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

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

Yingtao Jiang, Ph.D.
Examination Committee Member

Henry Selvaraj, Ph.D.
Examination Committee Member

Mingon Kang, Ph.D.
Graduate College Faculty Representative

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,

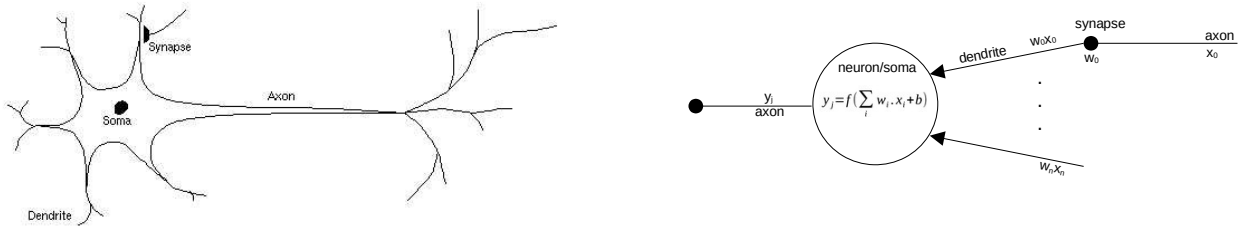

Figure 1.1: Neuron model (a) biological model, (b) mathematical model, weighted sum in a neuron x,w,f,b are input activations, weights, activation function, and bias, respectively (figures adopted from [6])

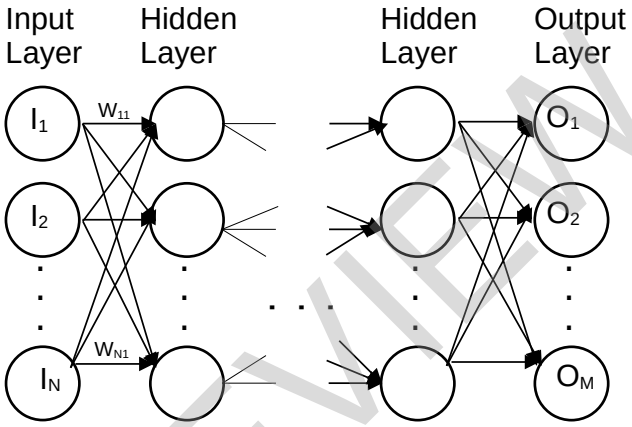

Figure 1.2: Example DNN model.

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

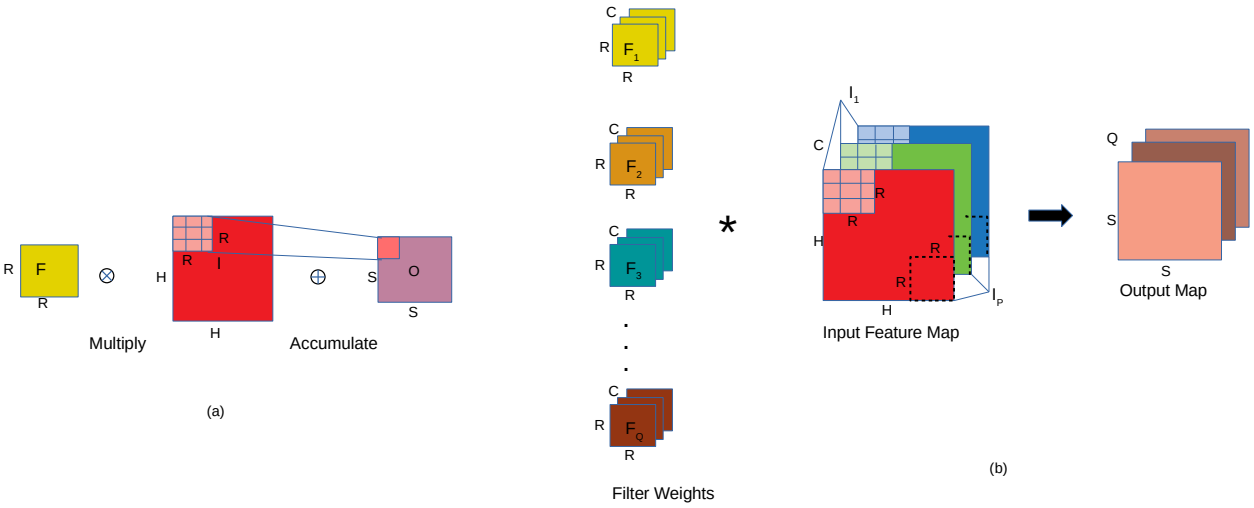

Figure 1.3: Convolution operation (a) 2D convolution, (b) higher dimensional convolution

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

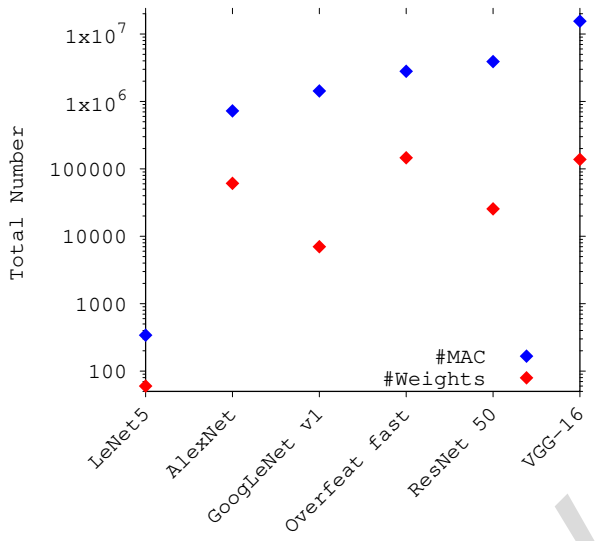

Figure 1.4: Some popular DNN models with number of weights and MAC operations.

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