Journal of Systems Architecture 126 (2022) 102466

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Data streaming and traffic gathering in mesh-based NoC for deep neural network acceleration

Binayak Tiwari [a], Mei Yang [a,*], Xiaohang Wang [b], Yingtao Jiang [a]

[a] Department of Electrical and Computer Engineering, University of Nevada, Las Vegas, NV 89154, United States of America
[b] Southern China Institute of Technology, China

ARTICLE INFO

Keywords:
NoC
DNN
Accelerators
Collective communication
Neural networks

ABSTRACT

The increasing popularity of deep neural network (DNN) applications demands high computing power and efficient hardware accelerator architecture. DNN accelerators use a large number of processing elements (PEs) and on-chip memory for storing weights and other parameters. As the communication backbone of a DNN accelerator, networks-on-chip (NoC) play 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 support the efficient one-to-many and many-to-one traffic largely existing in DNN workloads. In this paper, we propose a modified mesh architecture with a one-way/two-way streaming bus to speedup one-to-many (multicast) traffic, and the use of gather packets to support many-to-one (gather) traffic. 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 Output Stationary (OS) dataflow architecture. The simulation results demonstrate that the gather packets can reduce the runtime latency up to 1.8 times and network power consumption up to 1.7 times, compared to the repetitive unicast method on modified mesh architectures supporting two-way streaming. Furthermore, the comparison with state-of-the-art mesh-based accelerator shows that the proposed gather supporting scheme has the advantages in both area efficiency and power efficiency.

1. Introduction

Deep Neural Networks (DNNs) are widely adopted for a variety of applications, ranging from speech recognition, object detection and self-driving cars, to cancer detection, drug discovery, and genomics [1–3]. DNNs are able to extract high-level features from input data, as in statistical learning, compared to hand-picked features from classic machine learning. This enables DNNs to achieve human-level accuracy, which comes at the cost of high communication and computation complexity. High complexity in DNNs is attributed to the huge number of parameters and multiply-and-accumulate (MAC) operations. Fig. 1 shows the number of weights and MAC operations used in some of the popular DNN models. AlexNet [4] consists of 61M weights and 724M MACs, while VGG-16 [5] consists of 138M weights and 15.5G MACs.

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 the process of taking the inputs from the user or sensor and make 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 also a time-consuming process that may take up to several weeks at a cloud/data center. On the other hand, an inference can be employed even on edge devices like mobile phones. Due to the involvement of a huge amount of parameters (weights and biases Fig. 1), it is not possible to store all of these parameters in the local memory of the processing elements (PEs). Different levels of memory, from DRAM with high access cost to register files with low access cost [6], are commonly used in DNN accelerators, which imposes the challenge of optimizing data movement in the memory hierarchy.

In a DNN accelerator, PEs perform MAC operations, while the involved parameters are usually stored in the global memory. There is a need to transfer data from global memory to PEs, and vice versa. PEs and the memory elements are often interconnected by a Networks-on-Chip (NoC) [7–9] to realize 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 spatial architectures.

As the communication backbone [10–12] of a DNN accelerator, NoC plays an important role in supporting various traffic patterns and dataflow models, enabling processing with communication parallelism

* Corresponding author.
E-mail addresses: btiwari@unlv.nevada.edu (B. Tiwari), mei.yang@unlv.edu (M. Yang), xh.wang@giat.ac.cn (X. Wang), yingtao.jiang@unlv.edu (Y. Jiang).

https://doi.org/10.1016/j.sysarc.2022.102466
Received 2 June 2021; Received in revised form 9 March 2022; Accepted 10 March 2022
Available online 19 March 2022

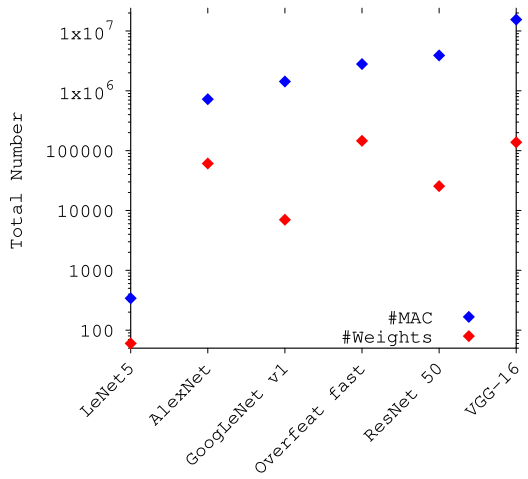

Fig. 1. Some popular DNN models with number of weights and MAC operations.

and enhancing scalability. NoC also offers a modular design property that helps in power gating. Mesh is a widely adopted NoC topology that is scalable and can support a large number of PEs. The artificial intelligence (AI) computing system from Cerebras [13] uses 2D mesh as a communication topology to connect thousands of AI cores. Groq [14] reorganizes 2D mesh into functional slices to optimize the microarchitecture. Existing mesh-based accelerator systems focus more on improving scalability and data reuse, and a little attention is given to enhancing communication support. Hence, this study is focused on efficient communication support for mesh-based DNN accelerators.

Observing the nature of data traffic in DNN processing, there are many inputs/weights transferred from the global memory to PEs and results (like partial sums) collected from PEs to the global memory. This traffic can be classified as one-to-many (multicast) and many-to-one (gather) traffic, respectively. In the literature, different approaches have been proposed to support multicast traffic in NoCs [12,15]. Noticeably, in a DNN workload, the multicast traffic tends to have a fixed communication pattern. Thus, existing multicast algorithms may not be suitable for DNN workloads. In addition, support of gather traffic has not been well addressed in NoCs, as many-to-one traffic rarely occurs in conventional parallel workloads.

In this work, based on the observation that the multicast traffic in DNN workloads has the same source and destination set most of the time, we propose a modified mesh architecture with a one-way/two-way streaming bus to speedup the input and weight distribution in an NoC. In support of the many-to-one type of traffic on a mesh-based NoC, gather packets are used to efficiently deliver the partial sum results to the global memory. Streaming architectures and gather packets can be used on different dataflow models. To evaluate the performance of the proposed modified mesh-based architecture with gather support, analysis and simulations are conducted on the output stationary dataflow model using AlexNet [4], ResNet-50 [16], and VGG-16 [5] as a DNN workload, and compared with the conventional repetitive unicast method.

The remainder of the paper is organized as follows. Section 2 reviews dataflow models, NoC architectures, and communication support proposed for DNN accelerators. Section 3 provides the background and motivation behind this work. In Section 4, we describe the proposed methods and analyze data streaming architecture and gather support improvement. Section 5 presents the experimental results of the proposed methods. Section 6 concludes the paper.

