Rethinking NoCs for Spatial Neural Network Accelerators

Hyoukjun Kwon
Georgia Institute of Technology
Atlanta, Georgia
hyoukjun@gatech.edu

Ananda Samajdar
Georgia Institute of Technology
Atlanta, Georgia
anandsamajdar@gatech.edu

Tushar Krishna
Georgia Institute of Technology
Atlanta, Georgia
tushar@ece.gatech.edu

ABSTRACT

Applications across image processing, speech recognition, and classification heavily rely on neural network-based algorithms that have demonstrated highly promising results in accuracy. However, such algorithms involve massive computations that are not manageable in general purpose processors. To cope with this challenge, spatial architecture-based accelerators, which consist of an array of hundreds of processing elements (PEs), have emerged. These accelerators achieve high throughput exploiting massive parallel computations over the PEs; however, most of them do not focus on on-chip data movement overhead, which increases with the degree of computational parallelism, and employ primitive networks-on-chip (NoC) such as buses, crossbars, and meshes. Such NoCs work for general purpose multicores, but lack scalability in area, power, latency, and throughput to use inside accelerators, as this work demonstrates. To this end, we propose a novel NoC generator that generates a network tailored for the traffic flows within a neural network, namely scatters, gathers and local communication, facilitating accelerator design. We build our NoC using an array of extremely lightweight microswitches that are energy- and area-efficient compared to traditional on-chip routers. We demonstrate the performance, area, and energy of our micro-switch based networks for convolutional neural network accelerators.

CCS CONCEPTS

• **Computer systems organization** → **Interconnection architectures**; **Neural networks**;

ACM Reference Format:
Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2017. Rethinking NoCs for Spatial Neural Network Accelerators. In *Proceedings of NOCS '17, Seoul, Republic of Korea, October 19–20, 2017,* 8 pages.
https://doi.org/10.1145/3130218.3130230

1 INTRODUCTION

Neural network (NN) based algorithms, such as deep convolutional neural networks (CNN), have shown tremendous promise over the past few years in performing object detection, recognition, and classification across state-of-the-art benchmark suites [7, 15] at accuracies surpassing those of humans. Modern CNNs [14, 23] have tens of layers and millions of parameters. This adds tremendous

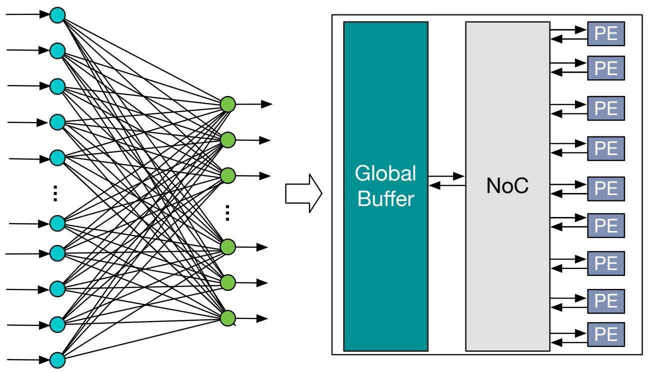

Figure 1: Architecture of a neural network accelerator. We generate the NoC between the global buffer and the PEs using novel latency, area, and energy-efficient microswitches.

throughput and energy-efficiency challenges on the hardware substrate to efficiently load these parameters on-chip and perform millions of computations (multiply-accumulate). CPUs are unable to provide such parallelism, while GPUs can provide high parallelism and throughput but consume massive amounts of energy due to the frequent memory accesses.

There has been flurry of research in the computer architecture domain for designing custom hardware accelerators for providing real-time processing of deep neural networks at stringent energy budgets. Most of these accelerators are spatial in nature, i.e., an array of interconnected processing elements (PEs) is used to provide parallelism [1, 3–5, 16]. The internal dataflow between the PEs is optimized to reuse parameters (input activations, weights, or output activations) that are shared by multiple neurons [3]. This reduces the number of memory accesses, thereby providing energy-efficiency. The PEs are fed new parameters from an on-chip global buffer, as shown in Fig. 1.

The microarchitecture of the PE (only compute, or compute with some local storage), and the nature of the dataflow between the PEs/global buffer to PEs is an area of active research currently and multiple alternate implementations have been demonstrated [1, 3–5, 16]. However, there has been little research on the architecture or implementation of the network-on-chip (NoC) interconnecting the PEs to each other and to the global buffer.

In a spatial NN accelerator, the NoC (Fig. 1) plays a key role, more so than in multi-cores, in realizing high-throughput. This is because most spatial accelerators operate in a dataflow style: a PE operation is triggered by data arrival, and the PE stalls if the next data to be processed is unavailable due to memory or NoC delay. Almost all NN accelerators have used specialized buses [3], or mesh-based NoCs [1, 4, 8], or crossbars [1], without a clear trade-off study on why one was picked over the other.

This work provides a comprehensive analysis of the NoC *within* an array of spatial PEs that are accelerating a CNN. We first characterize traffic inside typical CNN accelerator implementations, in detail. We demonstrate that interconnecting tens of processing cores on a CMP or SoC is very different from interconnecting hundreds of tiny PEs from both a performance (latency and throughput) and cost (area and power) perspective. We also argue that the NoC cannot be completely custom and static, such as those in application specific embedded systems, since the traffic patterns vary based on the actual neural network being mapped. We conclude that the conventional NoCs (bus/crossbar/mesh/custom-trees) used in CMPs/SoCs today are not appropriate choices as they either limit the achievable throughput from the PE array, or add significant area and power penalties, proportional to that of the PEs themselves.

