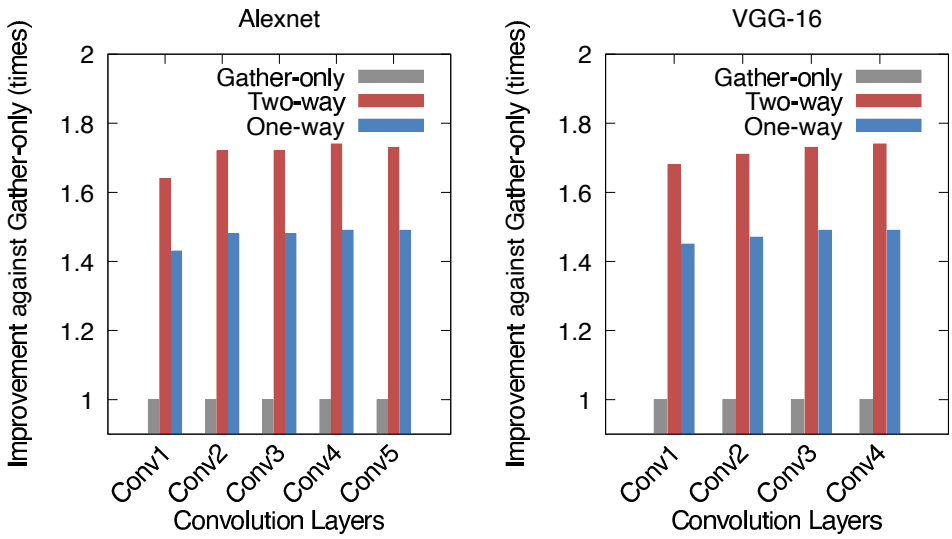


Figure 4.4: Simulated improvement on the runtime latency of different convolution layers in Alexnet [7] and VGG-16 [8] over Gather-only [41]

This dissertation proposes a streaming bus architecture to cater to the need of the DNN accelerators. It is clear that the runtime latency improvement using two-way streaming is better than using one-way streaming in the OS dataflow model. However, one-way streaming also outperforms the baseline model considerably. There is a trade-off in the use case for one-way and two-way streaming, and depending on the application requirement we can switch one for the other. Since one-way streaming uses one bus, it occupies less silicon area compared to two-way streaming. One-way consumes less area however, it suffers degradation in the latency which is around 13% less than the two-way streaming.

CHAPTER 5

SUPPORTING MULTIPLE PEs PER ROUTER

This chapter presents an architecture that supports multiple PEs in a DNN accelerator. Section 5.1 explains the motivation behind this study. Section 5.2 presents the router architecture to support multiple PEs per node. Section 5.3 presents the analysis of the gather parameter δ and the size of the gather packet for the number of PEs per node. Section 5.4 presents the result of the performance evaluation and finally, Section 5.5 summarizes this chapter.

5.1 Motivation

A DNN accelerator would require having enough computation units to sustain the computation bandwidth of the DNN layers at the same time not requiring too much of the overhead in the inter-PE synchronization and memory per PE. Fig. 1.4 makes it abundantly clear that MAC operation is the major computation in a DNN execution. The accelerator that helps in improving this computation is going to accelerate the DNN workloads.

The PE architecture in the DNN accelerator needs to be simple and lightweight. This helps in various other issues which can arise due to a complex PE which is not desirable. A simple PE consists of a local Register File (RF), ALU, and a control unit which can perform MAC, average, comparison, etc in each cycle. Fig. 5.1 shows a simple low complexity FSM control unit for a PE that handles the task of fetching the data from the local interconnection

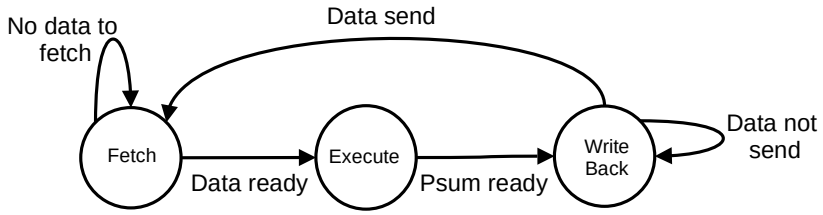


Figure 5.1: PE control FSM

port or an RF, executing on the data, and finally, writing back the result to either an RF or a local interconnection port. The lightweight design of PEs makes it possible for the DNN accelerator to scale the number of PEs to gain computation bandwidth. Due to the flexibility in having a simple PE, DNN accelerators are able to scale these PE to gain computation bandwidth. Furthermore, these PE can be clock gated when not in use or underused to save energy consumption as done in [30] which saved 45% using this technique.

Moreover, Chapter 2 also describes the importance of spatial architecture and its advantage in enabling data reuse via various dataflow models. Effective communication via an efficient dataflow pattern will help accelerate DNN execution. When the number of PEs increases, the network interface architecture need to be redesigned to improve the communication efficiency i.e., maximum PE utilization and data reuse. This chapter focuses on the solution to improve the communication efficiency among the PEs to enable better data reuse which reduces the memory footprint in the DNN accelerator.

5.2 Multiple PEs per Router

This work also consider an expanded mesh, where multiple PEs connect to a router. Fig. 5.2 shows such an architecture. The Network Interface (NI) unit handles the packet movement between the router and PEs. The streaming units feed input(s)/weight(s) (I/W)

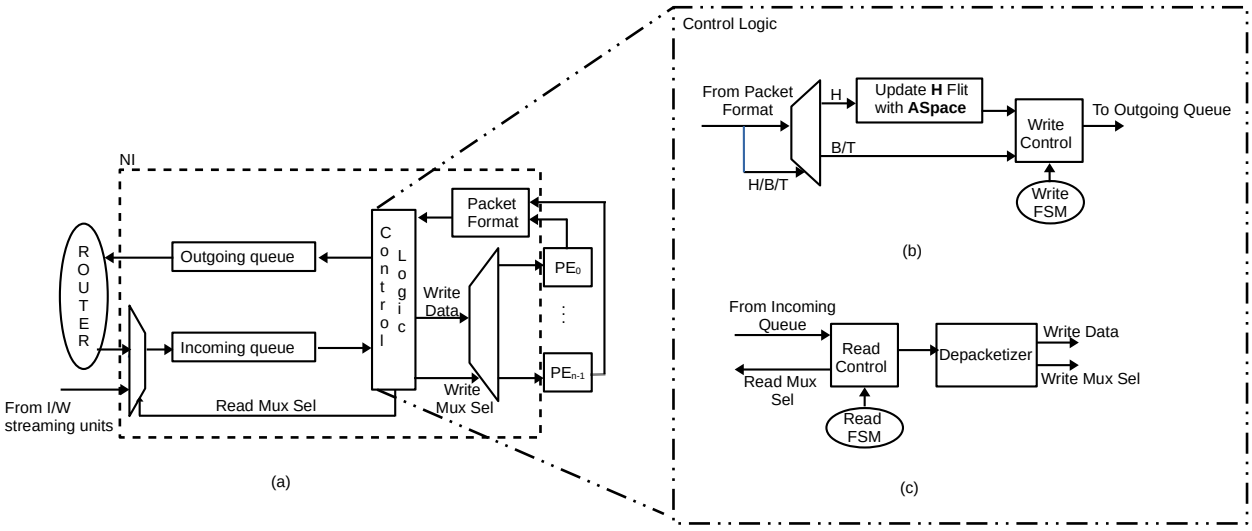
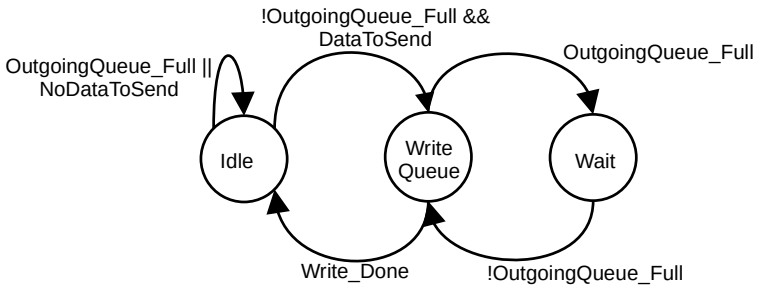


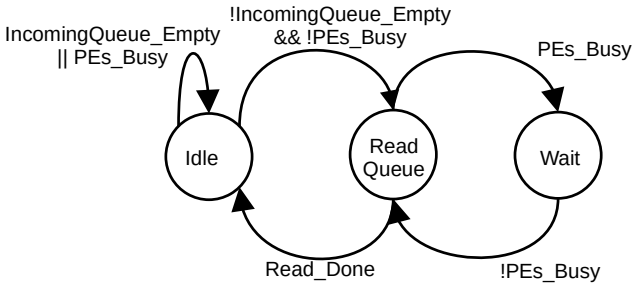
Figure 5.2: NI for multiple PE/router (a) network interface connection, (b) outgoing control logic, (c) incoming control logic

to the NI, as shown in Fig. 5.2(a). The NI dequeues an incoming packet from the router or streaming unit from the incoming queue into a control logic. The NI further disassembles the packet and forwards it to the respective PEs. The control logic keeps track of the type of packet, start and end of the packet, and other necessary information to correctly decompose the data for a respective PE. When PEs are ready to inject data into the network, the packet format unit will collect the outgoing data from the PEs and generate a packet. The generated packet is then forwarded to the control logic, which creates flits to be enqueued in an outgoing queue. The router will access the outgoing queue and inject the packet into the network. These PEs are simple, as proposed in [59], which supports MAC operation and an activation function with predictable pipeline stages. Hence, the synchronization requires no extra overhead.

The control logic in the NI as shown in Fig. 5.2(a) performs multiple functions to ensure the correct flow of operation. It consists of two parts: the outgoing control logic (Fig. 5.2b)



(a)



(b)

Figure 5.3: Outgoing and incoming control (a) outgoing queue write FSM, (b) incoming queue read FSM

which serves as the interface from the PEs to the router and the incoming control logic (Fig. 5.2(c)) which serves as the interface from the router to the PEs. The outgoing control logic encapsulates the incoming payload with information including FT, $ASpace$, etc. in the header (H) flit and controls the ordering of the flits before enqueuing them into the outgoing queue. The incoming control logic arbitrates the input to the incoming queue and extracts the incoming data before forwarding it to the respective PE. The Finite State Machine (FSM) shown in Fig. 5.3(a) explains how the write FSM in Fig. 5.2(b) orchestrates the flow in the outgoing queue. The state in the FSM represents the command of write control at a given time. The outgoing queue is only written when the queue has space available to ensure no packet from the PEs is lost. Fig. 5.3(b) demonstrates a similar FSM for read control

which ensures proper reading of the incoming queue. The state in the FSM represents the command of read control at a given time.

Supporting multiple PEs per router allows more partial sum generation in parallel and makes better utilization of a gather payload, which can help accelerate the DNN execution with reduced power consumption. Depending on the bus width, multiple input activations and weights can be streamed in each NI at one time. As shown in Fig. 3.2, these input activations and weights may have different combinations depending on how the PEs are grouped. One option is multiple PEs on the same column sharing one router; then multiple sets of input activation and one set of filter weight will be streamed in the NI. For example, for two PEs/router, $(I_{1,1}...I_{1,CXRXR})$, $(I_{2,1}...I_{2,CXRXR})$, and $(F_{1,1}...F_{1,CXRXR})$ will be streamed in the NI connected to $PE_{0,0}$ and $PE_{0,1}$ over multiple clock cycles. This can be further extended for 4 and 8 PEs/router. Another option is multiple PEs on the same row sharing one router; then one set of input activations and multiple sets of filter weights will be streamed in the NI. Other options are possible, with the cost of a more complex design at the control logic. The streaming units can receive this information as a configuration file at the beginning of the operation.