2. Related work

DNN processing involves tens of layers and a large number of MAC operations using millions of parameters, which imposes tremendous throughput and energy-efficiency challenges to the computing platforms. Recently, spatial DNN accelerators like [17–19] are gaining attention, as they are optimized to handle DNN processing effectively. Commonly used in FPGA and ASIC based designs [6,17,20,21], spatial architectures use a distributed approach, which adopts a large number of PEs, each having its own control logic and limited local memory and shared global memory. Communication between PEs is allowed, which enables the data movement between them. 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 DNN processing.

The dataflow determines the processing order, as well as where data is stored and reused, i.e., the way data (i.e., inputs, weights, and partial sums) communication happens between the PE and memory element. In the literature, various dataflow models have been proposed [6], including Weight Stationary (WS), Output Stationary (OS), Row Stationary (RS), and No Local Reuse (NLR); each has its own memory usage and energy advantages. In the WS model [17], weights are stationary at the PEs, while the inputs and partial sums 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 [19]. The NLR model [22] focuses on increasing the size of the global buffer at the expense of a register file, and thus decrease the DRAM accesses. The RS model increases the reuse of all data types rather than focusing on the reuse of one type.

DNN workloads contain different types of communication traffic that manages the data movement, such as partial sums, weights, and inputs streams to and from the memory. In a DNN accelerator [23], data movement is costly in terms of energy, consuming around 50% of the total energy. In some cases, data movement can even increase the latency [18] due to the communication bottleneck. Although a bus-only based system has been proposed in some prior work, this kind of system will quickly become a bottleneck when the DNN size increases [24]. This observation leads to the works focused on the NoC architecture, and communication support of DNN accelerators [8,25–29].

Various studies have been done in NoC topology to accelerate a DNN workload [23,24,29,30]. In [29], a hierarchical Neu-NoC architecture that adopts a hybrid ring–mesh topology is proposed where multiple PEs are connected in a group of rings via a mesh topology. This structure reduces the communication distance and shows better performance against the bus and tree structures. Other research [30] 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 [25] is proposed to simulate spiking neural networks with torus network topology. In [24], the study examines different topologies and concludes that the mesh NoC is better for realizing spiking neural networks, compared with the tree, point-to-point, and bus-based structures. In [23], 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 a change in the communication cost and support of different traffic patterns. Another study [18] proposes a mesh-based interconnection network called a 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 system's need. The NeuronLink [26] 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.

Various routing methods are adopted to fulfill the communication needs, especially multicast and gather traffic, in a DNN accelerator. Research in [26] adopts XY routing for unicast traffic and a table-based

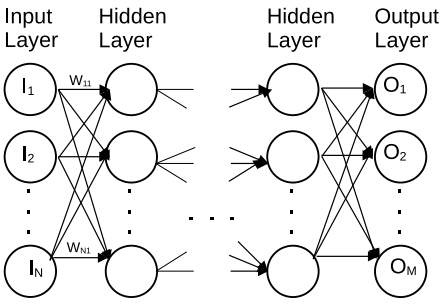

Fig. 2. Example DNN model.

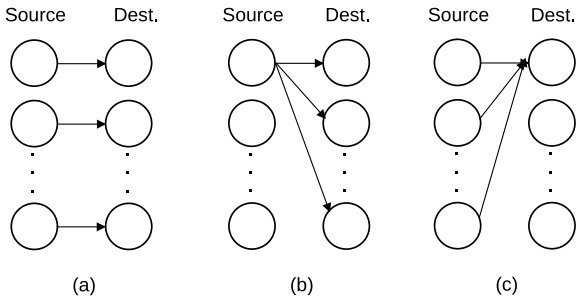

Fig. 3. Traffic pattern in a DNN accelerator (a) Unicast, (b) Multicast, (c) Gather.

routing for multicast traffic. However, another study [18] proposes different NoC configurations for each datatype i.e, input activation, weights, and partial sums. Further, this method is suitable for an 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 an OS dataflow model. Other research [31] propose an array of microswitches that are configured to handle different kinds of DNN traffic by creating a tree. In ClosNN [27], one or more layers can be executed on the network by mapping the neurons (PEs) on the input/output ports. Various stages of switching are done in order 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 maintain this pace. As widely adopted NoC topology, mesh is used in most of the recently proposed DNN accelerators [13,14,24,26,32]. As many-to-one and one-to-many traffic are not inherently supported in a mesh topology, one-to-many has many efficient solutions proposed in the literature [15] that can be well adopted to DNN workloads, but many-to-one does not have an efficient method. Recent work modifies the topology to simulate a tree or Clos network [27,31] to support many-to-one traffic, while other work [32,33] models this traffic as repetitive unicast (RU). In this work, we focus on providing communication support to this traffic on a mesh topology rather than proposing an alternate topology.

3. Background and motivation

Our study is focused on the inference phase of a feed-forward neural network. In general, the traffic patterns existing in a workload running on an NoC-based system significantly impact overall system performance [34]. Hence, it is important to study the nature of traffic in a DNN workload and provide a communication mechanism to support this traffic efficiently.

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 connect to neurons (activations) in another layer in full or in part via synapses (weights), as shown in Fig. 2. While implementing these layers in hardware, neurons are typically 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 is the input to the neurons in the adjacent layer. This sharing of data between adjacent PEs (neurons) creates traffic inside accelerators, which can be classified as one-to-one (unicast), one-to-many (multicast), and many-to-one (gather), as shown in Fig. 3.

Unicast traffic usually occurs when sending an input activation or weight from a memory element to a PE, or any other inter-PE traffic. Multicast traffic mainly distributes the filter weights from the memory element to multiple PEs. Different dataflow mechanisms support multicast traffic for weight distributions. Gather traffic collects the output from multiple PEs to the memory element. Due to the limited number of computing resources, multiple rounds are needed for the inference phase. When one round of MAC operations is completed, the intermediate results are gathered back to the memory element before initiating a new round.

The output of each neuron in Fig. 2 can be expressed as the operation shown in (1).

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

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 $\mathcal{F}()$ is the activation function in the model. Many DNNs consist of multiple layers, where both convolution and fully-connected layers perform MAC operations, as shown in (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 a costly task in terms of latency and energy [6].