We propose a new NoC design paradigm for NN accelerators using an array of reconfigurable *micro-switches*. The micro-switch array can be configured cycle by cycle to provide dedicated paths for three kinds of traffic that occurs in all CNN implementations: scatter (buffer to PE array - either via unicast, multicast or broadcast), local (PE to PE), and gather (PE array to buffer). We achieve three simultaneous goals: extremely low-area, energy-efficiency, and performance. The micro-switch array increases the performance of CNN implementations by 49% on average, compared to traditional (bus, tree, mesh, crossbar, hierarchical) NoCs. More importantly, it reduces average area and power by 48% and 39% respectively. In the same area, micro-switches can house 2.32X PEs than a mesh.

The rest of the paper is organized as follows. Section 2 provides relevant background on NN accelerator dataflows. Section 3 motivates this work by analyzing traffic flows across NN accelerator implementations. Section 4 demonstrates our micro-switch array topology and microarchitecture. Section 5 presents evaluation results. Section 6 describes related work and Section 7 concludes.

2 BACKGROUND

Neural networks are a rich class of algorithms that can be trained to model the behavior of complex mathematical functions. The structure of neural networks is modeled after the human brain with a large collection of “neurons" connected together with “synapses". In machine learning, a deep neural network (DNN) is built using multiple layers of neurons, each layer essentially acting as a feature extractor, with the final layer performing classification.

2.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) are a class of DNNs that are used widely for image processing. Each convolution layer receives inputs in the form of a raw image or input feature maps (the output of a previous convolution layer) and convolves it with a filter to produce an output feature map, as Fig. 2 illustrates.

2.2 Dataflows in CNN Accelerators

There has been a surge in accelerators for CNN over the past few years [4–6, 8]. The dataflow graph for CNN computations can be mapped onto a PE array in multiple ways leading to different dataflow characteristics. We follow the taxonomy introduced in Eyeriss [6] that classifies CNN accelerator implementations into the following categories.

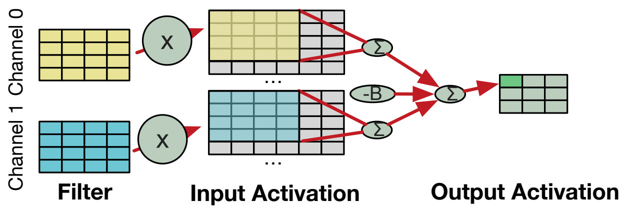

Figure 2: Convolution operation performed by a CNN accelerator. Input activations convolves with filter weights in respective channel. The accelerator accumulates bias values and partial sums generated from different channels to obtain output activations. The accelerator repeats such calculations over entire input activation by sliding the filter by stride value until it generates all the output activations.

- *Weight Stationary (WS):* In a WS accelerator, each PE fetches a unique weight element from the global buffer (GB) and retains it until the PE completes all calculations involving that weight. GB transfers input activations via a broadcast toward each PE. PEs may forward psums back to the GB (to be redistributed later), or accumulate them locally within the array.
- *Output Stationary (OS):* An OS accelerator maps one output pixel on to one PE in every iteration. Each PE fetches both weights and input activations from global buffer and internally accumulates partial sums. When the accumulation completes, or output activations are generated, each PE sends the output activation to the global buffer.
- *Row Stationary (RS):* A RS accelerator [6] maps a row of partial sum calculations on a column of the PE array, which facilitates data reuse of weight and input activations. Partial sums are accumulated by forwarding locally along the column, and the PEs at the top of the column send the final output activations to the global buffer.

The dataflows in CNN accelerators remain fairly uniform within each layer to maintain uniform utilization across all PEs.

3 MOTIVATION

3.1 Traffic inside Neural Network Accelerators

Given the highly parallel nature of the computation, most neural network accelerators - CNN [6, 8], RNN [10], SNN [1], - employ a multitude of compute units, which we refer to as processing elements (PE)s. Each PE contains some scratch pad memory and the compute logic. In addition to PEs there is also a larger on chip memory present in the accelerator, which we will refer to as the global buffer (GB). We assume the PE to be the most primitive building block of the accelerator - managing computation for one partial sum, a primitive output in CNNs.

We identify that there are primarily three kinds of traffic flows in spatial accelerators:

- **Scatter:** Scatter is data distribution from the GB to the PE array. Scatters can either be unicast or multicast, depending on the dataflow and the mapping of compute on PEs.
- **Gather:** Gather is the traffic flow which occurs when multiple PEs send data to the GB at a given interval of time. Gather can either occur at the end of the computation, or in the middle of the computation due to insufficient number of PEs.

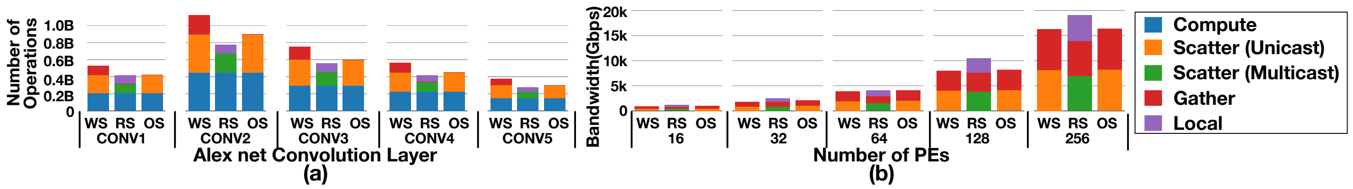

Figure 3: (a) Compute (Multiplications and Additions) vs. Communication (Scatter/Gather/Local) of each Alexnet layer [14] across different CNN implementations: weight stationary (WS), row stationary (RS), and output stationary (OS). (b) Average NoC bandwidth requirement for Alexnet vs. number of PEs

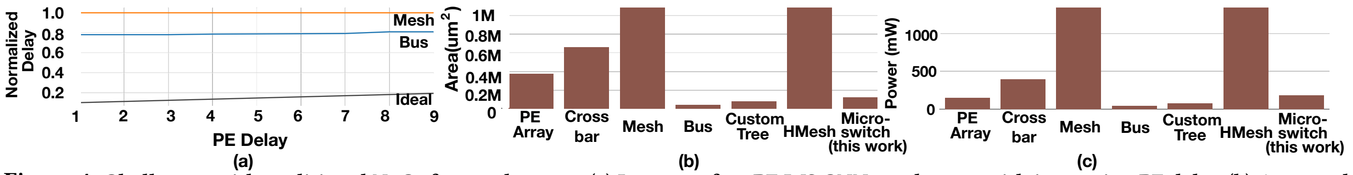

Figure 4: Challenges with traditional NoCs for accelerators. (a) Latency of 64-PE WS CNN accelerator with increasing PE delay (b) Area, and (c) Power

- **Local:** Local refers to the inter-PE communication traffic. It could be in the form of unicasts, multicasts or reductions.