In the proposed method, there are two different networks: one for gather traffic and the other for a streaming bus. In the mesh network, a credit-based flow control mechanism [22] is used. The streaming bus can also use a similar end-end flow control mechanism, but this may create an extra wire overhead from each node to the streaming unit. Hence, a similar credit-based mechanism used in [44] to ensure the single-cycle data delivery to the PE is adopted. The global buffer maintains the status of the credits for the PEs, i.e., incoming queue in the NI, as shown in Fig. 5.2 (a). The streaming unit will only perform

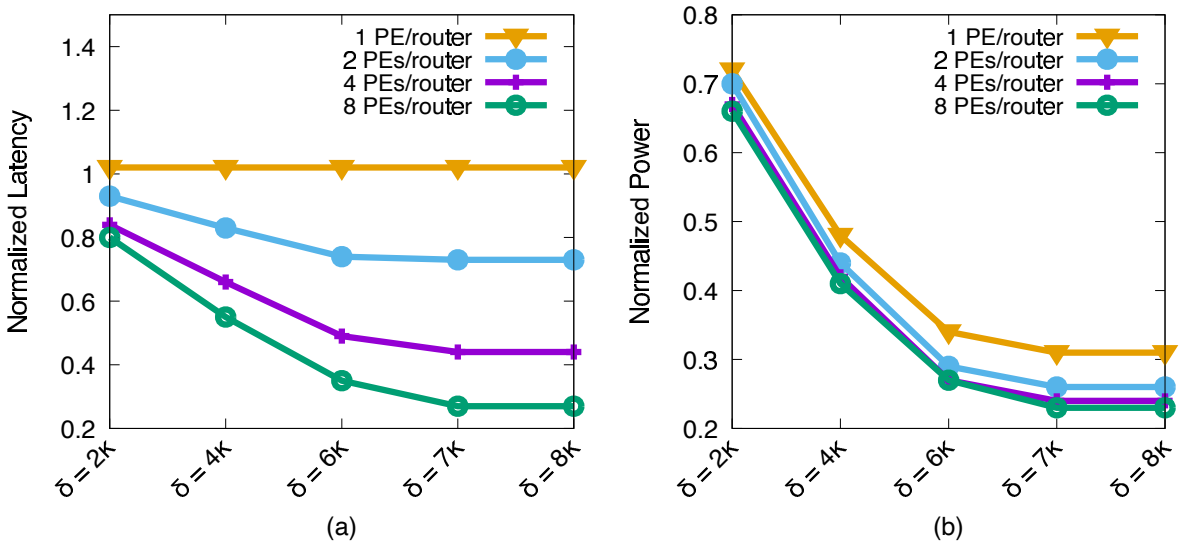


Figure 5.4: Analysis of δ on 8x8 mesh for different number of PEs/router

the streaming if all the nodes have free space to hold the data. This ensures the integrity of the MAC operation.

5.3 Analysis of Routing Parameters

The timeout period δ plays an important role in the performance of the gather supported routing. The δ value defines the waiting time (in cycles) for a PE with a gather payload to wait before initiating its own packet, with the anticipation that the gather packet sent from its neighbor will arrive. Fig. 5.4 shows the impact of δ on the total runtime latency, as well as the total power consumption of gather supported routing. The analysis is done on an 8x8 mesh under a similar traffic scenario as in Fig. 3.1, where the nodes in one row are trying to deliver the gather payload to the global memory on the right side of the mesh.

The time out period (δ clock cycles) actually depends on the router pipeline stages (κ). When $\delta < \kappa$, the header flit of a gather packet will not reach its adjacent node before

the expiration of the δ clock cycle. As such, each PE will initiate its own packet. This situation is similar to sending the result from each node in a repetitive unicast way, which will cause congestion in the network and result in increased runtime latency. As the value of δ increases, the gather packet initiated by a PE will subsequently reach the adjacent nodes before the timeout occurs at those nodes. This piggyback mechanism will effectively decrease the network traffic and improve resource utilization in the NoC.

As shown in Fig. 5.4(a), the normalized runtime latency (vs. $\delta < \kappa$) is reduced with δ value increased except for the case with 1PE/router, which is almost the same. With more PEs/router, the gather packet size is increased (3, 5, 9, 17 flits for 1, 2, 4, 8 PEs/router) to accommodate more partial sums. Noticeably, no further improvement is observed after δ becomes sufficiently large (7κ). This is because the δ value is large enough to allow all the gather payload to be collected by a gather packet. Therefore, for NXN mesh, δ is set to $(N-1)\kappa$ to ensure that the header flit of the leftmost gather packet will arrive at all nodes in the same row so that all the gather payloads can be uploaded into the same gather packet.

Fig. 5.4(b) shows the normalized power consumption for different values of δ (vs. $\delta < \kappa$). With the increase in δ value, the gather packets are able to collect all the partial sum results generated in the network and thus reduce the total number of packets that are generated in the network. This helps to reduce the total number of hops traversed and optimizes the NoC resource utilization for 1,2,4,8 PEs/router, thus consuming less total power. Although the runtime latency does not improve for the 1PE/router case, some significant improvement in the power consumption is observed.

This dissertation further studies the tradeoff between different gather packet sizes for different network sizes and different numbers of PEs/router. Fig. 5.5 compares the perfor-

mance of gather traffic using different numbers of gather packets for different numbers of PEs/router on 8x8 and 16x16 mesh. The ideal case is using one gather packet to collect all the gather payload, however, this setting can be expensive in terms of flits per packet for the OS dataflow model where reduction across the PEs is not performed. As the number of gather packets increases, with fewer flits per packet used, the performance tends to get close to the repetitive unicast method where eventually each node will have its own gather packet.

Fig. 5.5(a) and (b) show the normalized runtime packet latency and power consumption (vs. repetitive unicast) for 8x8 mesh. We can see a clear tradeoff, where using one gather packet with a larger number of flits is better in terms of latency improvement compared to using a higher number of gather packets. As the number of gather packets increases, the congestion in the network tends to increase. We observe a significant increase in power consumption for the case of 1 PE/router with 4 gather packets/row because the total payload size (256 bits) is much smaller than the total gather payload capacity (512 bits), excluding the header. However, for 2 to 4 PEs/router, using 2 or 4 gather packets is better at improving the power consumption than using 1 gather packet. A similar trend on 16x16 mesh is also shown in Fig. 5.5(c) and (d).

For the 1 PE/router case, a slight increase in runtime latency occurs, as this is the case in which the network does not have much load for the *RU* case, i.e., one packet per row. In addition, we notice that with smaller gather flit size, one can expect small runtime; however, it is the opposite, as shown in Figs. 5.5(a) and (c). The expiration of δ clock cycles causes this effect; for the 2 gather packets per row case, the second packet is only injected when the first packet reaches the node, with no space left for further payload. In such a case, the

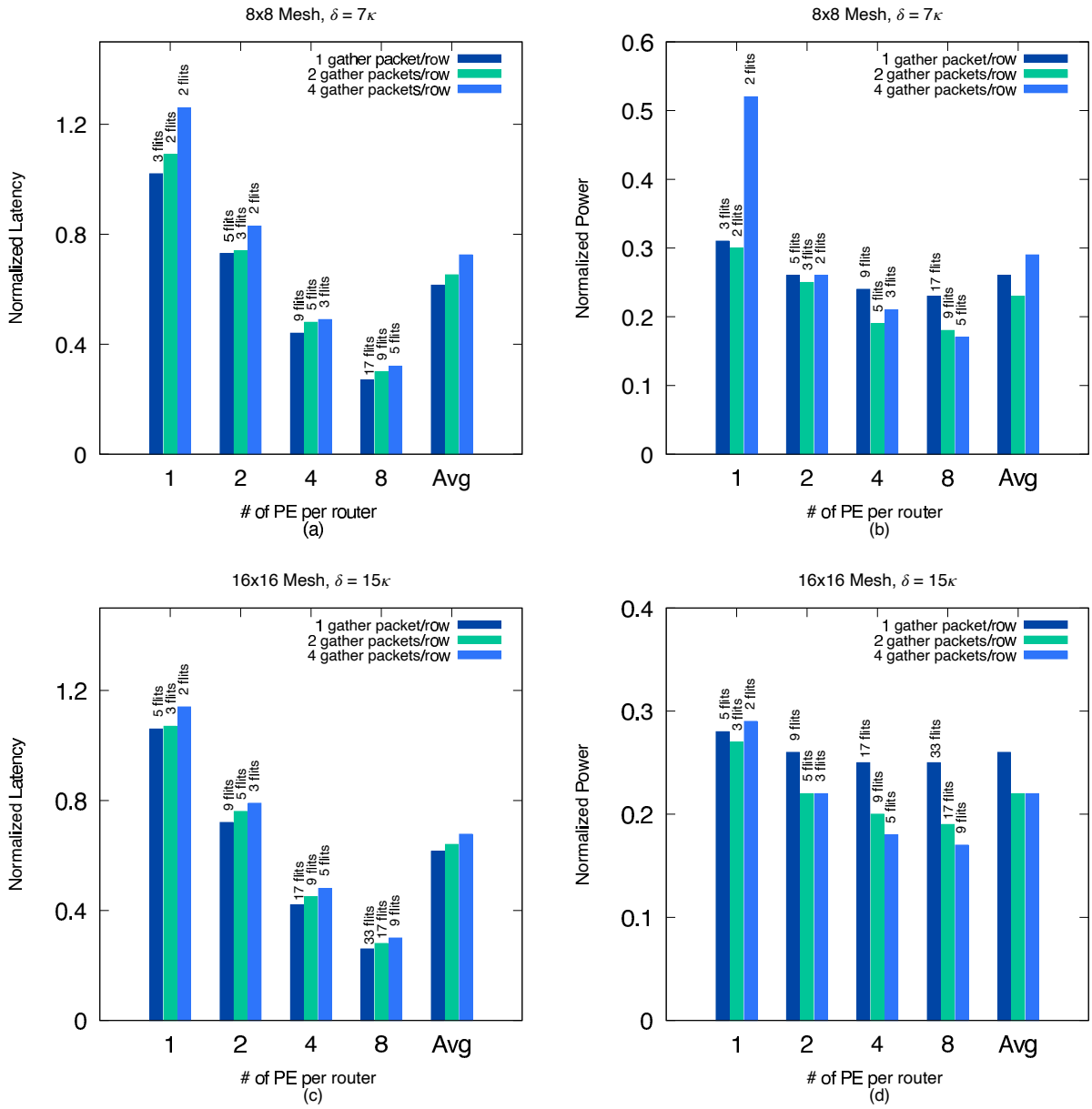


Figure 5.5: Analysis of different gather packet size on 8x8 mesh (a),(b) and 16x16 mesh (c),(d) for different number of PEs/router

first node to encounter such a situation will initiate a new gather packet, which is the second gather packet. To avoid this scenario, the router can be hardwired with the information to initiate its own gather packet without waiting for the incoming one, which reduces the scope and scalability of the proposed method. To balance the tradeoff of latency and power consumption, in the following performance analysis, one gather packet is used for 8x8 mesh and 2 gather packets are used for 16x16 mesh with 3, 5, 9, 17 flits/gather packet set for 1, 2, 4, 8 PEs/router, respectively. This result is taken based on the average performance over different numbers of PEs/router.

5.4 Performance Evaluation

To evaluate the performance of the proposed method, we ran simulations for different CNN workloads and compared them with the repetitive unicast method on mesh-based NoCs modified with the streaming architectures for the OS dataflow model. In this section, we describe the experiment settings, followed by presenting the performance analysis.

5.4.1 Experiment Setup

This work compare the proposed method and repetitive unicast method in terms of the runtime latency and power consumption. We assume that there is a higher level entity or a mapping framework similar to [29] [28] [39] [40] [60] that does the task of mapping neurons to the PEs, controlling timing for better synchronization without stalls, so that our focus is on evaluating the performance of the on-chip network. In order to fully utilize the spatial PE arrays, we use the parameters obtained from Pytorch framework [47] to model the traces for the NoC. The neurons are organized in a 2D mesh represented by the PEs. The total

Table 5.1: Network configuration for multiple PEs per router simulation

Topology	8x8 Mesh, 16x16 Mesh
Virtual Channels	2
Latency	router: 4 cycles, link: 1 cycle
Buffer Depth	4 flits
Flit Size	128 bits/flit
Gather Payload	32 bits
Number of PE per router	1,2,4,8
Gather Packet Size	3,5,9,17 flits/packet for 1,2,4,8 PEs/router resp.
Unicast Packet Size	2 flits/packet
T_{MAC}	5

neurons in each layer are divided to fit the PEs in a mesh. The memory elements (global memory) are located on the north, east, and west sides of the network. Each PE receives the input activation and filter weights from the streaming units on the north and east sides of the mesh. Accumulation happens locally to generate the partial sums, which are then collected from the left side to the global buffer at the right side of the mesh. The output feature map of the current layer is completely generated before moving ahead with another layer.

We have used a cycle-accurate C++ based NoC simulator [46] to simulate the generated traces for AlexNet [7], ResNet-50 [12], and VGG-16 [8]. The three different CNNs are chosen because of their diversity in terms of a number of parameters and the number of convolution layers. Orion 3.0 [48] is used to estimate the power consumption for NoC, and DSENT [61] to estimate the power consumption for the streaming bus. We have performed the simulations on 8x8 and 16x16 2D mesh networks. Table 3.1 shows the NoC setting used for performance analysis. As the number of PEs/router increases, the gather payload also increases. To

accommodate these gather payloads, we can either use a fixed number of flits or a dynamic flit size per gather packet. For the DNN workload, each node in the same row will generate a prefix sum result. Hence, a fixed number of flits is chosen for our experiments which avoids the extra overhead in router design compared to a dynamic flit size. The gather packet size is set as 3,5,9,17 flits/packet for 1,2,4,8 PEs/router, respectively. This flit size is enough to collect all the gather payloads for an 8x8 network; however, for a 16x16 NoC, two gather packets are needed, as the first one will be full halfway to the global memory.

5.4.2 Performance Analysis

Figs. 5.6(a) and (c) shows the improvement in the total runtime latency of the proposed method against the repetitive unicast method for all convolution layers in AlexNet [7] on 8x8 and 16x16 mesh-based NoCs. It is clear that as the number of PEs is increased across 8x8 or 16x16 mesh, we can see an improvement in the total runtime latency. This improvement is attributed to more parallel operations enabled by an increasing number of PEs per router. With more PEs, more MAC operations are done in parallel in one round, which reduces the number of rounds needed. For a lower number of PEs per router, the runtime improvement is minor as the network is not congested enough for the gather packet to improve the latency. The delta analysis from Fig. 5.4(a) also shows a similar effect.