When designing a DNN hardware accelerator, dataflow is another crucial aspect to consider. It depends on many factors such as the input size, as well as the number of kernels, stride, and mapping scheme of the DNN workload onto the PE arrays, along with DNN optimizations like pruning and sparsity [8]. 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 higher latency and energy inefficiency. Compared with other dataflow models, the OS dataflow model achieves good performance with less complexity. In this work, we analyze our proposed method using the OS dataflow model on a mesh-based NoC. Fig. 4 shows the OS dataflow model, 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.

Fig. 5 shows an example of how efficient communication support affects the performance in a many-to-one type of communication traffic. The green-colored nodes in a 6×6 mesh are trying to send the data to a memory element. Fig. 5(a) and (b) illustrates the delivery of data using unicast communication and the possible gather communication scheme, respectively. Using unicast communication, each node in the same row sends its packet to the same destination, which increases the amount of traffic. However, it is possible to reduce the network traffic when multiple senders are sending a payload to the same destination using a gather communication. With the gather support, the gather packet is initiated at the left-most node and collects the data payload from the intermediate nodes on its way to the destination node. As shown in Fig. 5(b), gather support significantly reduces the network traffic and reduces the total hop count from 15 to 5, proving to be efficient in delivering all data to the memory element using the least amount of resources. In addition, noticeably in DNN workloads, the weights and inputs used to calculate Eq. (1) are continuously streamed to certain groups of PEs, as shown in Fig. 4. In that sense, direct links

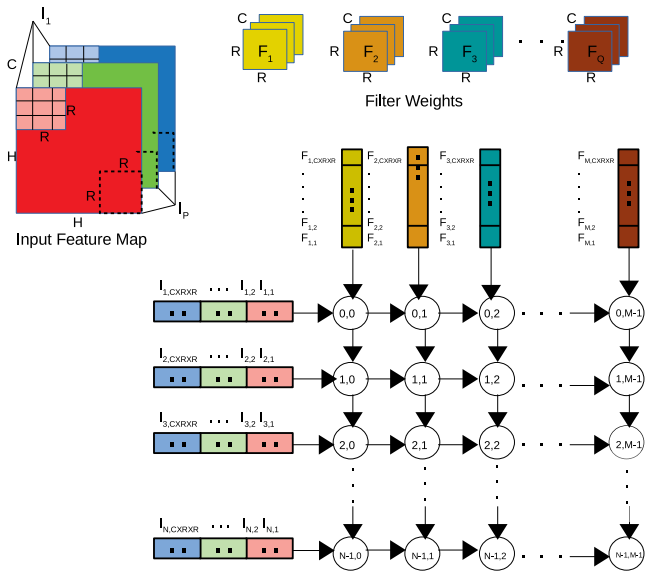

Fig. 4. Dataflow in NxM Mesh NoC.

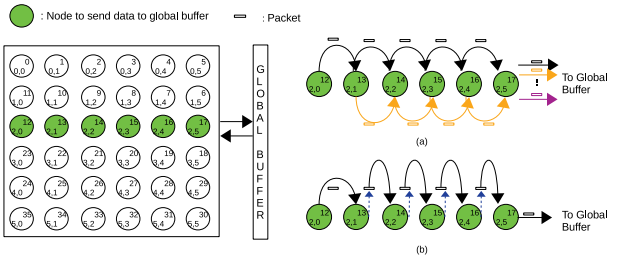

Fig. 5. 6 × 6 mesh example (a) without gather support, (b) with gather support. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

may be added for distributing weights/inputs, thus eliminating the unnecessary hop counts. As such, both many-to-one and one-to-many traffic can be supported in mesh-based NoC.

4. The proposed method

In this section, we describe the gather supported routing scheme first, followed by the data streaming architectures and a support on multiple PEs per router. We also present the analysis of the performance improvement in this section.

4.1. Gather support

To implement a convolutional layer on an NXM mesh-based NoC, multiple PEs perform MAC operations in a distributed fashion as shown in Fig. 4 where P input activations of size $H \cdot H$, each with C channels, are streamed in X-dimension. Similarly, Q filter or weight streams of size $R \cdot R$, each with C channels, are streamed in Y-dimension. Both the weight and input streams meet at all PEs in the respective dimension, to complete a total of $C \cdot R \cdot R$ MAC operations. As shown in Fig. 4, input activations and weights are streamed from the left and top side of the PE array respectively, so that the partial sums are generated in PEs. It is also clear from Fig. 4 that multiple rounds are required to complete all MAC operations due to the resource limitation in the network. When the first round of MAC operations are completed, the partial sum (PS) result, as shown in Eq. (2), is collected using gather traffic and sent to the global buffer before the start of the next round. Additionally, after the completion of one layer, output activations will also be moved to the memory element so that a new layer can operate on the PEs.

$$PS_{\substack{i,k \\ i\in\{1...P\};k\in\{1...Q\}}} = \sum_{j=1}^{C\cdot R\cdot R} (I_{i,j} \cdot F_{k,j}) \tag{2}$$

We propose to use gather-supported routing to enable many-to-one traffic. The leftmost PE in each row after completing the operation (PS or output activation) will initiate a gather packet, with the packet format as shown in Fig. 6. The packet consists of multiple fields including FT to identify the flit type (head, body, or tail); PT to identify the packet type (unicast, multicast, or gather); $ASpace$ to indicate the available space for the gather payload; and Src and Dst to indicate the source and destination address and $MDst$ for multicast destinations, respectively.

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 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. 9(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 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.

Algorithm 1: Flow for the Gather Routing

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 packet*
7 **if** *(F.FT = T)* **then** Load ← 0

4.2. Router architecture

An NoC router typically consists of multiple pipeline stages. Fig. 7 shows a four-stage router pipeline including stages of route computation (RC), virtual channel allocation (VC), switch allocation (SA), and switch traversal (ST) [35]. For an incoming packet, only the header flit undergoes the RC stage to determine the output port for the packet.

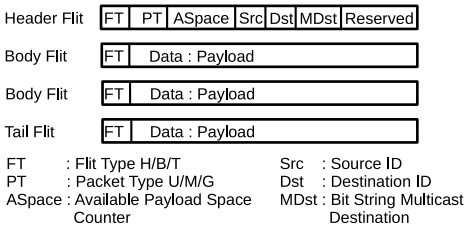

Fig. 6. Packet format.