Fig. 3 (a) plots the total number of computations and communication flows (scatter/gather/local) within the convolution layers of AlexNet [14] for the accelerator implementations described in Section 2.2. It shows raw compute to communication ratio across typical CNN dataflows, and demonstrates that communication is critical in spatial CNN accelerators to get full throughput. The operation of most of such accelerators occurs in a dataflow style; a delay in communication would essentially lead to a stall. Because PEs are tiny compute units, they are incapable of exploiting ILP/TLP mechanisms for hiding delays, unlike conventional CPUs or GPUs.

Fig. 3 (b) translates the raw communication into a bandwidth requirement as a function of the number of PEs. We simulate WS, OS and RS implementations and estimate the average traffic requirements across all the convolution layers of AlexNet. For all designs, scatter bandwidth (unicast for WS and OS, multicasts for RS) is extremely crucial. For WS architectures, the bandwidth required by gathers is significant. As the number of PEs increases, so does the bandwidth across all traffic flows.

The traffic analysis demonstrates that compute is highly dependent on communication in CNN accelerators. They require an interconnect to support frequent scatters, local traffic, and gathers, to multiplex many neurons across finite PEs. A neural accelerator designer would decide the number of PEs (based on the area budget on-chip), and optimize a PE microarchitecture for the target application and its required accuracy. The total traffic bandwidth divided by the PE delay determines the network bandwidth requirement per cycle to sustain full-throughput, by Little's Law. The NoC inside the accelerator needs to support this bandwidth. Next, we discuss the challenges with traditional NoCs for this purpose.

3.2 Traditional and Application-Specific NoCs

Traditional NoCs such as buses, meshes, and crossbars are common across multicores today. Naturally, they have also found their way into multi-PE DNN accelerators. For instance, Eyeriss [3] and DNNWeaver [22] use buses, DianNao [4] and ShiDianNao [8] use meshes, and TrueNorth [1] uses crossbars and meshes in a hierarchical manner. However, these NoCs add scalability challenges when used inside accelerators, as we discuss in section 5.

Application specific NoCs [19] generate NoCs in accordance with the application's communication graph that is known apriori and are common in MPSoCs in the embedded domain. They may seem natural as NoCs inside such accelerators to tailor the NoC to the neural network dataflow. However, the traffic inside the accelerator is not static; it varies layer by layer [6], and is dependent on the mapping of the dataflow over PEs, and the input parameters, as Fig. 3(a) shows. Nevertheless, we also implemented a tree-based custom NoC inside the accelerator optimized for scatters and gathers.

We perform a limit study with traditional NoCs with a WS dataflow. In this limit study, we assume no storage in PEs, which could mitigate the performance challenges of traditional NoCs at the cost of increased area and power in each PE.

Performance. Fig. 4 (a) plots the runtime across the CONV1 layer of AlexNet for a weight-stationary accelerator with 64 PEs. We compare the performance of a mesh, and a multi-bus/multi-tree topology against an “ideal" NoC which is a single-cycle zero-contention network. We make two observations.
(1) With a 1-cycle PE, we observe that the mesh and a single bus or tree is 10× slower than the ideal. The reason is heavy contention at links near the GB. Assuming that the GB can sustain higher injection/ejection bandwidth, we also simulated NoCs with multiple buses/trees and found that even with 64 buses, the design is 2× slower than the ideal.
(2) As the PE delay increases, normally the overall delay should increase as well, as we observe with the ideal. However, with the mesh or single bus/tree, the overall delay is almost constant demonstrating that the NoC is choked and is the bottleneck.

Area. Fig. 4 (b) plots the area of traditional NoCs relative to the area of 64-PE array. All numbers are from RTL synthesis with in 15nm Nangate PDK [18]. The PE array is from Eyeriss [6][1]. The most telling observation is that traditional scalable networks like mesh routers, prevalent across multicores, consume significantly

[1]We thank the Eyeriss authors for sharing the RTL implementation of a PE.

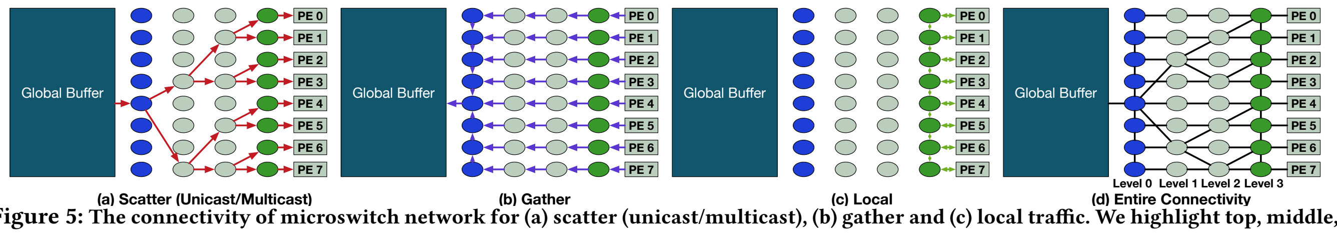

Figure 5: The connectivity of microswitch network for (a) scatter (unicast/multicast), (b) gather and (c) local traffic. We highlight top, middle, and bottom switches with blue, gray and green colors, respectively.