The performance improvement is higher in the case of 16x16 mesh when compared with the 8x8 mesh. For the 16x16 mesh, repetitive unicast traffic creates much higher congestion in the network, and the benefit of using gather traffic is more significant than on the 8x8 mesh. Fig. 5.7(a) and (c) show the improvement in total runtime latency for all convolution layers in ResNet-50 [12] for 8x8 and 16x16 meshes. Similarly, Fig. 5.8(a) and (c) show

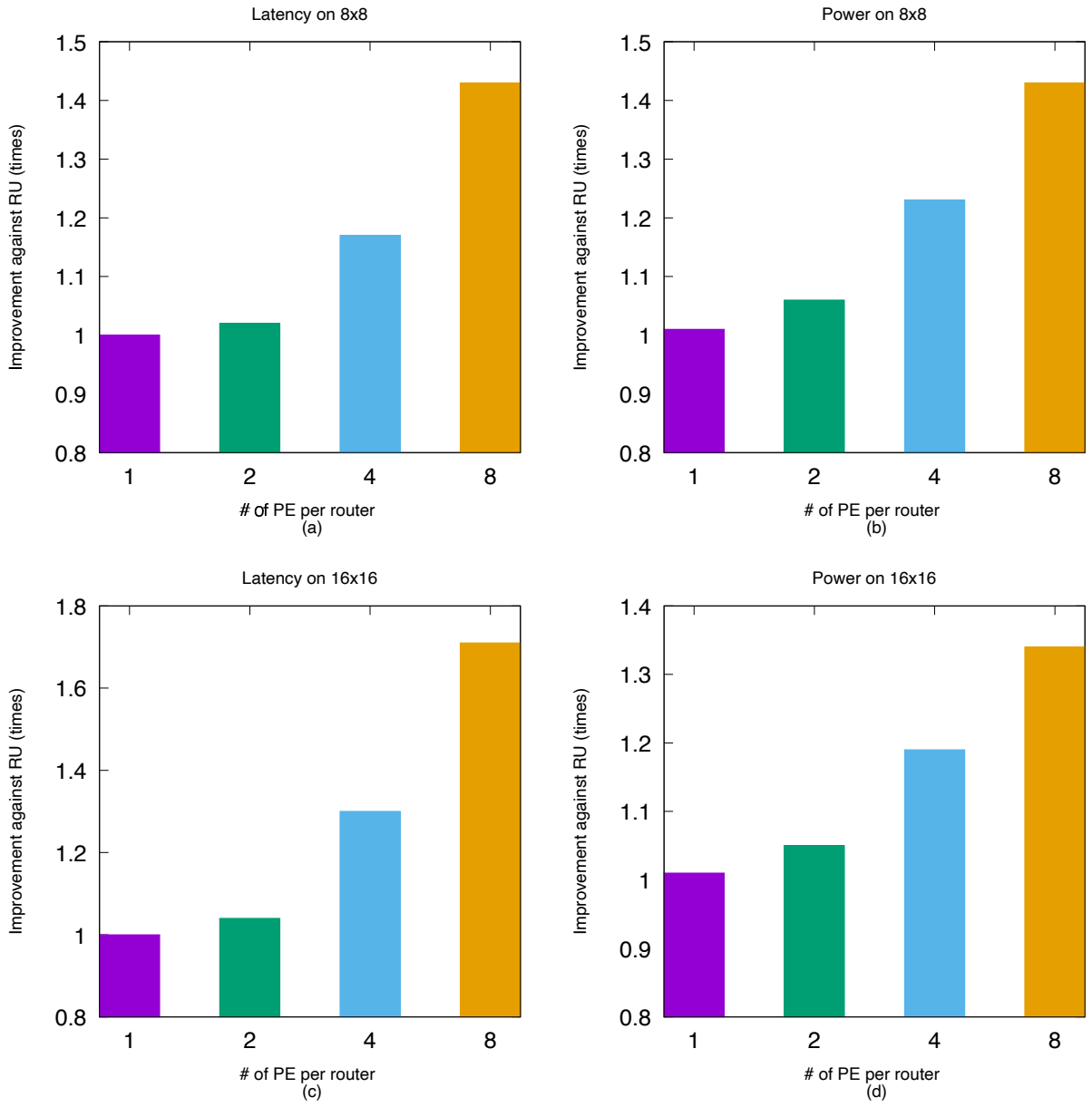


Figure 5.6: Improvement on total runtime latency (a),(c) and power consumption (b),(d) for AlexNet [7] over RU for different number of PEs/router

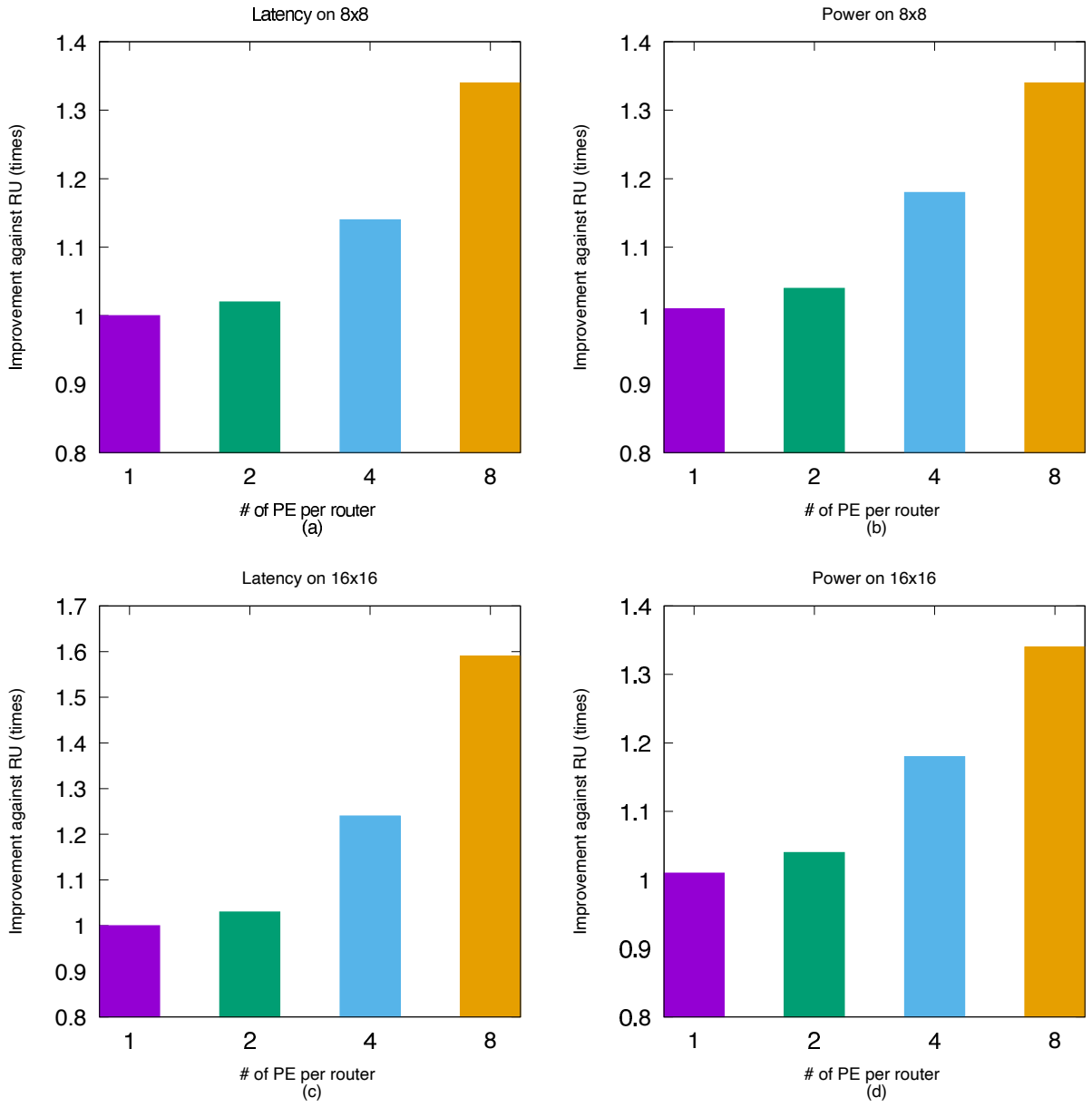


Figure 5.7: Improvement on total runtime latency (a),(c) and power consumption (b),(d) for ResNet-50 [12] over RU for different number of PEs/router

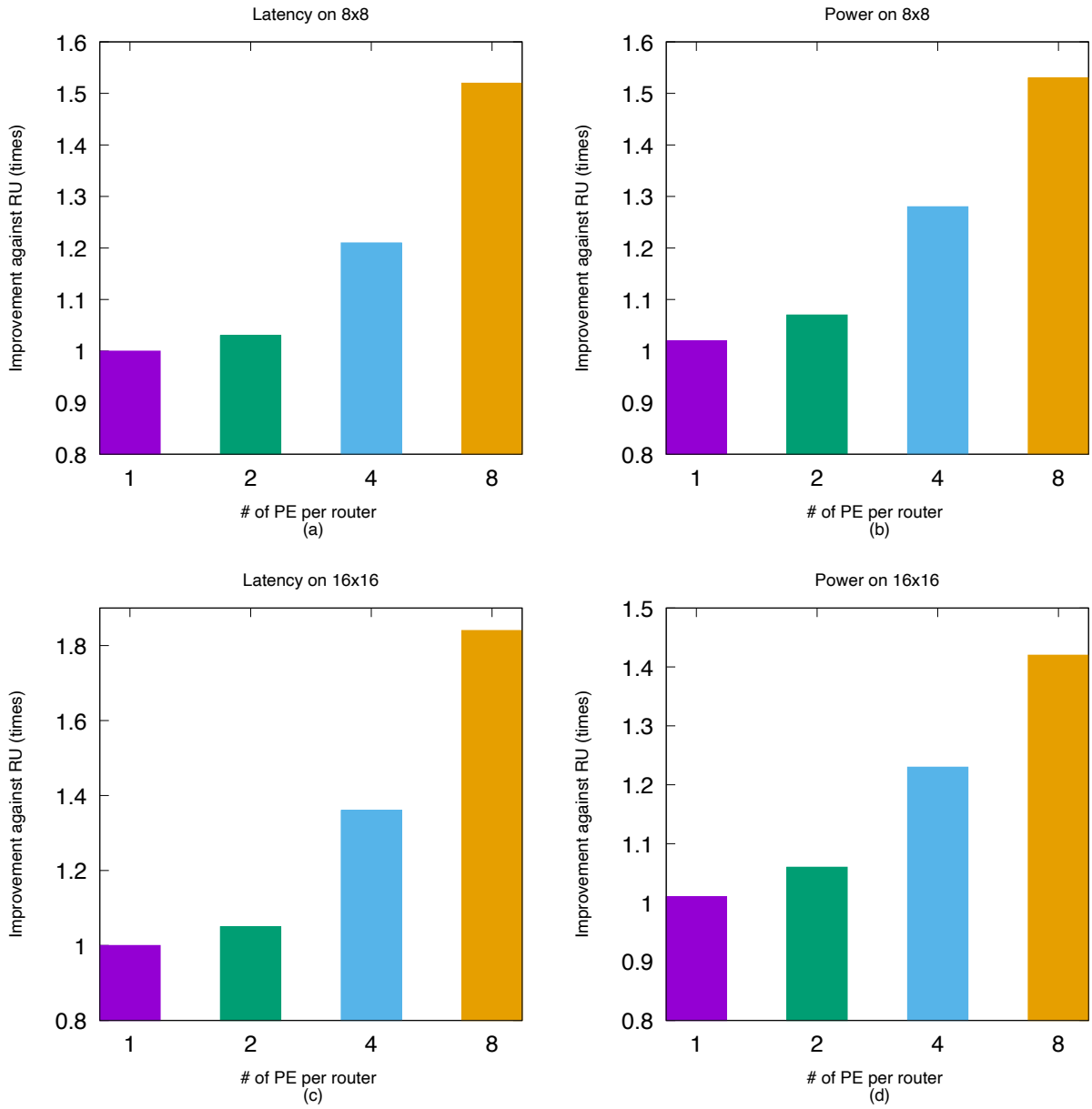


Figure 5.8: Improvement on total runtime latency (a),(c) and power consumption (b),(d) for VGG-16 [8] over RU for different number of PEs/router

the improvement in total runtime latency for all convolution layers in VGG-16 [8] for 8x8 and 16x16 meshes. For both ResNet-50 and VGG-16, we see a similar trend in performance improvement, with the 16x16 mesh offering more improvement on VGG-16 (up to 1.84 times) than the 8x8 mesh, and the improvement is better with the increasing number of PEs per router. On average, performance improvement is higher in VGG-16 compared with AlexNet and ResNet-50, as it has a lot more parameters to process than AlexNet and ResNet-50. Bigger than those in the other two networks, the convolution layers in VGG-16 require more rounds to complete, which makes the benefit of using gather supported routing more prominent.