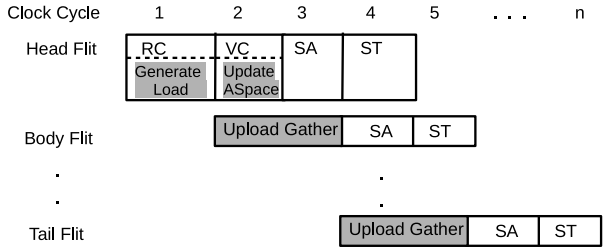

Fig. 7. Modified router pipeline.

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. 7 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 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. 8. The *Load* Signal Generator block, shown in Fig. 9(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. 9(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 packet upon the expiration of δ cycles controlled by a counter set by the PE.

4.3. Multicast support

As shown in Fig. 4, in the OS dataflow model, a partial sum will be accumulated at a PE with the input activations and weights streaming from the memory element [19,36]. Each router can connect to up to n PEs, so that these PEs will receive n sets of inputs and weights. Convolution operation accounts for a large portion of energy and latency in a DNN operation [6]. A significant portion of the communication traffic involves the distribution of input activations and weights, which can be treated as multicast traffic because of the dataflow pattern, as shown in Fig. 4. Based on this observation, we propose a modification to stream the input activations and weights using a bus from the memory

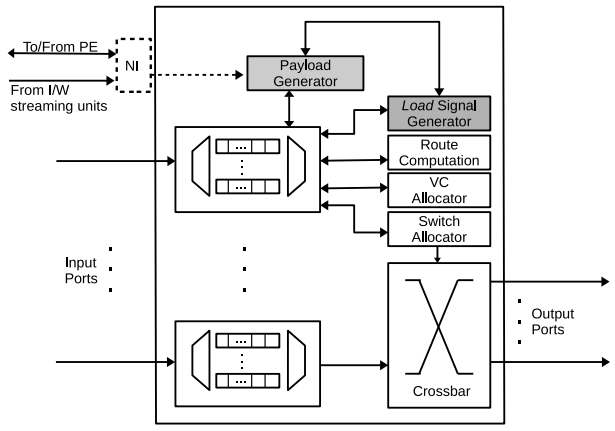

Fig. 8. Router microarchitecture.

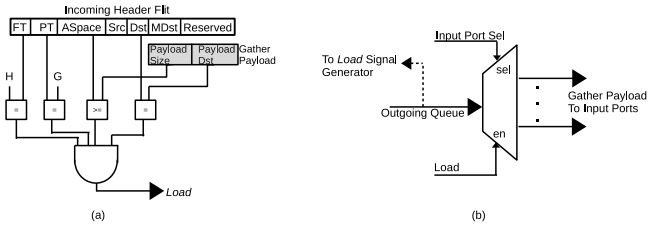

Fig. 9. (a) *Load* Signal Generator, (b) Payload Generator.

elements to the PEs in the same row and column. The streaming bus will help in overcoming the additional routing overhead, and thus, improve the runtime latency of a DNN workload.

Fig. 11(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 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. 4. 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.

Fig. 11(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. 11(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.

4.4. Multiple PEs per router

We also consider an expanded mesh, where multiple PEs connect to a router. Fig. 10 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) to the NI, as shown

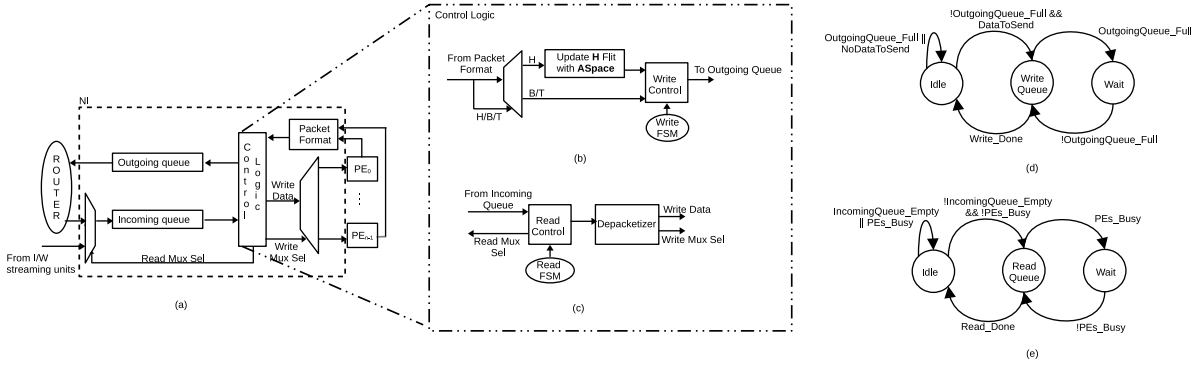

Fig. 10. Multiple PE/router (a) Network Interface Connection, (b) Outgoing Control Logic, (c) Incoming Control Logic, (d) Outgoing Queue Write FSM, (e) Incoming Queue Read FSM.

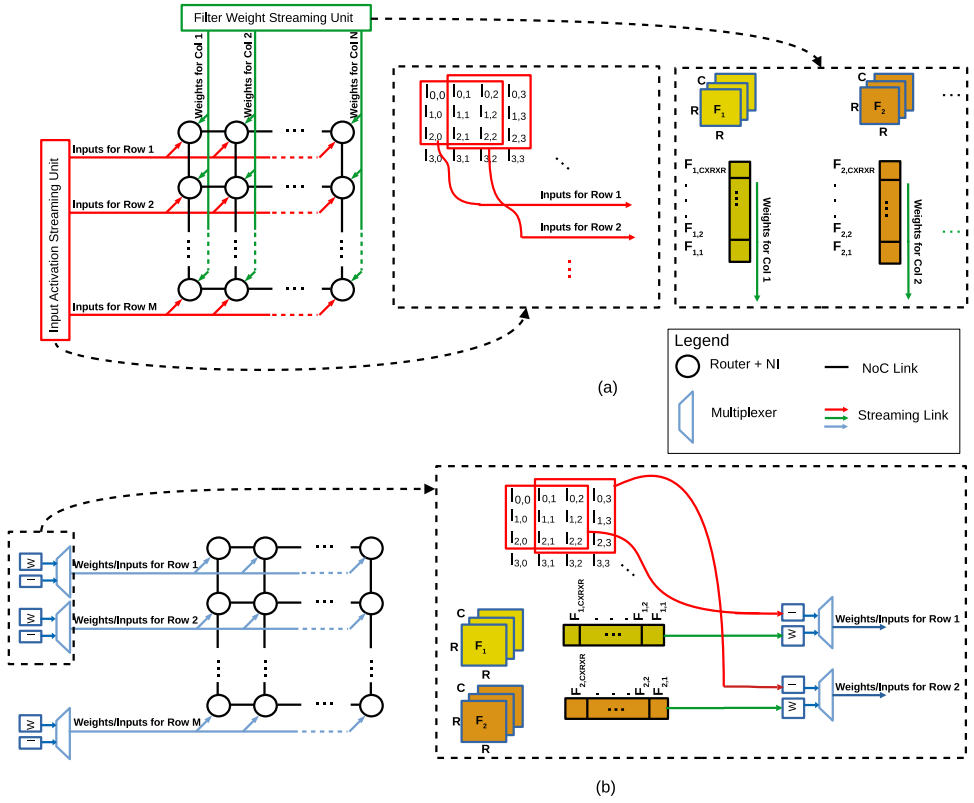

Fig. 11. Modified architecture with direct streams (a) Two-way streaming, (b) One-way streaming.