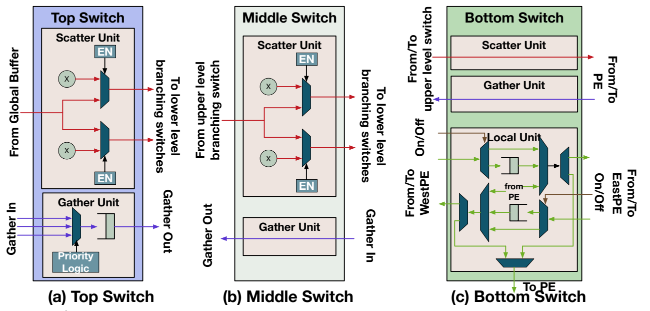

Figure 6: The microarchitecture of three microswitches.

higher area than even the compute PEs. This is primarily because routers in meshes are larger than a PE; utilizing a multicore mesh NoC inside an accelerator is thus not an efficient design choice as it would reduce the area available for the actual compute units, reducing overall throughput. Crossbars are known to scale horribly with number of nodes and this is also apparent from our area results. Buses and the custom tree are better in terms of area.

Power. Fig. 4 (c) shows a similar trend with power. Traditional meshes and crossbars end up consuming more power than the entire compute array. This becomes worse as the number of PEs increases.

Our conclusion from this study is two-fold:
(1) Meshes are not scalable solutions as NoCs inside accelerators. From a performance perspective, they get throughput limited when handling scatters and gathers. From an area and power perspective, routers consume much higher area and power than PEs.
(2) Buses and trees are effective for an area and power point of view, but they are non-configurable, which limit the performance of accelerators. That is, they are not flexible enough to support diverse demands of accelerators to support myriad CNN topologies, mappings, and input sizes.

4 MICROSWITCH NETWORK GENERATOR

Accelerators achieve high throughput and energy efficiency in *computation* by distributing computations to tiny processing elements, exploiting massive parallelism. Similarly, we achieve high network throughput and energy efficiency in *communication* by distributing communication to tiny microswitches. A microswitch consists of a small combinational circuit and up to two FIFOs; in contrast to the building blocks of traditional NoCs such as mesh routers that house buffers, a crossbar, arbiters and control. We describe the microswitch architecture in Section 4.2.

We design a NoC generator that aggregates multiple microswitches and connects them in our proposed topology to build a light-weight interconnect, that can be plugged into NN accelerators. Multiple microswitches can be traversed within a single-cycle, (24 switches within a GHz at 15nm, as we show later), enabling single-cycle communication inside the NoC. We call this $\text{MPC}_{\max}$ for maximum microswitches per cycle.

4.1 Topology

For a N PE design, we use a $Nlog(N)$ microswitch array, as shown in Fig. 5. We divide the array into $log(N)$ levels, with N microswitches each. We numerically label the switches from the global buffer side from Level0 to Level $log(N)$ The microswitches in Level 0 are called *top switches*, Levels 1 to $log(N) - 1$ are called *middle switches*, and Level $log(N)$ are called *bottom switches*. We layout our proposed topology over the array to efficiently handle three traffic flows that appears in any CNN implementation described in Section 3: scatter (unicast and multicast), gather, and local, as shown in Fig. 5.

Scatter (unicast and multicast). For scatters, we construct a tree structure in a microswitch array, with the root at one of the top switches, and the leaves at the bottom switches, as shown in Fig. 5(a). This simulates the functionality of bus: delivering data to multiple destinations simultaneously within a cycle. Unlike a bus that broadcasts data to every PE, however, our design delivers data only to designated recipient PEs (i.e., a unicast or a multicast). Such selective data delivery enhances energy efficiency by suppressing redundant broadcasting; implementation is lightweight, comprising of two one-bit registers in each branching switch and control signal propagation wires whose width is $2 \times (N - 1)$ when the number PEs is N. (i.e., N-1 one-bit registers and an 2(N-1)-bit wire). We discuss the control signal generation logic in detail in Section 4.5. Higher throughput from the global buffer is available by simply connecting to multiple top-switches, as we discuss later in this section.

Gather. For gather, each PE has dedicated connections up to the top switches in Level 0, via bypass links within the middle and bottom switches, as shown in Fig. 5(b). This provides high-bandwidth. Top switches send gather data towards one (or more) top switch connected to the global buffer's I/O port. The top switch connected to the global buffer I/O port selects one of the incoming gather flits using a round-robin-based priority logic and sends the flit to the global buffer in a pipelined manner.

Local. For PE to PE local traffic flows, we construct a bi-directional linear network using the bottom switches, as shown in Fig. 5(c). This network allows single-cycle traversals between any two PEs by controlling the microswitches appropriately. For example, if PE1 is communicating with PE2, PE3 with PE6, and PE7 with PE4, all of these can be supported simultaneously. We discuss this further in Section 4.4. The design thus minimizes the latency and maximizes the throughput of local traffic flows. The local traffic flow network is supported by the bottom switches using multiplexers and a FIFO, as discussed in Section 4.2. We manage the flow control using on/off reverse signaling in which each bottom switch latches incoming local flit if the buffer in the next bottom switch is not available.

Supporting Higher Bandwidth. The bandwidth of the scatter and gather networks is limited by the number of IO ports at the GB, which is logically visible from Fig. 5. The goal of a network architecture for accelerators has to be to make sure that the data delivery bandwidth does not become a bottleneck, which would lead to PEs stalling. As discussed earlier in Section 3.1, the required bandwidth depends not just on the traffic, but also one the delay and context state in each PE, which comes as an input to the microswitch NoC generator. We support higher bandwidth communication from the GB using wider channels and/or multiple parallel networks.