Figs. 5.6(b) and (d) shows the improvement in the total network power consumption of the proposed method against the repetitive unicast method for AlexNet on the 8x8 and 16x16 mesh-based NoCs. Different from runtime latency, the total traffic communicated determines the network power consumption. For a smaller number of PEs per router the power improvement is minor because the power consumption due to streaming is higher than the power saving from the gather traffic. As the number of PEs/router increases, improvement in the total network power also increases (up to 1.4 times) because of the reduction in streaming power. More weights or inputs can be streamed with an increasing number of PEs/router, and the advantage of gather traffic over the repetitive unicast traffic is more significant. For the 16x16 mesh, we can see that the improvement is slightly less than the 8x8 mesh, which is due to the increased number of gather packets for the same PEs/router. Figs. 5.7(b) and (d) show the improvement in power performance for ResNet-50 [12] for 8x8 and 16x16 meshes and Figs. 5.8(b) and (d) show improvement in power performance for VGG-16 [8] for both meshes. For both models, we see a similar performance trend as in

Table 5.2: Hardware overhead

Metric \ Router	Baseline	Proposed
Area (μm^2)	$72,106$	$74,950$
Power (mW)	26.30	27.87

Table 5.3: Comparision with NeuronLink [39]

	NeuronLink[26]	Ours
Topology	4 chips, 4x4 mesh	8x8 mesh
Technology	$32nm$	$45nm$
Frequency	$1.2GHz$	$1GHz$
Area	$41.2mm^2$	$40.19mm^2$
Power	$15.4W$	$23.23W$
GOPS/mm^2	508.9*	398.09
GOPS/W	1361.7**	688.73

When scaled to same techonology parameter as ours,
* GOPS/mm^2 is 195.08
** GOPS/W is 586.53

Fig. 5.6(b) and (d). Similarly, the improvement of power consumption is higher in VGG-16 compared to AlexNet and ResNet-50 for the same reason as the latency improvement.

5.4.3 Hardware Overhead

We used DSENT [61] to estimate the area and power of a baseline router with the configuration shown in Table 5.1 without the gathering features. The baseline router operating on a 1 GHz clock consumes 26.3 mW power with an area of $72,106$ μm^2. Table 5.2 shows the hardware overhead of the proposed router from Fig. 3.7 evaluated using the synthesis tool from the Synopsys Design Compiler with a 45 nm CMOS library. The power consumption of a proposed router is 27.87 mW and the area is $74,950$ μm^2. This increase in area and power is due to the addition of the payload generator and *load* signal generator blocks in the

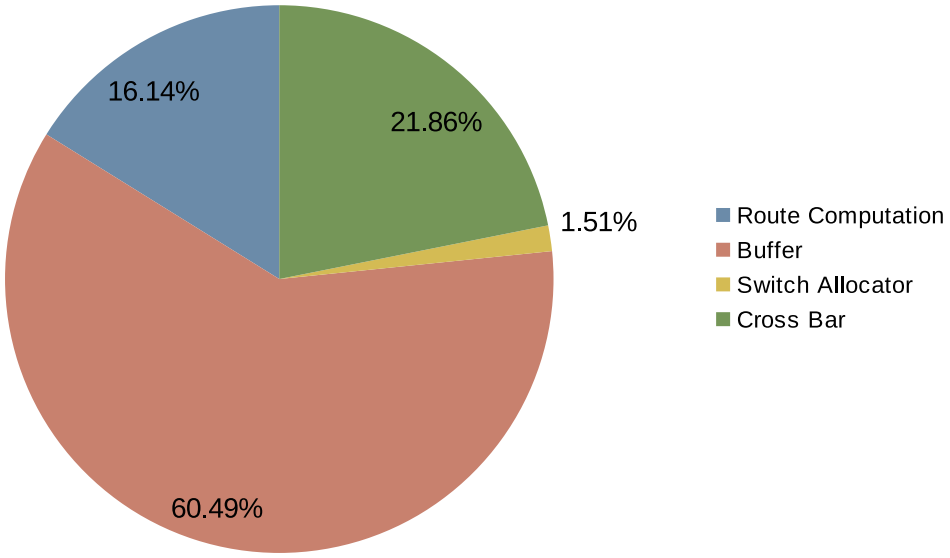


Figure 5.9: Dynamic power breakdown of the proposed router

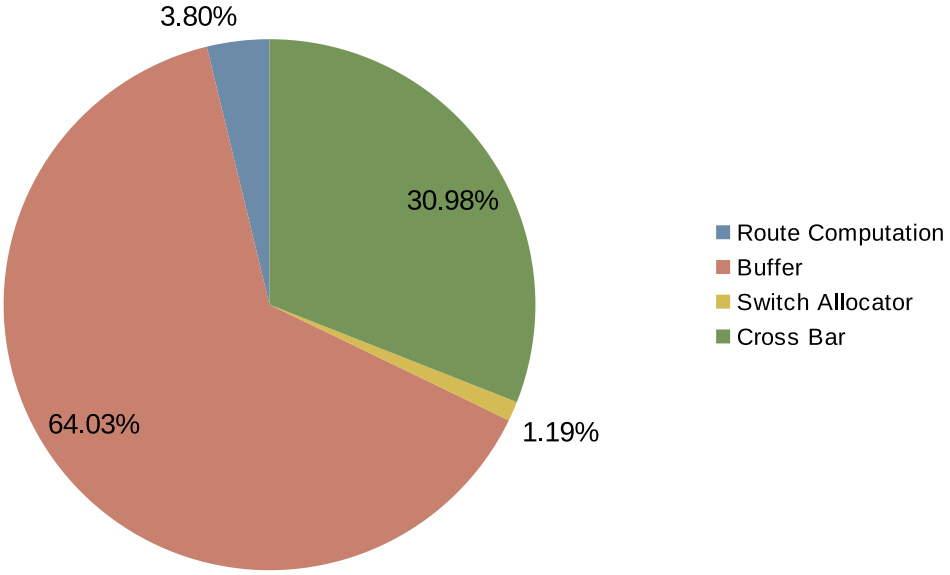


Figure 5.10: Dynamic area breakdown of the proposed router

router in order to support the proposed gather based routing. With the proposed changes in the router, the overhead is around a 4% increase in area and 6% increase in power, which is worthwhile considering the performance improvement with the changes. Fig. 5.9 and 5.10 show the dynamic power and area breakdown of the proposed router. Other components of the accelerator system include the streaming buses and PEs. One streaming bus (128-bit wide) along one row or column of mesh consumes 37.7 mW of power with the total wire area of 180 μm^2. Similarly, a PE structure consumes $63,933$ μm^2 and 30.2 mW area and power respectively to perform a 32-bit MAC and maxpool operation.

Table 5.3 shows the comparison of our proposed method with one of the most recent works NeuronLink[39]. Although, NeuronLink [39] does not support gather type traffic, both methods use mesh-based networks with similar routing methods. We used the method from [62] to scale NeuronLink [39] to $45nm$ node to compare with our work. The key metrics used to compare with NeuroLink [39] are area efficiency (GOPS/mm^2) and power efficiency (GOPS/W). The area efficiency and power efficiency of NeuronLink [39] when scaled to $45nm$ are 195.08 and 586.53, respectively. Our work has an area efficiency of 398.09 and the power efficiency of 688.73. Our proposed method is 2.04× better in area efficiency and 1.17× better in power efficiency than the scaled result of the NeuronLink [39].

5.5 Summary

This chapter presented an architecture to support multiple PEs and evaluate the performance using gather support and direct data streaming architectures on mesh-based NoC to handle abundant many-to-one and one-to-many traffic in DNN workloads. The OS dataflow

model is adopted to study the proposed method, which is evaluated using three DNN models: AlexNet [7], ResNet-50 [12], and VGG-16 [8]. The analysis shows that the two-way streaming architecture achieves a more significant improvement in the runtime latency of a convolutional layer. Simulation results confirm the effectiveness of the proposed method, which achieves up to 1.8 times improvement in the runtime latency and up to 1.7 times improvement in the network power consumption. The hardware overhead of the proposed method is justifiable for the performance improvements achieved over the repetitive unicast method. Further, the presented method supports all three kinds of communication traffic necessary for a DNN workload in an area and power-efficient way. Our method outperforms the most recent mesh-based accelerator [39] by 2.04× and 1.17× in terms of area efficiency and power efficiency, respectively.

CHAPTER 6

IN-NETWORK ACCUMULATION

While Chapters 3-5 are about the communication and computation support schemes in a DNN accelerator, this chapter focuses on reducing the network load and potentially memory transactions. Section 6.1 explains the motivation behind this study. Section 6.2 presents the proposed architectural support required to perform In-Network Accumulation (INA). Section 6.3 presents the result of the performance evaluation and finally, Section 6.4 summarizes this chapter.

6.1 Motivation

The dominant computing architecture today relies on memory to provide data to the PE when required. This approach of computing has hit the memory wall i.e., there is a big gap to fill in between the memory latency and CPU execution latency. To overcome this gap, near memory computing or in-memory computing [63] is considered a promising candidate. The main philosophy behind these architectures is to process the data close to where it is stored. Inspired by this philosophy, in this chapter we present in-network accumulation to accelerate the DNN workload in accelerators. The fundamental principle of this architecture is to process the data in the network while transitioning.

There exist enough operations in the DNN workload that can be offloaded to the memory as near-memory computing. Authors in [64] proposed an in-memory computing architecture

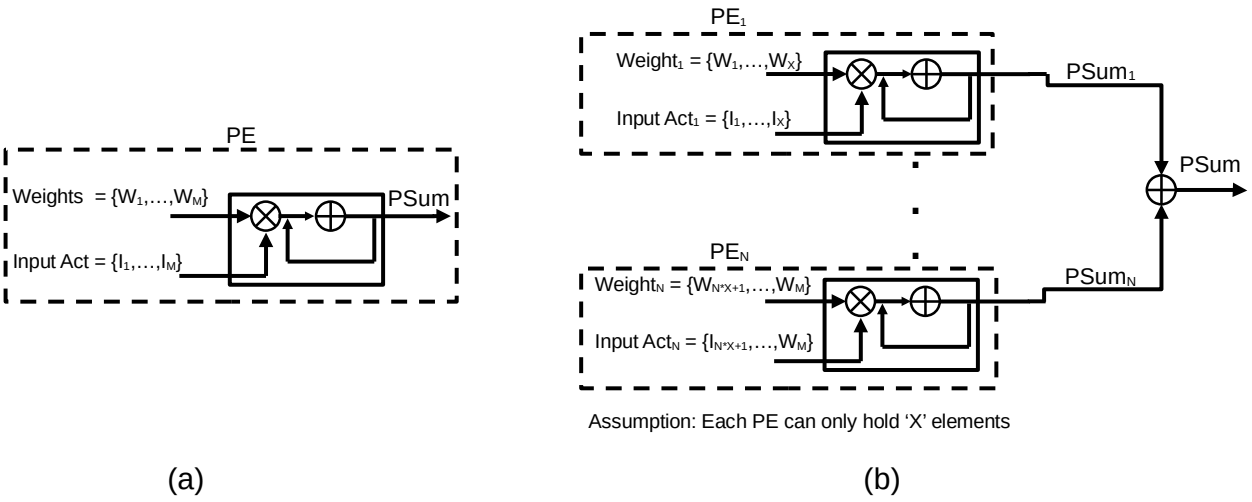


Figure 6.1: Accumulation organization (a) OS dataflow model (b) WS/RS dataflow model

for neural network applications, however, this method is not suitable for CNN workloads. This architecture does not offer flexibility in data reuse which leads to frequent reloading of input feature maps and weights during the DNN execution. Similarly, authors in [65] propose a Computing-On-the-Move (COM) architecture to address the shortcomings of [64]. COM [65] is implemented on a 2D Mesh NoC to enable the inter-memory computation like psum addition.

However, the aforementioned architectures cannot be easily integrated with existing accelerator designs adopting different dataflow models. Let us assume a WS dataflow model, where filter weights are stored at local PEs and kept there until all the MAC operations involving the weights are exhausted. The major bottleneck in this approach is to keep the filter weights in the local PE where we have limited scratch memory or a register file. To address this problem, only a portion of the weights are kept in a PE, and overall multiple PEs share and store the weights of one filter. This distribution of filter weights results in the distribution of the partial sum across multiple PEs which need accumulation. A similar

distribution also occurs in the RS dataflow model.

Fig. 6.1 shows the psum generation method which is explained by showing MAC operation in different dataflow models. A weight vector consisting of M elements $W_1, ..., W_M$ and an input vector consisting of M elements $I_1, ..., I_M$ are streamed from the streaming bus to the PE in the OS dataflow model where the psum accumulation is limited to a single PE as shown in Fig. 6.1 (a). However, due to the distributed nature of weights/inputs in the WS/RS dataflow model particularly due to the memory limitation, psum accumulation needs to happen across different PEs as shown in Fig. 6.1 (b). Assuming that each PE can hold only X elements from the weights and input vector, only a part of the psum is generated at each PE. This provides an opportunity to optimize the way psum accumulation happens in the WS/RS dataflow model.