in Fig. 10(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 [37], 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. 10 performs multiple functions to ensure the correct flow of operation. It consists of two parts: the outgoing control logic (Fig. 10(b)) which serves as the interface from the PEs to the router and the incoming control logic (Fig. 10(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. 10(d) explains how the write FSM in Fig. 10(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. 10(e) 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. 4,

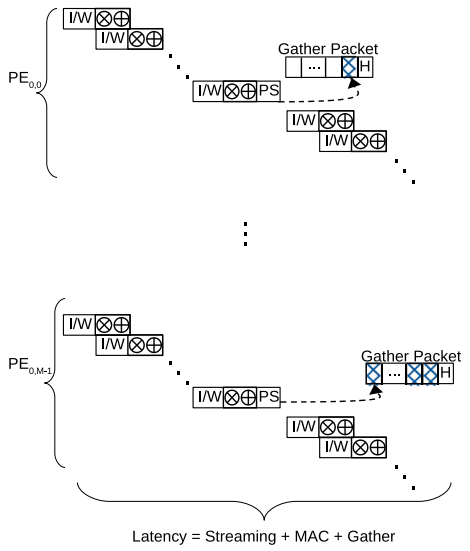

Fig. 12. Pipelined operation of a partial sum(PS) generation/gather in a row of PEs.

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 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, we have two different networks: one for gather traffic and the other for a streaming bus. In the mesh network, we use a credit-based flow control mechanism [35]. 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, we adopt a similar credit-based mechanism to that used in [31] to ensure the single-cycle data delivery to the PEs. The global buffer maintains the status of the credits for the PEs, i.e, incoming queue in the NI, as shown in Fig. 10(a). The streaming unit will only perform the streaming if all the nodes have free space to hold the data. This ensures the integrity of the MAC operation.

4.5. Analysis of proposed modification

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. 12 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 $CXRXR$ 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.

Eqs. (3)–(4) 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. 4; 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. 4.

$$Latency_{RU} = \left(\frac{C \cdot R \cdot R \cdot n}{f_l} + T_{MAC}\right)\frac{P}{N} \cdot \frac{Q}{M} \cdot \frac{1}{n} + M \cdot \kappa + \left\lceil \frac{L}{W} \right\rceil - 1 + \Delta_R \tag{3}$$

Eq. (3) 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.

$$Latency_G = \left(\frac{C \cdot R \cdot R \cdot n}{f_l} + T_{MAC}\right)\frac{P}{N} \cdot \frac{Q}{M} \cdot \frac{1}{n} + \sum_{i=0}^{\left\lceil \frac{M \cdot n}{\eta} \right\rceil - 1} \left((M - i \cdot \frac{\eta}{n}) \cdot \kappa + \left\lceil \frac{L'}{W} \right\rceil - 1\right) + \Delta_G \tag{4}$$

Eq. (4) 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, $\left\lceil \frac{M \cdot n}{\eta} \right\rceil$ 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. (3) and (4), 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. We will evaluate the effects of Δ_R and Δ_G through simulations in Section 5.2.

5. Performance evaluation

To evaluate the performance of the proposed method, we ran simulations for different CNN workloads and compared 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 the study of timeout period (δ) and gather packet size, before presenting the performance analysis.

5.1. Experiment setup

We 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 [17,19,26,27,33] 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 [38] to model the traces for the NoC. The

Table 1
Network configuration.

Topology	8 × 8 Mesh, 16 × 16 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

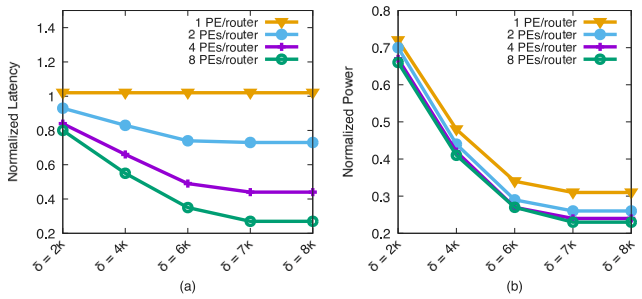

Fig. 13. Analysis of δ on 8 × 8 mesh for different number of PEs/router.

neurons are organized in a 2D mesh represented by the PEs. The total 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 [39] to simulate the generated traces for AlexNet [4], ResNet-50 [16], and VGG-16 [5]. The three different CNNs are chosen because of their diversity in the number of parameters and the number of convolution layers. Orion 3.0 [40] is used to estimate the power consumption for NoC, and DSENT [41] to estimate the power consumption for the streaming bus. We have performed the simulations on 8 × 8 and 16 × 16 2D mesh networks. Table 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 8 × 8 network; however, for a 16 × 16 NoC, two gather packets are needed, as the first one will be full halfway to the global memory.

5.2. Analysis of δ and gather packet size

The timeout period δ plays an important role in the performance of the gather 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. 13 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 8 × 8 mesh under a similar traffic scenario as in Fig. 5, 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 timeout occurs at those nodes. This piggyback mechanism will effectively decrease the network traffic and improve the resource utilization in the NoC.

As shown in Fig. 13(a), the normalized runtime latency (vs. $\delta < \kappa$) is reduced with δ value increased except for the case with 1PE/router, which is almost 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. 13(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 reduces 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, we see some significant improvement in the power consumption.