4.2 Microarchitecture

We define the *level* of a microswitch as the number of layers between that microswitch and the global buffer, as described in Fig. 5. Because the traffic pattern for the top (the first layer from the global buffer), middle, and bottom level of microswitch array varies, we present three types of microswitches for each.

Top switch. Top switches manage the gather and scatter (unicast and multicast), from PE to global buffer and vice versa respectively. Therefore, top switches contain two components: scatter and gather units, as shown in Fig. 6 (a). The *scatter unit* passes incoming flits to the branching nodes in the next level depending on the value of two one-bit control registers (one per scatter output port), determined by destinations of traversing flits. The traversal is completely bufferless, with flits branching to any one or both directions depending on the setup - unicast or multicast. The setup of the control registers is described in Section 4.5. The *gather unit* delivers incoming flits towards the global buffer I/O ports. There can be up to three gather flits entering a top microswitch, depending on its location, as Fig. 5(b) shows. A round-robin arbiter is used inside this unit. There is a an output FIFO after the arbiter to buffer the gather while it waits to win arbitration at the next microswitch.

Middle switch. Middle switches, which belong to the level between the first and last level, manage scatter and gather traffic, as shown in Fig. 6 (b). The *scatter unit* is the same as that in top switches; the *gather unit* is just a wire that simply forwards incoming gather flits toward top switches. Such a gather unit minimizes the latency of gather flits. However, if the number of PEs increases, gather flits need to traverse more number of microswitches in the middle layer. Then, we need to insert pipeline latches to meet the operating clock frequency, which is managed by our generator. Our synthesis results using NanGate 15nm standard cell library [18] shows that flits can pass 24 microswitches within a cycle (MPC_{max}) when the operating clock frequency is 1GHz, which allows 24 middle layers that cover 2^{24} PEs, a number large enough to cover state-of-the-art neural network accelerators. Note that most neural network accelerators today operate with a clock frequency lower than 1GHz and employ less than 256 PEs [3–5, 16]. *Thus our NoC can provide single-cycle traversals up to the top switches for gathers.*

Bottom switch. Bottom switches, which belong to the last level adjacent to PEs, manage scatter, gather, and local traffic. The scatter and gather units are wires. The local unit consists of a small number of components: three muxes, four demuxes, two FIFOs, and combinational logic that generates the mux/demux control signals, as shown in Fig. 6 (c). Although the number of components in a bottom switch is larger than that of components in top or middle switches, the overall overhead is not significant because the number of bottom switches increases linearly with the number of PEs. Local traffic units enable single-cycle multi-switch traversal, all the way from the source to the destination. Single-cycle multi-hop designs require extra control logic that introduces extra area and power overheads [13] to manage conflicts dynamically. We minimize such overheads by presetting microswitches to create multiple paths between PEs, as long as there are no conflicting links. We also allow flits to arbitrate for part of/the entire set of local links, like a bus.

We still require buffers in bottom switches for two reasons. (1) the buffer at the destination PE may be full; as a result the flit on the local network needs to wait. (2) the maximum number of microswitches to be traversed may be greater than MPC_{max}. Recall that our synthesis results at 15nm demonstrate a MPC_{max} of 24 at 1GHz. We force the bypassing flits to be latched after traversing MPC_{max} microswitches. The network interface between a PE and a bottom switch inserts a one-hot encoded bit vector that represents the number of remaining traversals. This value decreases via a simple shift in each bottom switch during traversal, with the signal getting latched when all bits are zero.

4.3 Routing

For scatters, the routing is predetermined by the microswitch control logic (Section 4.5). The control enables broadcasts, multicasts, and unicasts within a single cycle. For gathers, the route of all flows is fixed - from the PE to the GB. For local traffic, the NIC of source PEs inserts a one-hot bit vector representing the number of microswitches to traverse until the destination.

4.4 Flow Control

The three kinds of traffic use different flow-control strategies, as determined by the switches they traverse. The overall goal is to provide single-cycle communication for all three traffic types, at the maximum possible throughput.

Scatter. For scatters, we employ a customized cycle-by-cycle circuit switching technique that sets up unicast/multicast/broadcast paths that are valid for one cycle for each flit. This is done by the network controller, described later in Fig. 7. The global buffer maintains credits for the input buffers in the PEs, and performs a scatter only if all destination PEs have at least one free buffer.

Gather. For gather traffic, since the traffic passes through unidirectional wires in the middle switches, no flow control is required here. Top switches, however, need a flow control for gathers, since an arbitration grant plus an empty FIFO slot in the next top switch is required before a flit can be dequeued. We use on/off back signaling to support this.

Local. For local traffic, we support two schemes. (1) Static: the bottom switches are preset to enable multiple parallel circuit-switched connections between different PEs. This scheme depends on the mapping scheme across PEs and uses On/off back signaling between bottom switches. We discuss this scheme in Section 4.5. (2) Dynamic: part of or the entire set of local links can be arbitrated for and used like a bus.

4.5 Network Reconfiguration and Control

A key property of our microswitch network is cycle-by-cycle reconfigurability. The reconfiguration is controlled by one-bit control