6.2 Architectural Support

This dissertation proposes the in-network accumulation design by modifying the router to support psum accumulation which makes it easy to integrate with the existing accelerator design. The INA is controlled by a central controller, hence the accelerator can be optimized for individual DNN layers unlike in COM [65] where the dataflow control is distributed. Fig. 6.2 shows an example of the WS dataflow model on a 4×4 mesh. Assume that a CONV operation with a filter $F \epsilon \{F_1, F_2, F_3, F_4\}$, input activation $I \epsilon \{I_1, I_2, ..., I_P\}$ each with a dimension of $C \times R \times R$ is implemented on a 4×4 mesh where each PE can hold half the elements of a filter. In order to generate one output activation, two PE are required as

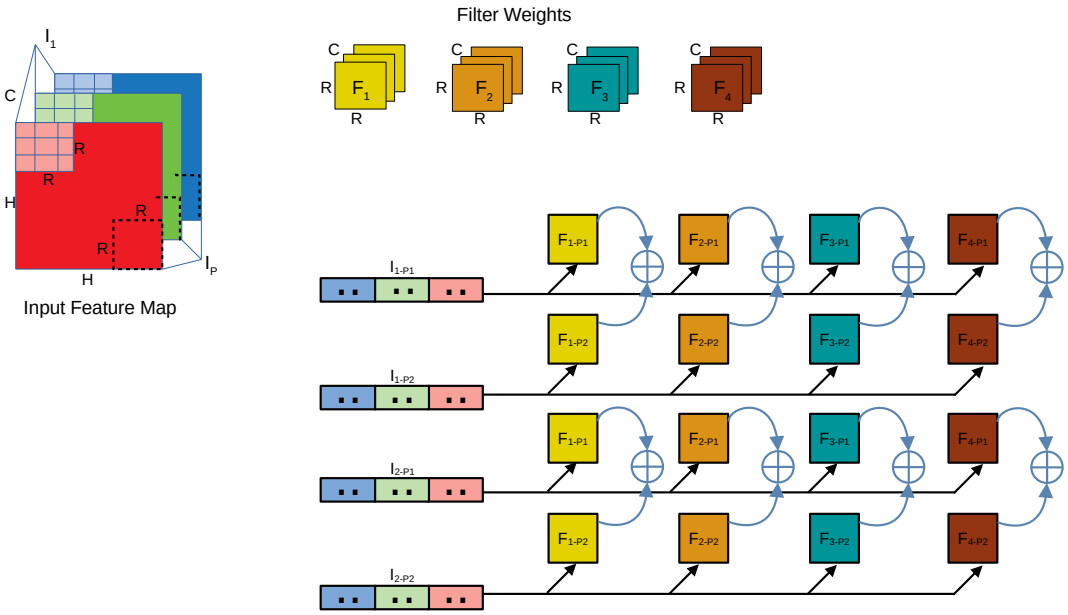


Figure 6.2: WS dataflow in 4x4 mesh NoC

shown in Fig. 6.2 where the filter weights are divided into two parts $P1$, $P2$ and distributed among two adjacent PEs in the same column. After both PEs generate a psum, they can be accumulated across the PEs to get the final output activations.

Fig. 6.3 shows the flow for psum accumulation with and without in-network support. Fig. 6.3 (a) shows the flow where ① - ④ are the sequential steps that need to be performed in order to process the accumulation of the partial sum generated at each node on a WS or RS dataflow model as shown in Fig. 2.3 and Fig. 2.4. In the absence of in-network accumulation, the incoming packet ① with partial sum is first ejected ② to the local port where accumulation happens locally and then a new packet needs to be injected ③ back to the network for the next hop ④. It seems an obvious way of performing the accumulation in a mesh based network without the in-network accumulation support.

Fig. 6.3 (b) shows the alternative approach of accumulation of the partial sum generated at each node on a WS or RS dataflow model in the presence of an in-network accumulation

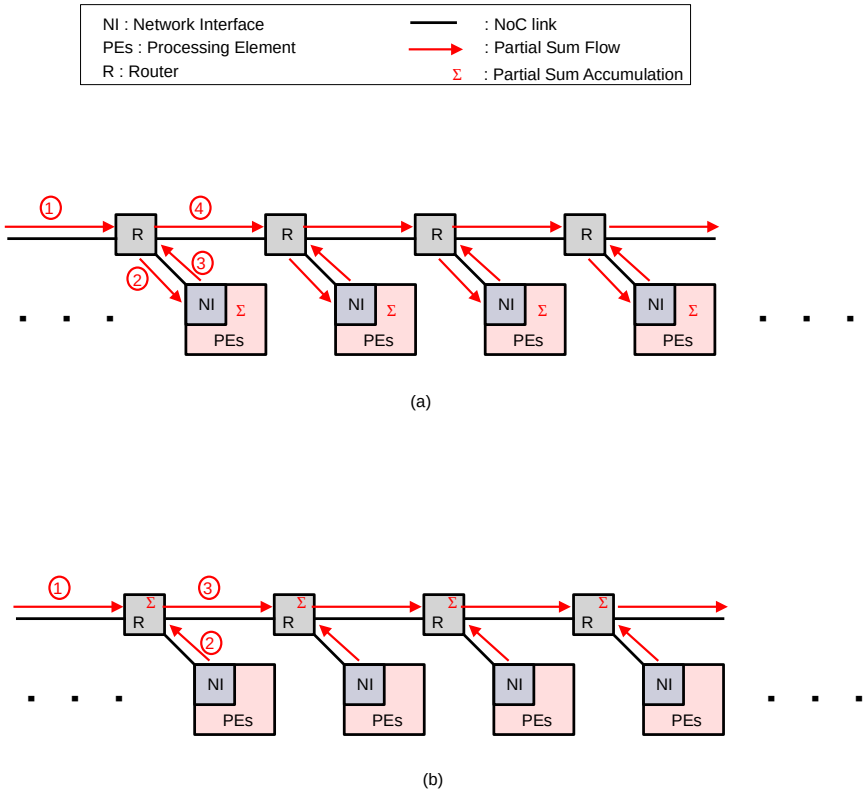


Figure 6.3: Partial sum (PSum) accumulation flow for WS/RS dataflow (a) without in-network accumulation support (b) with in-network accumulation support

unit. Steps ① - ③ show the flow of partial sum accumulation in the presence of in-network accumulation support. This method of accumulation brings the computation close to the data source and helps in removing the redundant and obvious network transactions. The incoming packet ① can be directly accumulated in the router with the partial sum as generated by the local node ② before forwarding it to the next hop ③. One thing to notice here is ejection and injection of the packets are saved with the support of in-network accumulation which not only helps in improving the latency but also saves the network power consumption.

6.2.1 INA Modeling

INA is beneficial for the WS and RS dataflow models where certain parameters are stored at the PE until resued. Consider a WS dataflow model on a $N \times N$ mesh with 1 PE/router with a memory capacity of M bits. For any CONV layer with $K \times K$ kernel size, C channels, F filters, $O \times O$ output feature map, q-bit precision. We can model the INA of partial sums using a series of equations shown below:

Condition to perform INA, iff Equation (6.1) holds true.

$$(K \times K \times C \times q) > M \iff INA \tag{6.1}$$

Number of PEs ($P_{\#}$) to distribute the filter weights is:

$$P_{\#} = \left\lceil \frac{K \times K \times C \times q}{M} \right\rceil \tag{6.2}$$

Rounds of INA ($INA_{\#}$) to complete one CONV layer on a given $N \times N$ mesh:

$$INA_{\#} = \left\lceil \frac{F}{N} \cdot \frac{O \times O}{\lfloor \frac{N}{P_{\#}} \rfloor} \right\rceil \tag{6.3}$$

Equation 6.1 shows that the condition to perform an INA during a CONV layer execution is dependent on the size of memory for PEs. With sufficient memory there is no need for the INA, however, this is not feasible practically. Hence, in order to keep M in the practical range, certain dataflow models like WS, RS needs to distribute the weights, inputs, etc. among multiple PEs leading to the INA as an effective solution. Note that $INA_{\#}$ represents the total rounds of psum accumulation on a $N \times N$ mesh for a given CONV layer. This

Table 6.1: INA evaluation for AlexNet [7]

Layer	K	C	F	O	$P_{\#}$	$INA_{\#}$, N=8	$INA_{\#}$, N=16
CONV1	11	3	64	55	1	NA	NA
CONV2	5	64	192	27	2	4374	1094
CONV3	3	192	384	13	2	2028	507
CONV4	3	384	256	13	4	2704	676
CONV5	3	256	256	13	3	2704	541

*Note: q=32bit, M=32KB, 1 PE/Router

Table 6.2: INA evaluation for VGG-16 [8]

Layer	K	C	F	O	$P_{\#}$	$INA_{\#}$, N=8	$INA_{\#}$, N=16
CONV1	3	64	3	224	1	NA	NA
CONV2	3	64	64	224	1	NA	NA
CONV3	3	128	64	112	2	25088	6272
CONV4	3	128	128	112	2	50176	12544
CONV5	3	256	128	56	3	25088	5018
CONV6	3	256	256	56	3	50176	10036
CONV7	3	256	256	56	3	50176	10036
CONV8	3	512	256	28	5	25088	4182
CONV9	3	512	512	28	5	50176	8363
CONV10	3	512	512	28	5	50176	8363
CONV11	3	512	512	14	5	12544	2091
CONV12	3	512	512	14	5	12544	2091
CONV13	3	512	512	14	5	12544	2091

*Note: q=32bit, M=32KB, 1 PE/Router

means the total number of accumulations in each round is different i.e., the number of parallel accumulations happening each round. Tables 6.1 and 6.2 show the number of rounds of INA operation for 8×8 and 16×16 mesh, respectively. It is seen that there exist enough INA operations that can be optimized. Also, we can see that VGG-16 [8] has a lot of INA rounds compared to AlexNet [7] since VGG-16 has a lot of filters with larger channels which makes the parameter fit in one PE's $M - bits$ memory difficult. Hence, the weights are distributed among different PEs leading to an increase in INA rounds.

Similarly, for multiple E PEs/router Equation 6.3 can be written as:

$$INA_{\#E} = \left\lceil \frac{F}{N \cdot E} \cdot \frac{O \times O}{\lfloor \frac{N}{P_{\#}} \rfloor} \right\rceil \tag{6.4}$$

It is also to note that, with the increase in computation capacity the performance does not scale linearly since the addition of E PEs/router will increase the communication load in the network as well.

6.2.2 Router Support

Fig. 6.4 shows the changes required to make in the existing router architecture from Fig. 3.7 to support the INA. The INA block (Fig. 6.5(a)) is added along with the required control (Fig. 6.5(b)) and signals to support the INA as shown in Fig. 6.5. The INA block is responsible for monitoring the operands from the incoming local port i.e., NI, and from the neighboring node i.e., N, S, E, W port. ①, ②, ③ from Fig. 6.3(b) can be equivalently mapped to the states *Acquire Operand* 1, *Acquire Operand* 2 and *Summation* states respectively from Fig. 6.5 (b). The source of *Operand* 1 is the NI where the local PE calculates the partial sum based on the set of weights and input activations distributed to this node and intends to forward it to the next node. The source of *Operand* 2 is the incoming packet from the neighboring node which contains the rest of the partial sum.