We further studied the tradeoff between different gather packet sizes for different network sizes and different number of PEs/router. Fig. 14 compares the performance of gather traffic using different numbers of gather packets for different numbers of PEs/router on 8 × 8 and 16 × 16 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. 14(a) and (b) show the normalized runtime packet latency and power consumption (vs. repetitive unicast) for 8 × 8 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 16 × 16 mesh is also shown in Fig. 14(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 Fig. 14(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 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

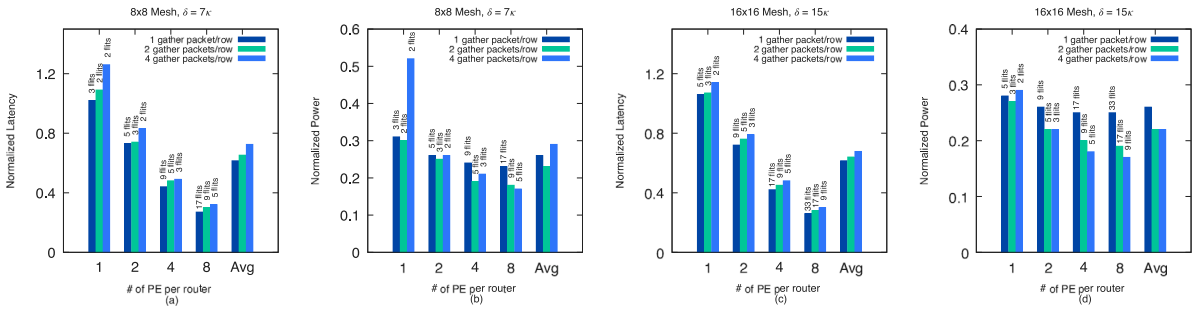

Fig. 14. Analysis of different gather packet size on 8 × 8 mesh (a),(b) and 16 × 16 mesh (c),(d) for different number of PEs/router.

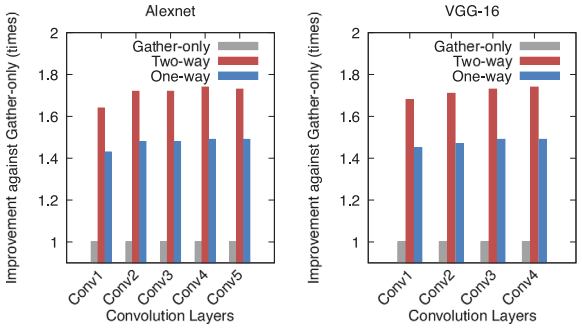

Fig. 15. Simulated improvement on the runtime latency of different convolution layers in AlexNet [4] and VGG-16 [5] over Gather-only [28].

method. To balance the tradeoff of latency and power consumption, in the following performance analysis, one gather packet is used for 8 × 8 mesh and 2 gather packets are used for 16 × 16 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.

5.3. Performance analysis

Fig. 15 shows the simulated runtime latency improvement of the proposed gather support with two-way streaming and one-way streaming architectures vs. gather-only using the NoC parameter from Table 1. On average, the gather support with two-way streaming architecture achieves 1.71× improvement, and the gather support with one-way streaming obtains 1.48× improvement compared against the gather-only method [28]. It is clear that the runtime latency improvement using two-way streaming is better than using one-way streaming in the OS dataflow model. Hence, we compare gather support against repetitive unicast with two-way streaming architecture in our experiments further.

Fig. 16(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 [4] on 8 × 8 and 16 × 16 mesh-based NoCs. It is clear that as the number of PEs is increased across 8 × 8 or 16 × 16 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 performed 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. 13(a) also shows the similar effect.

The performance improvement is higher in the case of 16 × 16 mesh when compared with the 8 × 8 mesh. For the 16 × 16 mesh, repetitive unicast traffic creates much higher congestion in the network, and the benefit of using gather traffic is more significant than on the 8 × 8 mesh. Fig. 17(a) and (c) show the improvement in total runtime latency for all convolution layers in ResNet-50 [16] for 8 × 8 and 16 × 16 meshes. Similarly, Fig. 18(a) and (c) show the improvement in total runtime latency for all convolution layers in VGG-16 [5] for 8 × 8 and 16 × 16 meshes. For both ResNet-50 and VGG-16, we see a similar trend in performance improvement, with the 16 × 16 mesh offering more improvement on VGG-16 (up to 1.84×) than the 8 × 8 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 routing more prominent.

Table 2
Hardware overhead.

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

Fig. 16(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 8 × 8 and 16 × 16 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×) 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 16 × 16 mesh, we can see that the improvement is slightly less than the 8 × 8 mesh, which is due to the increased number of gather packets for the same of PEs/router. Fig. 17(b) and (d) show the improvement in power performance for ResNet-50 [16] for 8 × 8 and 16 × 16 meshes and Fig. 18(b) and (d) show improvement in power performance for VGG-16 [5] for both meshes. For both models, we see a similar performance trend as in Fig. 16(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. Hardware overhead

We used DSENT [41] to estimate the area and power of a baseline router with the configuration shown in Table 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 2 shows the hardware overhead of the proposed router from Fig. 8 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

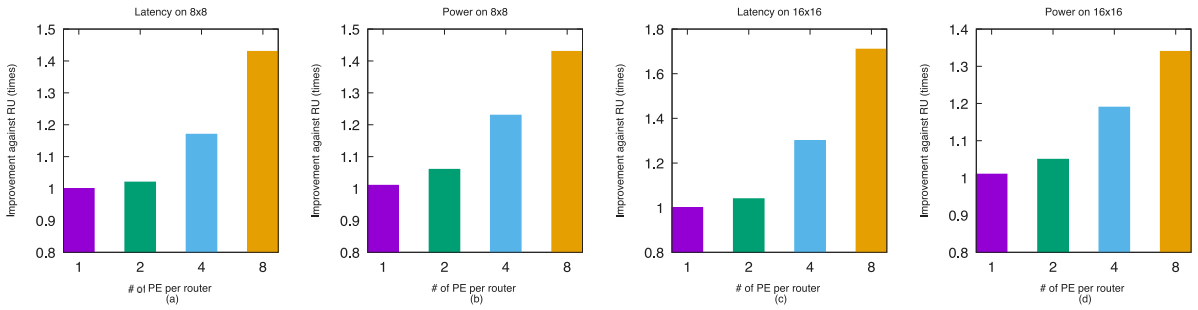

Fig. 16. Improvement on total runtime latency (a),(c) and power consumption (b),(d) for AlexNet [4] over RU for different number of PEs/router.