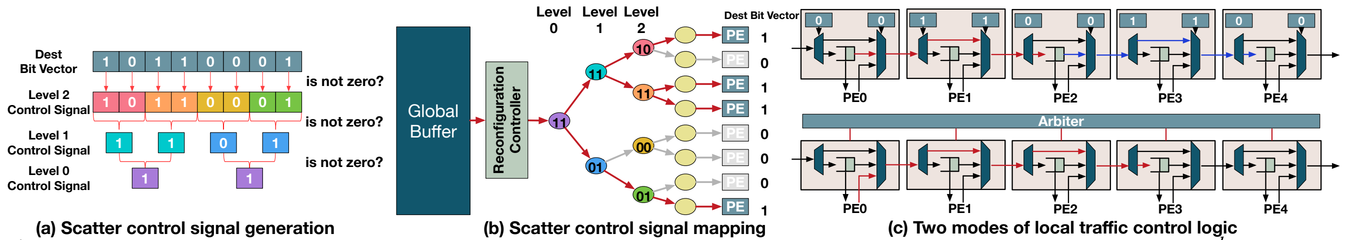

Figure 7: An example of scatter tree reconfiguration. (a) Control signal generation. The controller recursively tests a set of 2^k consecutive bits in a destination bit vector if is not zero until it reaches level 0. If a test bit vector is not zero, the corresponding switch is active. Therefore, the parent node switch at the lower level is active as well to provide data to the child switch. Our control logic is based on such an observation. (b) Control signal mapping for a multicast scatter. For simplicity, we only show microswitches that belong to the scatter tree. The 2-bits in each microswitch is the control register value, one for each branch of the sub-tree. For example, if the control register values are 10, incoming scatter flit is forwarded to the upper subtree in the figure. (c) Local traffic control. Mode 1 (Static) - Control register manage flow control; if the value is zero, an incoming flit stops at the bottom switch, else it bypasses. For example., this mode allows PE0 to PE2, and PE2 to PE4 communication simultaneously. Mode 2 (Dynamic) - An arbiter selects one flit and grants the flit access through multiple switches exclusively.

registers for muxes at each microswitch, to enable single-cycle traversals across the fabric over multiple microswitches. The top and middle switches can be configured for single-cycle scatters (unicast, multicasts, and broadcasts), and the bottom switches for single-cycle local traffic. Gather network uses conventional flow-control and delivers flits in a pipelined manner.

4.5.1 Control Signal Generation.

Scatter Network. The reconfiguration for scatters is controlled by two one-bit control registers in each middle and top switch that are branching nodes in the tree we construct, as the example in Fig. 7 shows. The value of control registers indicates if an incoming data may flow toward their corresponding sub branches of a branching node in the tree. The network controller converts destination bits of a flit into control register values and sends the register values one cycle before the data flit traverses the scatter tree. The reconfiguration and the data flit traversal are pipelined so the controller inserts a data flit at every cycle. That is, while a data flit traverses the tree network, the controller generates and sends a control signal for the next data flit. Therefore, the control logic does not degrade the overall throughput.

The controller receives a destination bit vector from the global buffer, which consists of N bits (N: number of PEs) that represents valid destinations, and generates a control signal that contains the value of control registers in branch switches of the scatter/broadcast tree. The control signal generation logic is based on the observation that each branching switch needs to send a flit toward a lower branch if the branch contains at least one of the valid destinations. That is, we can determine the control signal by examining two, four, and 2^k consecutive bits in a destination bit vector for the level $log(N) - k$, where N is the number of PEs and k is an integer between 0 and $log(N)$. We provide an example in Fig. 7(a). The logic checks if an individual bit in the destination bit vector is nonzero; the results are the control signals for the last level. In the next step, the logic checks if consecutive two-bit values are nonzero; the results are the control signals for the next level. The logic repeats to double the size of test consecutive bits and check if each chunk is nonzero until the test bit size covers the half of the destination bit vector. If the number of PEs is not a power of two (i.e., number of PEs $< 2^k$), the logic regards the destination bit width as 2^k and pads zeros for invalid destinations.

Local Network. On the local network, we provide the ability to partition the set of local links into single-cycle circuit-switched paths between any two PEs (subject to the number of switches being less than MPC_{max}). The local network configuration is done across larger time epochs rather than every cycle. For instance, for CNNs, this is done at the start of every convolutional layer. Since the network controller manages delivery of scatters, it also knows which PEs will communicate with which other PEs, and accordingly tries to provide neighbor-to-neighbor communication as much as possible, which can be supported in parallel, as shown in Fig. 7(c). Each bottom microswitch has 2-bits to determine whether incoming flits need to be forwarded to the next microswitch or stop. Flits that stop at a bottom switch are read by the appropriate PE if the destination matches. Thus the bottom switch allow the local links to form configurable buses of different lengths.

If partitioning the bus statically is not possible for handling all local communication flows simultaneously, say for fully-connected layers of CNNs, some/all bottom switches operate in a forward mode and the local links behaves like a bus via dynamic arbitration, as shown in Fig. 7(c). We also enable part of the local links to operate like an arbitrated bus, and the remaining to be statically configured. This is all managed by the reconfiguration controller.

4.5.2 Control Signal Mapping.

We utilize a separate control plane to configure each microswitch. Recall that each switch has a 2-bit configuration state. The number of bits in the control plane is a trade-off with reconfiguration time, and multiple implementations can exist. We support two:

Dedicated. We use $2 \times NlogN$ wires, to enable cycle by cycle reconfiguration. As an energy optimization, the controller only sends bits to switches that need to update their configuration. A challenge with this design is that the configuration plane may become too wide at large PE counts.

Ring. We also support an alternate design for the control plane where all switches are linked via a configuration ring (analogous to scan chains today) to carry a switch id and the 2-bit configuration. The controller sends configurations for each switch multiple cycles in advance, keeping the delay of traversing the ring in mind. This is possible since the dataflow is fixed after the mapping is complete.

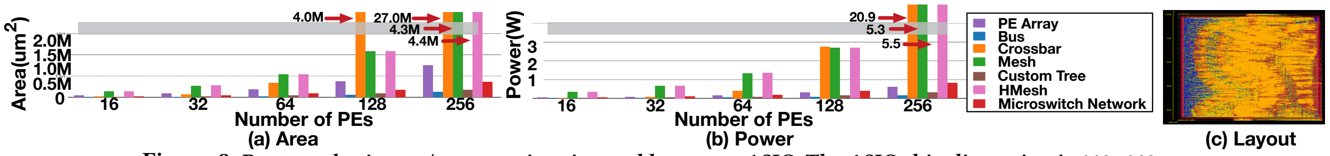

Figure 8: Post-synthesis area/power estimation and layout on ASIC. The ASIC chip dimension is 440x440um.