During the DNN execution, weights are distributed to different nodes in part. The number of parts depends on the memory size of the node and the size of weights for a CONV layer. The scheduler or the top level controller will assign at the runtime i.e., during streaming which node should initiate the INA packet. This is usually the first node that

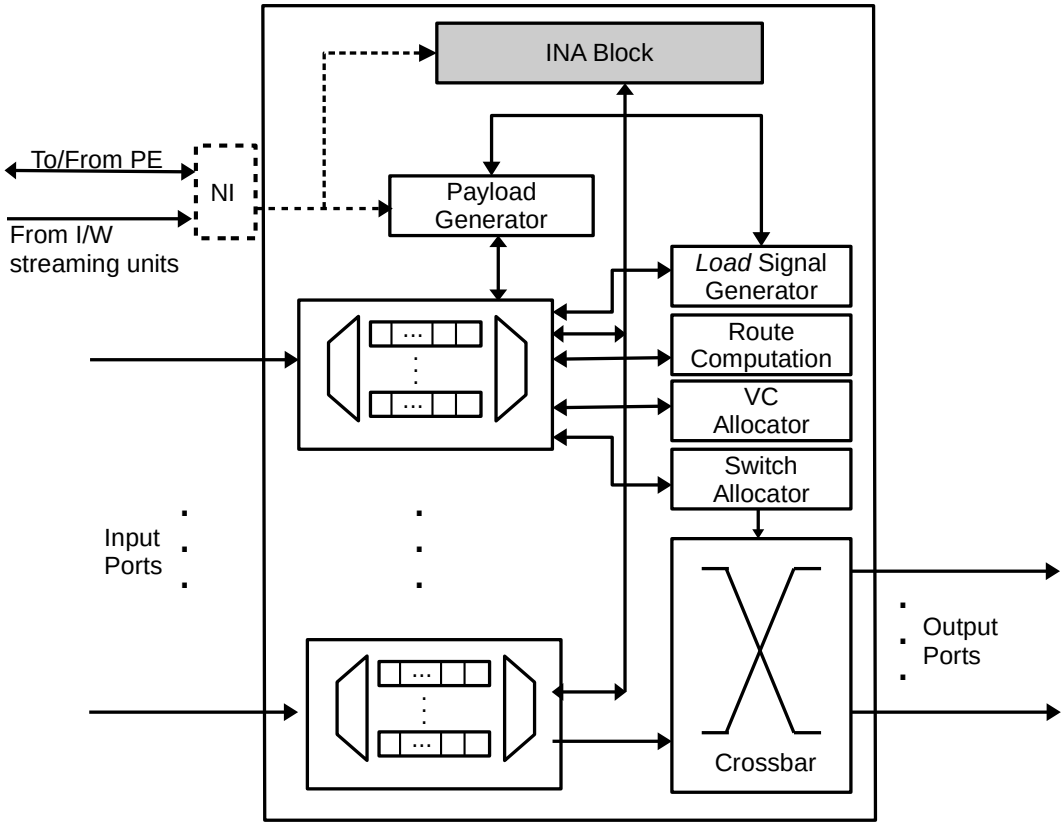


Figure 6.4: INA router microarchitectural

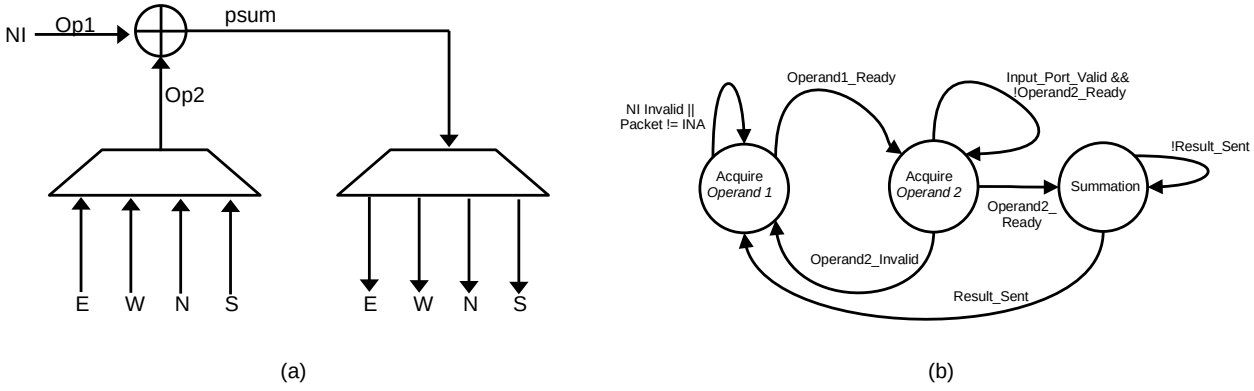


Figure 6.5: INA support (a) INA block (b) INA control logic

holds the first part of the weight during the distribution. We can use the packet format shown in Fig. 3.3 to add this information in the field Packet Type (PT).

The purpose of this work is to show the effectiveness of DNN execution by evaluating the network efficiency. There are multiple ways of implementing the adder unit in the router to achieve INA. Various analog adders have been proposed based on ReRAM technology one such example is [66]. To adopt the ReRAM based structure, we need some additional structures like the digital to analog converter and vice versa. There are other fast digital adders [67]-[69] as an alternative to analog adders. For the multiple PEs/router scenario we can further extend the adders into a simple SIMD/Vector adder unit, some of the alternative choices are presented in [70]-[72]. In this work, our primary goal is to show the efficiency of the INA, hence we prefer to use a digital adder from [67] to validate our concept.

6.3 Performance Evaluation

We assume that there is a higher level entity or a mapping framework similar to [28], [29], [39], [40], [60] that does the task of mapping neurons to the PEs, streaming of the inputs and weights in parts, controlling timing for better synchronization without stalls, so that our focus is on evaluating the performance of the on-chip network.

To evaluate the performance of INA for the WS dataflow model, we ran simulations for different CNN workloads on 8x8 mesh-based NoCs modified with a two-way streaming architecture. For the WS dataflow model, all the weights are streamed in full or in parts as needed depending on the memory availability, these weights are stored in the local memory of the PE and used for multiple rounds of MAC operation for all the input activations. Hence,

input activation will not be streamed and the PE should not start the MAC operation until all the weights are streamed to the local memory of the PE. In this section, we describe the experiment settings, followed by presenting the results.

6.3.1 Experiment Setup

Table 6.3 shows the NoC setting used for performance analysis. To accommodate the gather payload after INA, we have used various gather flit sizes. After the INA, the gather packet can have 1 flit to 9 flits depending upon the number of nodes used to get the accumulation result. For the 1 PE/router case, if the INA is performed on 2 nodes then the gather packet needs to collect the result of 4 nodes on 8x8 mesh. This information is identified at the compile time by the higher level entity or a mapping framework.

We compare the WS dataflow model with and without INA both using gather packets in terms of the runtime latency and power consumption, we further extended the result to compare the INA-enabled WS dataflow model with the OS dataflow model with gather support from Section 5.4.1. We assume that the INA is using a similar digital adder proposed in [67] which is fast and has various bit widths suitable for our analysis, we also assume that the accumulation latency in both cases, with INA and without INA are comparable. We use the parameters obtained from Pytorch framework [47] to model the traces for the NoC. The neurons are organized similarly to the experiment setup from Section 5.4.1. Accumulation happens using INA in the router as explained in Fig. 6.3. We have used a cycle-accurate C++ based NoC simulator [46] to simulate the generated traces for AlexNet [7], ResNet-50 [12], and VGG-16 [8]. Orion 3.0 [48] is used to estimate the power consumption for NoC.

Table 6.3: Network configuration for INA simulation

Topology	8x8 Mesh
Virtual Channels	2
Latency	router: 4 cycles, link: 1 cycle
Buffer Depth	4 flits
Flit Size	128 bits/flit
Gather Payload	32 bits
Number of PE per router	1,2,4,8
Gather Packet Size	3,5,9 flits/packet for multiple PEs/router
Unicast Packet Size	2,3 flits/packet for multiple PEs/router

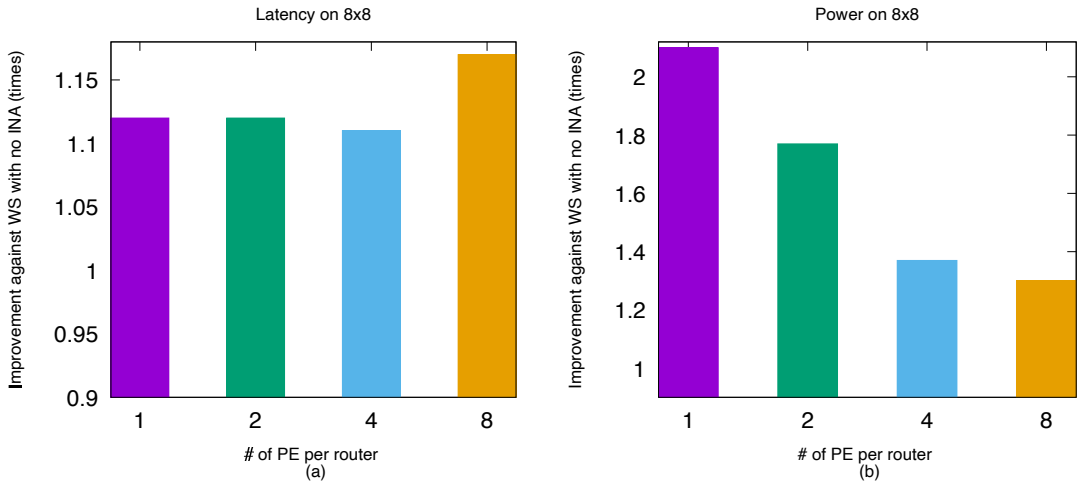


Figure 6.6: Improvement on total runtime latency (a) and power consumption (b) for AlexNet [7] over WS without INA for different number of PEs/router

6.3.2 Results

Fig. 6.6 (a), (b) shows the improvement in the total runtime latency and power consumption of the WS dataflow model with INA against the one without INA case for all convolution layers in AlexNet [7] on 8x8 mesh-based NoCs. We can see that INA can boost the latency up to 1.17× and power consumption up to 2.1× compared to without INA case. A similar improvement is seen across other DNN workloads as shown in Fig. 6.7 and Fig. 6.8 for ResNet [12] and VGG-16 [8], respectively.

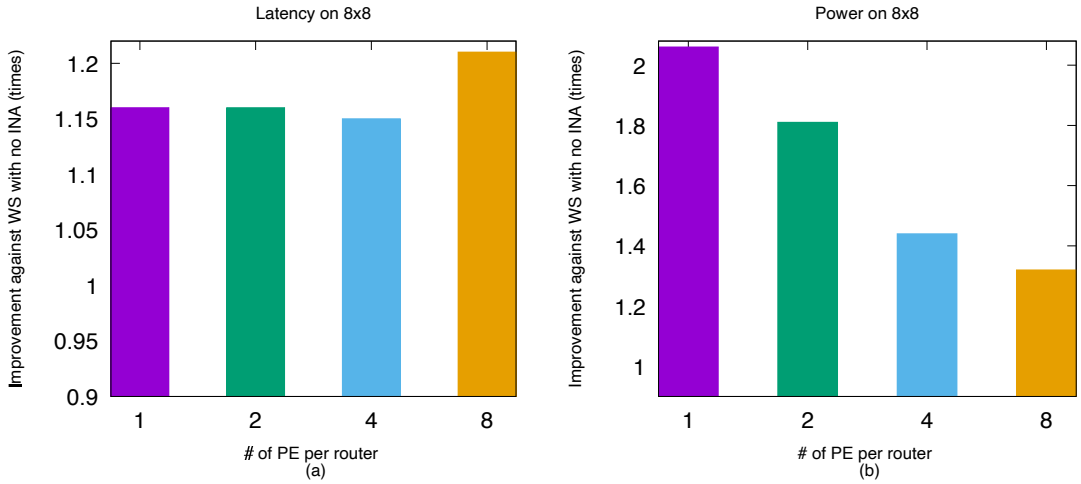


Figure 6.7: Improvement on total runtime latency (a) and power consumption (b) for ResNet-50 [12] over WS without INA for different number of PEs/router

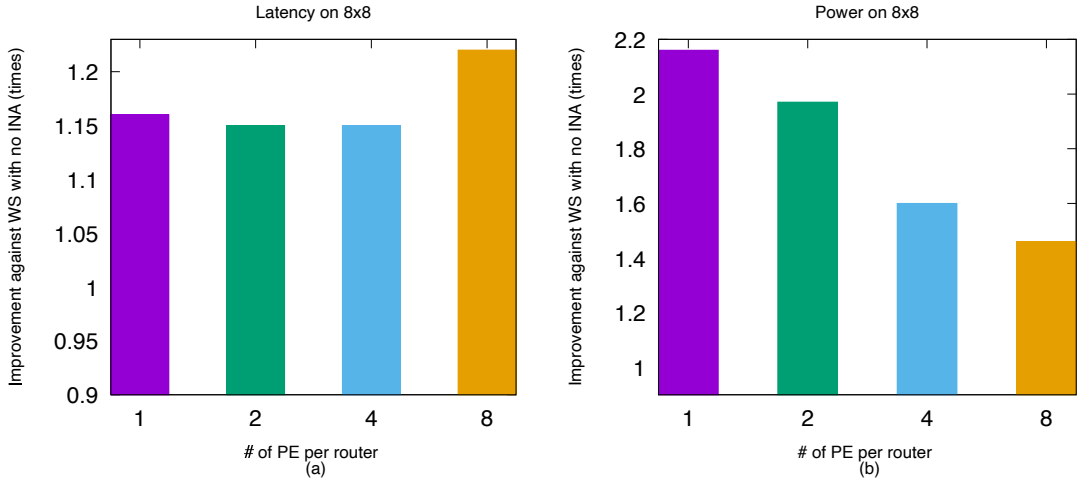


Figure 6.8: Improvement on total runtime latency (a) and power consumption (b) for VGG-16 [8] over WS without INA for different number of PEs/router

For all the workloads, the improvement in latency is almost similar in 1, 2, and 4 PEs/router, and the improvement increases for 8 PE/router. Latency improvement in this comparison is determined by the packet size used for both INA and without INA. For 1,2,4 PEs/router, both the cases will use a similar number of flits per packet, and hence we can also see a similar improvement in latency. However, for 8PEs/router packet size should increase, and without INA case larger packet size adds up to more latency. We can also see that VGG-16 on average has a larger improvement than other workloads due to the multiple rounds of INA ($INA_{\#}$) needed in VGG-16. Even though ResNet-50 is bigger in terms of the number of CONV layers than VGG-16, most of the ResNet-50 does not need to split the weights among multiple PEs ($I_{\#}$) and hence ResNet-50 is not showing the highest improvement even with a larger network model.