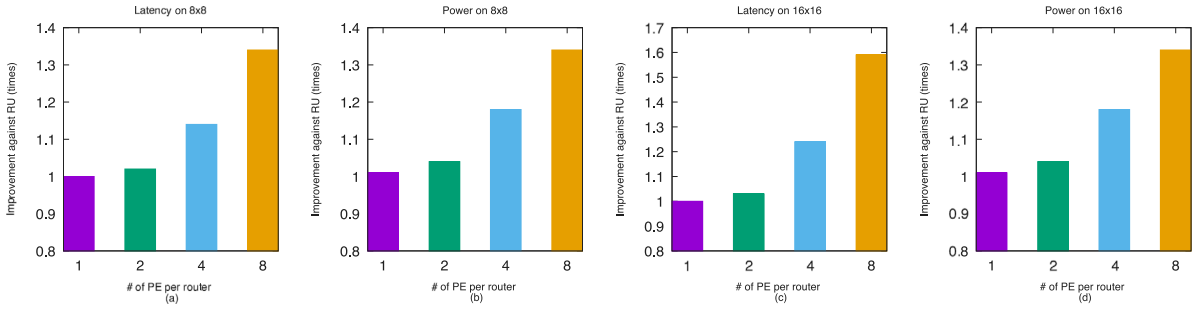

Fig. 17. Improvement on total runtime latency (a),(c) and power consumption (b),(d) for ResNet-50 [16] over RU for different number of PEs/router.

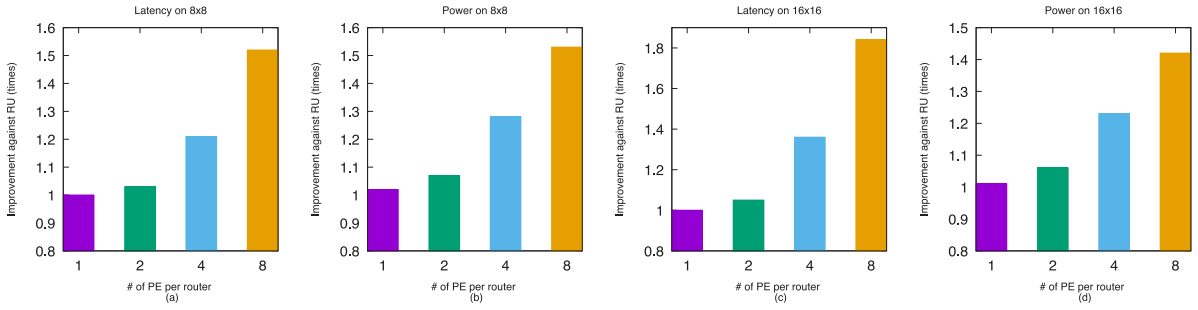

Fig. 18. Improvement on total runtime latency (a),(c) and power consumption (b),(d) for VGG-16 [5] over RU for different number of PEs/router.

Table 3
Comparison with NeuronLink [26].

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

When scaled to same technology parameter as ours.
[a]GOPS/mm^2 is 195.08.
[b]GOPS/W is 586.53.

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 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. 19 shows 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 3 shows the comparison of our proposed method with one of the most recent work NeuronLink [26]. Although, NeuronLink [26] does not support gather type traffic, both methods use mesh-based networks with similar routing method. We used the method from [42] to scale NeuronLink [26] to 45 nm node to compare with our work. The key metrics used to compare with NeuroLink [26] are area efficiency (GOPS/mm^2) and power efficiency (GOPS/W). The area efficiency and power efficiency of NeuronLink [26] when scaled to 45 nm are 195.08 and 586.53, respectively. Our work has the 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 [26].

6. Conclusion

In this paper, we proposed using the gather packet 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, ResNet-50, and VGG-16. The analysis

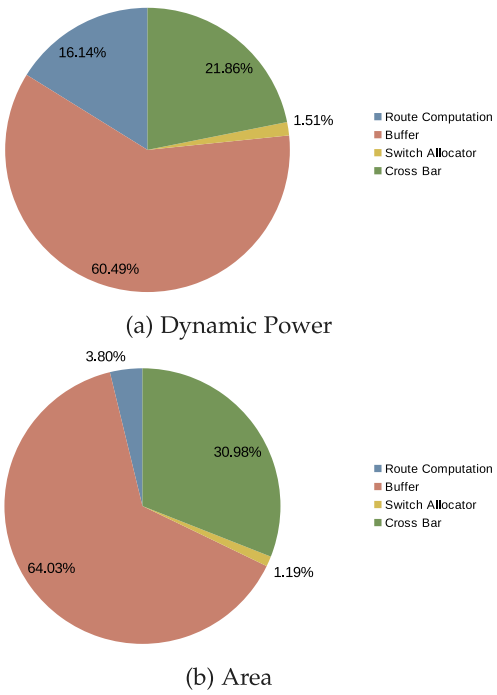

Fig. 19. Dynamic power and area breakdown of the proposed router.

shows that the two-way streaming architecture achieves 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× 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. Further, the proposed method supports all three kind of communication traffic necessary for a DNN workload in an area and power efficient way. Our method outperforms the most recent mesh-based accelerator by 2.04× and 1.17× in terms of area efficiency and power efficiency, respectively. Future work includes applying the proposed method to other dataflow models as well as thorough performance study of different dataflow models.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported in part by the National Science Foundation under grant no. 1949585. The authors would like to thank the reviewers for providing suggestions to improve the quality of the paper.