Network	Bus, Custom tree, Crossbar, Mesh, Hierarchical Mesh (4 clusters, 1X or 2X BW at GB), Microswitch
Language	Bluespec System Verilog (BSV) [20]
Technology	15nm NanGate PDK [18]
Traffic Patterns	WS (without local accumulation) and RS
Application	Alexnet (CNN) [14]

Table 1: A summary of evaluation configuration

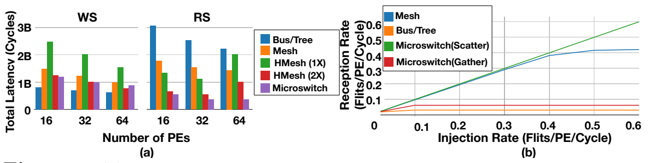

Figure 9: (a) Total latency of each accelerator and NoC combination for entire Alexnet. (b) Throughput evaluation of mesh, bus, and microswitch network with 32 PEs and randomized synthetic traffic.

5 EVALUATIONS

Table 1 presents our evaluation methodology and configurations. In addition to traditional NoCs, we also implement and evaluate the performance of hierarchical designs, which are popular in recent DNN accelerators [1, 5, 22]. We use a hierarchical mesh with four clusters (e.g., for 64 PEs, each cluster contains 16 PEs).

5.1 Area and power estimation

Fig. 8 presents the post-synthesis area and power results of our micro-switch NoC compared to a traditional NoCs. The first stark observation is that the mesh adds too much overhead, both in terms of area, and power, compared to even the PE array. The crossbar area and power are reasonable at 32-64 PEs, but it shoots up at large PE counts. The mesh and crossbar consume 7.4X more power and 7.2X more area compared to the PE array at 256 PEs. The bus[2], tree and micro-switch array are the most scalable for area and power. On average, the micro-switch array consumes 47.8% lower area and 39.2% lower power than all baselines. Assuming a 512B SRAM in each PE [3], we find that a micro-switched based accelerator can house 2.32X more PEs than a mesh in the same area.

5.2 Throughput and latency

Fig. 9 (a) presents the total latency for running Alexnet in WS and RS accelerators with 16, 32, and 64 PEs. Since multicast-scatter is dominant in weight-stationary traffic, as Fig. 3 shows, the bus and tree performs well with WS accelerators. However, as RS accelerators involve local traffic, the micro-switch network performs the best because it exploits the local traffic network between the bottom switches. Mesh performs the worst in every case because it needs to serialize all the scatter traffic. An optimization that clones a scatter flit in each router is feasible, but such an optimization demands more area and power. Considering the area and power overhead of mesh is already prohibitively high (Section 5.1), it is not practical even if such an optimization were to be applied. The HMesh with 2X bandwidth at the global buffer performs better than the mesh, but is still worse than the bus and microswitch which have 1X bandwidth due to the lack of multicast support. 'For RS traffic, the HMesh had worse performance because of mapping inefficiencies caused by the fixed size of clusters.

In Fig. 9(b), we compare the performance of the networks with synthetic random scatter/gather traffic. The performance of the microswitch scatter network scales linearly without saturating as it guarantees single-cycle traversal to multiple destinations via the single-cycle multiple-hop network. The microswitch gather network saturates early due to heavy congestion at the link going into the GB, and we recommend using multiple gather networks or wider links at the top switches to enhance throughput. The bus and tree networks saturate very early.

Fig. 10 shows the performance breakdown of the NoCs for running each layer of AlexNet. The micro-switch fabric provides the lowest runtime, a 49% savings on average across all NoCs, as it eliminates the scatter and/or gather bandwidth bottlenecks present in other NoCs.

5.3 Energy consumption

Since a bus always broadcasts flits to the PE array, it requires more energy for each flit. The worst case of such an inefficiency is unicast that has only one destination but bus consumes energy for broadcast. More number of PEs aggravate the energy inefficiency of bus, as Fig. 11 (a) highlights. The amount of energy required for single flit traversal affects the overall energy consumption of entire computation. The total energy consumption for Alexnet convolution layers in Fig. 11 (b) shows the micro-switch NoC being the most efficient in terms of overall energy as it activates only the required minimal links for each flit traversal, for both scatters and gathers.

In summary, we can observe that the micro-switch network performs well on all metrics - latency, throughput, area, power, and scalability, when used inside a neural network accelerator, while traditional NoCs fail on one or more of these fronts.

5.4 Bottom switch bypass for local traffic

Depending on the operating clock frequency, the number of bottom micro-switches a local traffic flit can traverse within a cycle i.e., MPC_{max}, varies, as shown in Fig. 11 (c). The MPC_{max} value affects the throughput of local traffic network based on the source-destination pattern. If an accelerator design requires end-to-end local traffic, then the delay of such local traffic flits is the number of PEs divide by MPC_{max}. However, assuming that the neural network mapping algorithm did a good job mapping communication PEs close to each other, such a worst case would be rare, and we expect most local traversals to take a single-cycle leveraging the single-cycle over MPC_{max}-hops feature of our micro-switch array. For example, a PE in an RS accelerator requires partial sums to

[2]Note that this is a post-synthesis result that does not take into account the RC of the final bus layout. Thus the observed power consumption is somewhat under-evaluated.

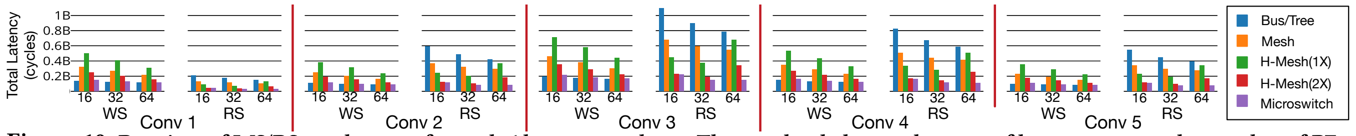

Figure 10: Runtime of WS/RS accelerators for each Alexnet conv layer. The number below each group of bars represents the number of PEs.