As for the power improvement, a smaller number of PEs shows the highest improvement. As the number of PEs increases, the number of flits per gather packet should also increase to accommodate all the payloads which contribute to the additional power in the case of INA. Dynamic flit would have made an impact here but in this experiment, we have assumed a static packet size. However, without INA, packets do not need to increase the flit size since the addition is happening between two nodes and a smaller flit can be used to move the partial sum for accumulation. A similar trend for performance is seen across different workloads where VGG-16 is the highest performing due to the similar reasons as explained for the latency improvement.

Fig. 6.9 - Fig. 6.11 shows the comparison between the WS dataflow model with INA and gather supported and OS dataflow model with gather supported. We see that as the number of PE increases the latency improvement of the WS dataflow model is decreasing because,

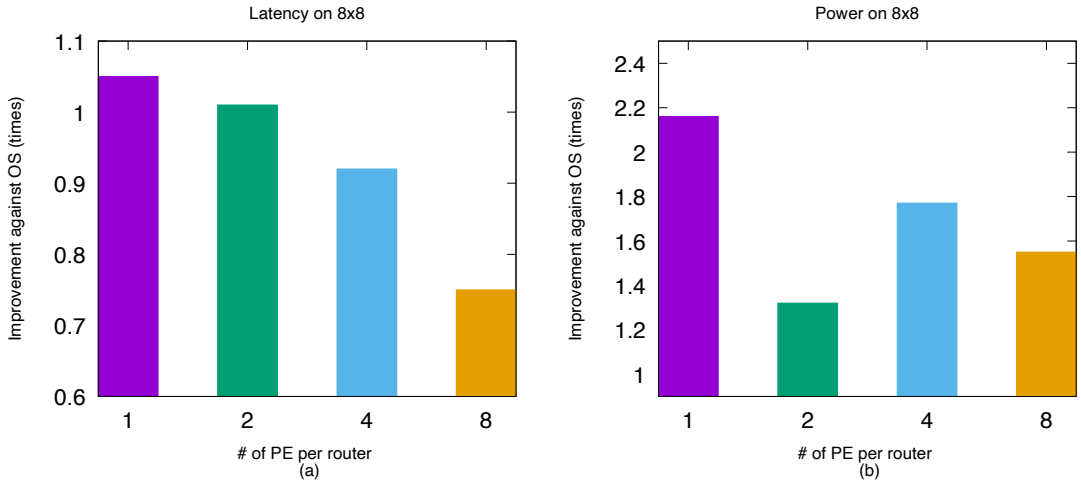


Figure 6.9: Improvement on total runtime latency (a) and power consumption (b) for AlexNet [7] over OS for different number of PEs/router

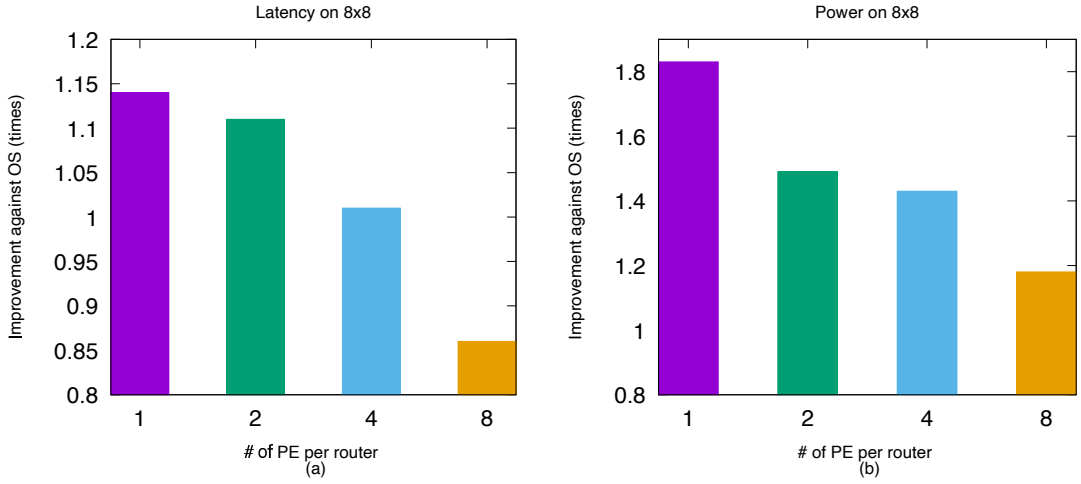


Figure 6.10: Improvement on total runtime latency (a) and power consumption (b) for ResNet-50 [12] over OS for different number of PEs/router

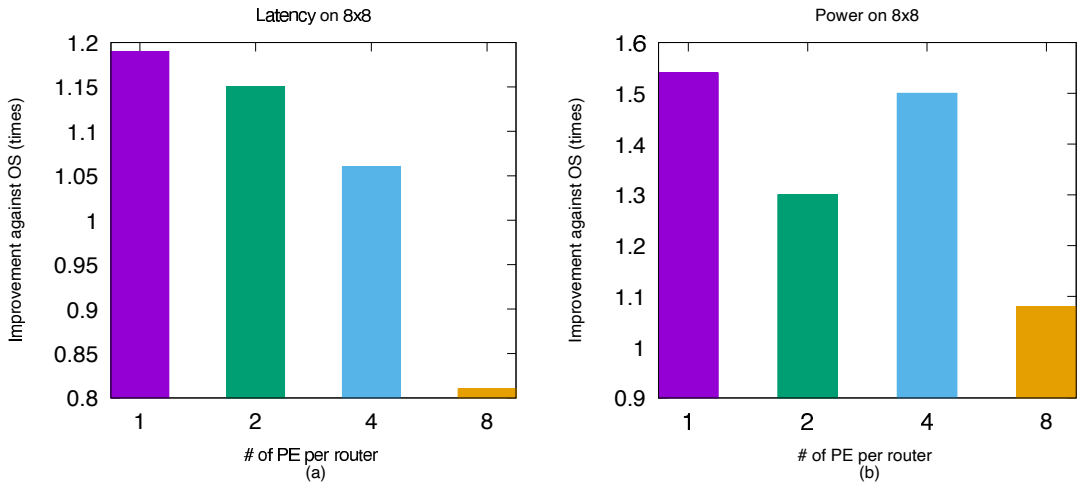


Figure 6.11: Improvement on total runtime latency (a) and power consumption (b) for VGG-16 [8] over OS for different number of PEs/router

for the WS dataflow model, weights need to be distributed before the psum accumulation begins. As the number of PE increases the distribution of weights also takes longer due to the larger packet size. However, on average we see on all the workloads WS dataflow model is performing better than the OS dataflow model. As for the power improvement, the WS dataflow model outperforms the OS dataflow model in all the cases. This improvement is mainly due to the better reuse of weights leading to less streaming than the OS dataflow model. We see fluctuations in power improvement going from 1 PE/router to 2 PEs/router and from 4 PEs/router to 8 PEs/router because this is the boundary where a change in the flit size happens due to the increase in the number of PEs in the router.

6.4 Summary

DNN workloads can be executed with a variety of dataflow models. Supporting different dataflow models is important to ensure effective data resue during the DNN execution. In this chapter, we present the In-Network Accumulation (INA), an architecture to support partial sum accumulation without ejecting the packet from the network on a WS dataflow model. We performed the simulation on different DNN workloads and showed the improvement of INA in both runtime latency and power consumption compared with the WS dataflow model without INA. We can see up to 1.22× improvement in the latency and 2.16× improvement in the power consumption. We further evaluated the WS dataflow model with INA and gather supported routing against the OS dataflow model with gather support. We can see up to 1.19× latency improvement and 2.16× improvement in the power consumption across different DNN workloads.

CHAPTER 7

CONCLUSION AND FUTURE WORKS

7.1 Contributions

Research work on application-specific computing domains is becoming popular, which has led to various studies focused on efficient and effective DNN workload execution. In this dissertation, the study is focused on efficient and scalable communication supporting solutions for various traffic patterns in NoC-based DNN accelerators. The contributions of this dissertation are summarized below:

- Gather supported routing for many-to-one traffic: This dissertation proposes a gather based routing algorithm that effectively accelerates the DNN traffic. Experimental result demonstrates the gather supported routing significantly reduces the network latency and power consumption than the repetitive unicast method. We also provide a modified router microarchitecture so that a minimal change will be required for the existing router to support gather traffic. This is also shown quantitatively by measuring the hardware overhead of the changes required in the router.

- Analytical model to evaluate the effectiveness of gather traffic: This dissertation also provides an analytical framework that evaluates the mathematical equivalency of the proposed algorithm. Simulation results also validate the model, further analysis of routing parameters like δ and packet size for gather supported routing is performed

which will help determine the best parameters for different network configurations.

- Streaming bus architecture for multicast: This dissertation proposes a streaming bus architecture to support multicast traffic in a DNN accelerator. Two versions of the streaming support are presented and the experimental evaluation of CONV layers shows the effectiveness of both the models. Depending on the application's requirement, one method can be chosen over the other.

- Scalable computation support: This dissertation provides a framework to support multiple PEs per router, this solution provides scalable support to increase the computation throughput. By this way, the computing resource scaling can still take the advantage of all the proposed communication infrastructure. Experimental results confirm the effectiveness of the proposed communication supporting and throughput enhancement solutions in accelerating various DNN workloads.

- In-Network Accumulation (INA): This dissertation proposes INA architecture that eliminates the unnecessary movement of the psum within the network leading to better latency and power. Experimental results on the WS dataflow model show the effectiveness of this method when compared with the OS dataflow model across various DNN workloads.

To summarize, this dissertation provides solutions to support all the kinds of communication traffic that exist in the DNN workload. The communication support is scalable across the mesh network of various sizes. The methods proposed in this dissertation are also compatible with the scaling of the computing resources. Since both computation and com-

munication scaling go side by side, the proposed solutions will help improve the performance of the NoC-based DNN accelerator.

7.2 Future Work

As the field of DNN is rapidly changing, the hardware architecture to support these applications should grow at the same pace. This rapid growth leads to various challenges and opportunities which are summarized below:

- Efficient mapping and scheduling solutions: As the DNN architecture is evolving the relation and interaction between the layers are also changing. There is a need for a better neuron mapping and scheduling process where the relation between the neurons is taken into account to map the neuron to the PEs. This process should also account for the communication overhead among the layers while making the best use of existing hardware resources.

- Better HW-SW co-design: While there is a hardware aspect of DNN execution, there is also a software aspect that is equally effective in reducing the complexity of DNN execution. Reducing the bit precision, exploiting the sparsity are a few software techniques that could help in effective DNN execution. Future work can further explore the co-design principle to incorporate the software input in better DNN hardware execution.

- Fused PE and Router: As shown from our work on In-Network Accumulation, the role of routers in DNN accelerators can be beyond communication. This can be the motivation to further explore the opportunities in the fused router and PE design where

we can fuse these two units as one. This lightweight fused node can process DNN operation much faster without having the overhead of routing as in traditional routers.

- Effectiveness of proposed method in other workloads: While the gather supported routing and streaming architectures are proposed for DNN execution, these methods can also be applied to other workloads. For sparse multicast workload, the streaming units can be more effective than path-based methods. Even in multi-threaded applications, better scheduling can lead to effective use of gather supported routing.

BIBLIOGRAPHY

[1] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444 (2015). https://doi.org/10.1038/nature14539

[2] C. Chen, A. Seff, A. Kornhauser and J. Xiao, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving," 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, 2015, pp. 2722-2730

[3] Esteva, A., Kuprel, B., Novoa, R. et al. Dermatologist-level classification of skin cancer with deep neural networks. Nature 542, 115–118 (2017). https://doi.org/10.1038/nature21056

[4] Norman P. Jouppi, et.al.,"A domain-specific architecture for deep neural networks," in Commun. ACM, Sept. 2018, pg.50–59

[5] Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers," in IBM Journal of Research and Development, 1956, pp. 210-229

[6] Fei-Fei Li, Andrej Karpathy, and Justin Johnson, "Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition", http://cs231n.stanford.edu/

[7] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," CoRR, vol. abs/1404.5997, 2014. [Online]. Available: http://arxiv.org/abs/1404.5997

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in Proc. ICLR, 2015.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Nature, 1986. pg.533–536

[10] A. Vaswani et. al., "Attention is all you need", in Proc. Neural Information Processing Systems (NeurIPS), 2017

[11] I. Goodfellow, et. al., "Generative adversarial nets," in Proc. Neural Information Processing Systems (NeurIPS), 2014

[12] He, K., Zhang X. et al., "Deep Residual Learning for Image Recognition," in Proc. CVPR, 2016.

[13] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks", in Proc. ICANN, 2014, pp. 281-290

[14] W. J. Dally and B. Towles, “Route packets, not wires: on-chip interconnection networks,” in Proc. 38th DAC, 2001, pp. 684–689.

[15] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna, ”MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects”, in Proc. ASPLOS, March 2018.