References

[1] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444, http://dx.doi.org/10.1038/nature14539.
[2] C. Chen, A. Seff, A. Kornhauser, J. Xiao, DeepDriving: learning affordance for direct perception in autonomous driving, in: Proc. IEEE Int'l Conf. on Computer Vision (ICCV), Santiago, 2015, pp. 2722–2730.
[3] A. Esteva, B. Kuprel, R. Novoa, et al., Dermatologist-level classification of skin cancer with deep neural networks, Nature 542 (2017) 115–118, http://dx.doi.org/10.1038/nature21056.
[4] A. Krizhevsky, One weird trick for parallelizing convolutional neural networks, 2014, CoRR, abs/1404.5997. [Online]. Available: http://arxiv.org/abs/1404.5997.
[5] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: Proc. ICLR, 2015.
[6] V. Sze, Y. Chen, J. Emer, A. Suleiman, Z. Zhang, Hardware for machine learning: challenges and opportunities, in: Proc. IEEE Custom Integrated Circuits Conf. (CICC), Austin, TX, 2017, pp. 1–8.
[7] W.J. Dally, B. Towles, Route packets, not wires: on-chip interconnection networks, in: Proc. 38th DAC, 2001, pp. 684–689.
[8] Hyoukjun Kwon, Ananda Samajdar, Tushar Krishna, MAERI: enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects, in: Proc. ASPLOS, 2018, pp. 461–475.
[9] S. Carrillo, et al., Scalable hierarchical network-on-chip architecture for spiking neural network hardware implementations, IEEE Trans. Parallel Distrib. Syst. 24 (2013) 2451–2461.
[10] B. Bohnenstiehl, et al., KiloCore: a 32-nm 1000-processor computational array, IEEE J. Solid-State Circuits 52 (4) (2017) 891–902.
[11] A. Touzene, On all-to-all broadcast in dense gaussian network-on-chip, IEEE Trans. Parallel Distrib. Syst. 26 (4) (2015) 1085–1095.
[12] B. Tiwari, M. Yang, Y. Jiang, X. Wang, Efficient on-chip multicast routing based on dynamic partition merging, in: Proc. 28th Euromicro Int'l Conf. on Parallel, Distributed and Network-Based Processing (PDP), Vasteras, Sweden, 2020, pp. 274–281.
[13] Cerebras Systems, Wafer-scale deep learning, in: Proc. IEEE Hot Chips 31 Symp. (HCS), Cupertino, CA, 2019, pp. 1–31.
[14] D. Abts, et al., Think Fast: a Tensor Streaming Processor (TSP) for accelerating deep learning workloads, in: Proc. ACM/IEEE 47th Annual Int'l Symp. on Comp. Architecture (ISCA), Valencia, Spain, 2020, pp. 145–158.
[15] A. Karkar, T. Mak, K. Tong, A. Yakovlev, A survey of emerging interconnects for on-chip efficient multicast and broadcast in many-cores, IEEE Circuits Syst. Mag. 16 (1) (2016) 58–72.
[16] K. He, X. Zhang, et al., Deep residual learning for image recognition, in: Proc. CVPR, 2016.
[17] N.P. Jouppi, et al., In-datacenter performance analysis of a tensor processing unit, in: Proc. 44th Int. Symp. Comp. Architecture (ISCA), 2017.
[18] Y. Chen, T. Yang, J. Emer, V. Sze, Eyeriss v2: a flexible accelerator for emerging deep neural networks on mobile devices, IEEE J. Emerg. Sel. Top. Circuits Syst. 9 (2) (2019) 292–308.
[19] Z. Du, et al., ShiDianNao: shifting vision processing closer to the sensor, in: Proc. ACM/IEEE 42nd Annual Int'l Symp. on Comp. Architecture (ISCA), Portland, OR, 2015, pp. 92–104.
[20] H. Sharma, et al., From high-level deep neural models to FPGAs, in: Proc. 49th Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), Taipei, 2016.
[21] E. Nurvitadhi, D. Sheffield, Jaewoong Sim, A. Mishra, G. Venkatesh, D. Marr, Accelerating binarized neural networks: comparison of FPGA, CPU, GPU, and ASIC, in: Proc. Int'l Conf. on Field-Programmable Technology (FPT), Xi'an, 2016.
[22] T. Chen, et al., DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning, in: Proc. ASPLOS, 2014.
[23] T. Luo, et al., DaDianNao: a neural network supercomputer, IEEE Trans. Comput. 66 (1) (2017) 73–88.
[24] D. Vainbrand, R. Ginosar, Network-on-chip architectures for neural networks, in: Proc. 4th Int'l Symp. Networks-on-Chip (NoCS), Grenoble, 2010, pp. 135–144.
[25] E. Painkras, et al., SpiNNaker: a 1-W 18-core system-on-chip for massively-parallel neural network simulation, IEEE J. Solid-State Circuits 48 (8) (2013) 1943–1953.
[26] S. Xiao, et al., NeuronLink: an efficient chip-to-chip interconnect for large-scale neural network accelerators, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 28 (9) (2020) 1966–1978.
[27] R. Hojabr, M. Modarressi, M. Daneshtalab, A. Yasoubi, A. Khonsari, Customizing Clos network-on-chip for neural networks, IEEE Trans. Comput. 66 (11) (2017) 1865–1877.
[28] B. Tiwari, et al., Improving the performance of a NoC-based CNN accelerator with gather support, in: Proc. IEEE 33rd Intl. System-on-Chip Conf. (SOCC), 2020.
[29] 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.
[30] A. Firuzan, M. Modarressi, M. Daneshtalab, M. Reshadi, Reconfigurable network-on-chip for 3D neural network accelerators, in: Proc. 12th Int'l Symp. Networks-on-Chip (NoCS), 2018, pp. 1–8.
[31] H. Kwon, A. Samajdar, T. Krishna, Rethinking NoCs for spatial neural network accelerators, in: Proc. 11th Int'l Symp. Networks-on-Chip (NoCS), 2017.
[32] Seyedeh Mirmahaleh, et al., Flow mapping and data distribution on mesh-based deep learning accelerator, in: Proc. 13th Int'l Symp. Networks-on-Chip (NoCS), 2019.
[33] Ying Wang, et al., A many-core accelerator design for on-chip deep reinforcement learning, in: Proc. 39th ICCAD, 2020, pp. 1–7.
[34] N.E. Jerger, L. Peh, M. Lipasti, Virtual circuit tree multicasting: a case for on-chip hardware multicast support, in: Proc. Int'l Symp. on Comp. Architecture (ISCA), Beijing, 2008, pp. 229–240.
[35] W. Dally, B. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
[36] B. Moons, 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.

[37] H. Esmaeilzadeh, A. Sampson, L. Ceze, D. Burger, Neural acceleration for general-purpose approximate programs, in: Proc. 45th Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), Vancouver, BC, Canada, 2012, pp. 449–460.

[38] A. Paszke, et al., PyTorch: An imperative style, high-performance deep learning library, in: 33rd Conf. on Neural Information Processing Systems (NeurIPS), Vancouver, Canada, 2019, pp. 8024–8035.

[39] X. Wang, T. Mak, M. Yang, Y. Jiang, et al., On self-tuning networks-on-chip for dynamic network-flow dominance adaptation, in: Proc. 7th Int'l Symp. Networks-on-Chip (NoCS), 2013, pp. 1–8.

[40] A.B. Kahng, B. Lin, S. Nath, ORION3.0: a comprehensive NoC router estimation tool, IEEE Embedded Syst. Lett. 7 (2) (2015) 41–45.

[41] C. Sun, et al., DSENT - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling, in: Proc. 6th Int'l Symp. Networks-on-Chip (NoCS), 2012, pp. 201–210.

[42] A. Stillmaker, B. Baas, Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm, Integr. VLSI J. 58 (2017) 74–81.