[16] S. Carrillo et al., ”Scalable Hierarchical Network-on-Chip Architecture for Spiking Neural Network Hardware Implementations,” in IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 12, pp. 2451-2461, Dec. 2013.

[17] B. Bohnenstiehl et al., ”KiloCore: A 32-nm 1000-Processor Computational Array,” in IEEE Journal of Solid-State Circuits, vol. 52, no. 4, pp. 891-902, April 2017.

[18] A. Touzene, ”On All-to-All Broadcast in Dense Gaussian Network On-Chip,” in IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 4, pp. 1085-1095, 1 April 2015.

[19] B. Tiwari, M. Yang, Y. Jiang and X. Wang, ”Efficient On-Chip Multicast Routing based on Dynamic Partition Merging,” 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Vasteras, Sweden, 2020, pp. 274-281.

[20] A. Karkar, T. Mak, K. Tong and A. Yakovlev, ”A Survey of Emerging Interconnects for On-Chip Efficient Multicast and Broadcast in Many-Cores,” in IEEE Circuits and Systems Magazine, pp. 58-72, 2016

[21] L. Benini and G. De Micheli, ”Networks on chips: a new soc paradigm,” Computers, Jan 2002, pp. 70-78

[22] W. Dally and B. Towles. Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[23] C. Bienia and K. Li, ”Parsec 2.0: a new benchmark suite for chip-multiprocessors,” in Proc. 5th Annu. Workshop Model., Benchmarking and Simul., 2009

[24] T. Chen et al., “DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning,” in ASPLOS, 2014.

[25] V. Sze, Y. Chen, J. Emer, A. Suleiman and Z. Zhang, ”Hardware for machine learning: Challenges and opportunities,” 2017 IEEE Custom Integrated Circuits Conference (CICC), Austin, TX, 2017, pp. 1-8

[26] ”Wafer-Scale Deep Learning,” 2019 IEEE Hot Chips 31 Symposium (HCS), Cupertino, CA, USA, 2019, pp. 1-31.

[27] D. Abts et al., ”Think Fast: A Tensor Streaming Processor (TSP) for Accelerating Deep Learning Workloads,” 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 2020, pp. 145-158.

[28] Z. Du et al., "ShiDianNao: Shifting vision processing closer to the sensor," 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, OR, 2015, pp. 92-104

[29] Jouppi, N. P. et al., "In-datacenter performance analysis of a tensor processing unit", in Proc. 44th Int. Symp. Comp. Architecture (ISCA), 2017.

[30] Y. Chen, T. Yang, J. Emer and V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 2, pp. 292-308, June 2019.

[31] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in Proc. CVPR, 2016.

[32] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in Proc. ICAN

[33] H. Sharma et al., "From high-level deep neural models to FPGAs," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, 2016.

[34] E. Nurvitadhi, D. Sheffield, Jaewoong Sim, A. Mishra, G. Venkatesh and D. Marr, "Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC," 2016 International Conference on Field-Programmable Technology (FPT), Xi'an, 2016.

[35] H. T. Kung, "Why systolic architectures?" Computer, vol. 15, no. 1, pp. 37–46, Jan 1982.

[36] T. Luo et al., "DaDianNao: A Neural Network Supercomputer," in IEEE Transactions on Computers, vol. 66, no. 1, pp. 73-88, 1 Jan. 2017.

[37] D. Vainbrand and R. Ginosar, "Network-on-Chip Architectures for Neural Networks," 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip, Grenoble, 2010, pp. 135-144

[38] E. Painkras, et al., "SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation," IEEE J. Solid-State Circuits, vol. 48, no. 8, pp. 1943–1953, Aug. 2013.

[39] S. Xiao et al., "NeuronLink: An Efficient Chip-to-Chip Interconnect for Large-Scale Neural Network Accelerators," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 9, pp. 1966-1978, Sept. 2020.

[40] R. Hojabr, M. Modarressi, M. Daneshtalab, A. Yasoubi and A. Khonsari, "Customizing Clos Network-on-Chip for Neural Networks," in IEEE Transactions on Computers, vol. 66, no. 11, pp. 1865-1877, 1 Nov. 2017.

[41] B. Tiwari et al., "Improving the Performance of a NoC-based CNN Accelerator with Gather Support", in Proc. IEEE 33rd Intl. System-on-Chip Conference (SOCC), 2020.

[42] X. Liu and et al., "Neu-NoC: A high-efficient interconnection network for accelerated neuromorphic systems," in Proc. 23rd ASP-DAC, 2018, pp. 141–146.

[43] A. Firuzan, M. Modarressi, M. Daneshtalab, and M. Reshadi, "Reconfigurable network-on-chip for 3D neural network accelerators," in Proc. 12th IEEE/ACM NOCS, 2018, pp. 1–8

[44] H. Kwon, A. Samajdar, and T. Krishna, "Rethinking NoCs for spatial neural network accelerators," in Proc. 11th IEEE/ACM NOCS, 2017.

[45] Seyedeh Mirmahaleh et al.,"Flow mapping and data distribution on mesh-based deep learning accelerator," in Proc. 13th IEEE/ACM NOCS, 2019

[46] X. Wang, T. Mak, M.Yang, Y. Jiang, and et al., "On self-tuning networks-on-chip for dynamic network-flow dominance adaptation," in Proc. 7th IEEE/ACM NOCS, 2013, pp. 1–8.

[47] A. Paszke and et al., "PyTorch: An imperative style, high-performance deep learning library," in Advances in Neural Inf. Process. Syst., 2019, pp. 8024-8035.

[48] A. B. Kahng, B. Lin, and S. Nath, "ORION3.0: A comprehensive NoC router estimation tool," IEEE Embedded Systems Letters, vol. 7, no. 2, pp. 41–45, 2015.

[49] B. Moons and M. Verhelst, "A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," in Proc. Symp. VLSI, 2016, pp. 1–2

[50] Xiaola Lin, P. K. McKinley and L. M. Ni, "Deadlock-free multicast wormhole routing in 2-D mesh multicomputers," IEEE Trans. on Parallel and Distrib. Syst., 1994, pp. 793-804

[51] M. P. Malumbres, J. Duato and J. Torrellas, "An efficient implementation of tree-based multicast routing for distributed shared-memory multiprocessors," in Proc. 8th IEEE Symp. Parallel and Distrib. Process., 1996, pp. 186-189

[52] N. E. Jerger, L. Peh and M. Lipasti, "Virtual Circuit Tree Multicasting: A Case for On-Chip Hardware Multicast Support," 2008 International Symposium on Computer Architecture, Beijing, 2008, pp. 229-240

[53] M. Ebrahimi et al., "An efficent dynamic multicast routing protocol for distributing traffic in NOCs," in Proc. Des., Automat. & Test Europe Conf. & Exhib., 2009, pp. 1064-1069

[54] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," in Computers, 1993, pp. 62-76

[55] V. V. Vazirani, "Approximation algorithms", in SpringerVerlag, 2003

[56] L. Wang, Y. Jin, H. Kim and E. J. Kim, "Recursive partitioning multicast: a bandwidth-efficient routing for networks-on-chip," in Proc. 3rd Intl. Symp. Networks-on-Chip,2009, pp. 64-73

[57] S. Ma, N. E. Jerger and Z. Wang, "Supporting efficient collective communication in NoCs," in Proc. Intl. Symp. High-Performance Comput. Archit., 2012, pp. 1-12

[58] J. Hestness, B. Grot, S. W. Keckler, "Netrace: dependency-driven, trace-based network-on-chip simulation." in Proc. 3rd Intl. Workshop Network on Chip Architectures (NoCArc), 2010.

[59] H. Esmaeilzadeh, A. Sampson, L. Ceze and D. Burger, "Neural Acceleration for General-Purpose Approximate Programs," 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, Vancouver, BC, Canada, 2012, pp. 449-460, doi: 10.1109/MICRO.2012.48.

[60] Ying Wang et al., "A many-core accelerator design for on-chip deep reinforcement learning," in Proc. 39th ICCAD, 2020, pp. 1-7.

[61] C. Sun et al., "DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling," in Proc. 6th IEEE/ACM NOCS, 2012, pp. 201-210

[62] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm," Integr., VLSI J.,2017, vol. 58, pp. 74–81

[63] G. Singh et al., "A Review of Near-Memory Computing Architectures: Opportunities and Challenges," 2018 21st Euromicro Conference on Digital System Design (DSD), 2018, pp. 608-617

[64] H. Jia, M. Ozatay et al.,"15.1 a programmable neural-network inference accelerator based on scalable in-memory computing," in 2021 IEEE International Solid- State Circuits Conference (ISSCC), 2021, pp. 236–238

[65] K. Zhou, Y. He, R. Xiao, J. Liu and K. Huang, "A Customized NoC Architecture to Enable Highly Localized Computing-on-the-Move DNN Dataflow," in IEEE Transactions on Circuits and Systems II: Express Briefs, 2022, pp. 1692-1696

[66] L. Song, X. Qian, H. Li and Y. Chen, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017, pp. 541-552

[67] M. A. Akbar, B. Wang and A. Bermak, "Self-Repairing Carry-Lookahead Adder with Hot-Standby Topology Using Fault-Localization and Partial Reconfiguration," in IEEE Open Journal of Circuits and Systems, doi: 10.1109/OJCAS.2022.3161873.

[68] L. Pilato, S. Saponara and L. Fanucci, "Performance of digital adder architectures in 180nm CMOS standard-cell technology," 2016 International Conference on Applied Electronics (AE), 2016, pp. 211-214, doi: 10.1109/AE.2016.7577275.

[69] S. Shriram, J. Ajayan, K. Vivek, D. Nirmal and V. Rajesh, "A high speed 256-bit carry look ahead adder design using 22nm strained silicon technology," 2015 2nd International Conference on Electronics and Communication Systems (ICECS), 2015, pp. 174-179, doi: 10.1109/ECS.2015.7124884.

[70] R. Rogenmoser, L. O'Donnell and S. Nishimoto, "A dual-issue floating-point coprocessor with SIMD architecture and fast 3D functions," 2002 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.02CH37315), 2002, pp. 414-415 vol.1, doi: 10.1109/ISSCC.2002.993108.

[71] S. Payer et al., "SIMD Multi Format Floating-Point Unit on the IBM z15(TM)," 2020 IEEE 27th Symposium on Computer Arithmetic (ARITH), 2020, pp. 125-128, doi: 10.1109/ARITH48897.2020.00027.

[72] L. Crespo, P. Tomás, N. Roma and N. Neves, "Unified Posit/IEEE-754 Vector MAC Unit for Transprecision Computing," in IEEE Transactions on Circuits and Systems II: Express Briefs, doi: 10.1109/TCSII.2022.3160191.

CURRICULUM VITAE

Binayak Tiwari, Ph.D.
binayaktiwari@gmail.com

Education

Ph.D. in Electrical Engineering, 2022
University of Nevada, Las Vegas

Bachelor in Electronics and Communication Engineering, 2013
Tribhuvan University, Nepal

Honors and Awards

- UNLV College of Engineering Poster Competition 2022 - 3rd Place
- UNLV Outstanding Graduate Student Teaching Assistant Nomination by College of Engineering 2020-21
- UNLV Graduate & Professional Student Research Forum 2020 - 1st Place Poster – Science Poster Session
- A Richard Newton Young Student Fellow Award Design Automation Conference (DAC) 2020

Publication

- B. Tiwari, M. Yang, X. Wang, and Y. Jiang, "Effect of hardware trojan attacks on the performance of On-Chip multicast routing algorithms," in The IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC 2019), USA, Jan. 2019.
- B. Tiwari, M. Yang, X. Wang, and Y. Jiang, "Efficient On-Chip Multicast Routing based on Dynamic Partition Merging," in The 28th Euromicro Intl. Conf. on Parallel, Distributed and Network-Based Processing (PDP 2020), Sweden, March 2020.
- B. Tiwari, M. Yang, X. Wang, and Y. Jiang, "Improving the Performance of a NoC-based CNN Accelerator with Gather Support," in Design Automation Conference (DAC'20), Virtual, July, 2020 [Poster]
- B. Tiwari, M. Yang, X. Wang, Y. Jiang and V. Muthukumar, "Improving the Performance of a NoC-based CNN Accelerator with Gather Support," in The 33rd IEEE SOCC, September 2020

- B. Tiwari, M. Yang, X. Wang, and Y. Jiang, "Data Streaming and Traffic Gathering in Mesh-based NoC for Deep Neural Network Acceleration," in Journal of System Engineering, 2022, https://doi.org/10.1016/j.sysarc.2022.102466.
- B. Tiwari, M. Yang, X. Wang, and Y. Jiang, "In-Network Accumulation: Extending the role of NoC for DNN acceleration," under review

Dissertation Title

EFFICIENT NETWORKS-ON-CHIP COMMUNICATION SUPPORT SOLUTIONS FOR DEEP NEURAL NETWORK ACCELERATION

Dissertation Examination Committee

Chair, Mei Yang, Ph.D.
Committee Member, Yingtao Jiang, Ph.D.
Committee Member, Henry Selvaraj, Ph.D.
Graduate College Representative, Mingon Kang, Ph.D.