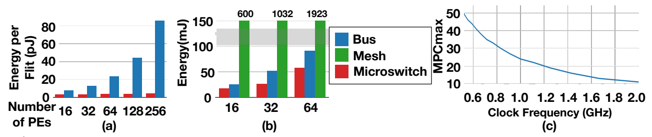

Figure 11: (a) Energy consumption for single flit traversal. (b) Total network energy for entire Alexnet convolution layers using an RS accelerator (c) MPC$_{max}$ over clock frequency values.

traverse to an adjacent PE in the same column of the PE array [3]. We can construct a column-first linear bottom switch network for it, and the number of bypass hops toward the destination is always one in such a configuration. This configuration works with either of the control logic we discussed in Section 4.5.

6 RELATED WORK

NoCs in neural networks accelerators. Vainbrand, et al. [25] compared interconnection networks in neural network accelerators from a theoretical perspective and concluded that a multicast mesh is the most optimized solution. The authors remarked that tree-based networks are not a good option for inter-PE communication, which we address in this work by providing local links in bottom switches. Theocharides *et al.* [24] and Emery *et al.* [9] suggested that mesh is the best NoC in neural network accelerators because of its scalability and ease of reconfigurability. Diannao [4] and Shidiannao [8] relied on mesh-based interconnects for data transfer. Dadiannao [5] employed fat tree for scatter and gather and 3D mesh via hyperTransport 2.0 to distribute data among nodes. Eyeriss [3] uses separate buses for its scatters and gathers. Carrillo *et al.* [2] described H-NoC infrastructure in which the neural nodes are arranged in three layers, module, tile and cluster. In H-NoC, any traffic moving from a lower to upper level is reduced, which maintains the scalability of the network. Spinnaker [11] implemented SNNs using clusters of multiprocessor nodes with a 2D triangular mesh for inter-node and a 2D mesh for intra-node communication. IBM Truenorth [1] employed a 256×256 crossbar for traffic inside a neurocore and mesh for inter-core communication. Our microswitch based NoC can enhance all these designs by lowering the area and power of the NoC, enabling more PEs to be added.

Lightweight and application-specific NoC design. NOC-out [17] designed a lightweight switch-based network and effectively addressed cache coherence traffic in many-core CMPs. Although both of NOC-Out and microswitch network are based on lightweight switches, microswitch network (1) exploits single-cycle multi-hop feature to implement multicast scatter, (2) allows multiple reduction points based on global buffer bandwidth, and (3) statically generate control signals to implement cycle-by-cycle circuit switching that eliminate the necessity of VC buffers, VC allocation, arbitration, credit management logic inside switches, which minimizes area and power. Ogras *et al.* [21] optimized general-purpose NoCs for selected applications by prioritizing application-specific traffic in selected long range paths. Kim *et al.* [12] employed a 2D mesh to address the multi-source-multi-destination traffic of augmented-reality applications in head-mounted displays with a scheduler for mapping the application over the NoC.

7 CONCLUSION

We present a novel NoC design for neural network accelerators that consists of configurable light-weight micro-switches. The micro-switch network is a scalable solution for all the four aspects - latency, throughput, area, and energy - while traditional NoCs (bus/mesh/crossbar) only achieve scalability for some of them. We also provide a reconfiguration methodology to enable single-cycle paths over multiple micro-switches to support dynamism across neural network layers, mapping methodologies and input sizes. While our evaluations focused on neural network accelerators, we believe that the micro-switch fabric can be tuned for any accelerator built using a spatial array of hundreds of PEs.

REFERENCES

[1] F. Akopyan *et al.*, *Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip*, TCAD (2015).
[2] S. Carrillo *et al.*, *Scalable hierarchical network-on-chip architecture for spiking neural network hardware implementations*, TPDS (2013).
[3] Y-H. Chen *et al.* , *Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks*, ISSCC, 2016, pp. 262–263.
[4] T. Chen *et al.*, *Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning*, ASPLOS, 2014.
[5] Y. Chen *et al.*, *Dadiannao: A machine-learning supercomputer*, MICRO, 2014, pp. 609–622.
[6] Y-H. Chen *et al.*, *Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks*, ISCA, 2016.
[7] J. Deng *et al.*, *Imagenet: A large-scale hierarchical image database*, CVPR, 2009, pp. 248–255.
[8] Z. Du *et al.*, *Shidiannao: Shifting vision processing closer to the sensor*, ISCA, 2015, pp. 92–104.
[9] R. Emery *et al.*, *Connection-centric network for spiking neural networks*, NOCS, 2009, pp. 144–152.
[10] N. Jouppi *et al.*, *In-datacenter performance analysis of a tensor processing unit*, ISCA, 2017.
[11] M. Khan *et al.*, *Spinnaker: mapping neural networks onto a massively-parallel chip multiprocessor*, IJCNN, 2008, pp. 2849–2856.
[12] G. Kim *et. al*, *A 1.22 tops and 1.52 mw/mhz augmented reality multicore processor with neural network noc for hmd applications*, SSC **50** (2015), no. 1, 113–124.
[13] T. Krishna *et al.*, *Breaking the on-chip latency barrier using smart*, HPCA, 2013, pp. 378–389.
[14] A. Krizhevsky *et al.* , *Imagenet classification with deep convolutional neural networks*, NIPS, 2012, pp. 1097–1105.
[15] A. Krizhevsky *et al.*, *Learning multiple layers of features from tiny images*, https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf, 2009.
[16] D. Liu *et al.*, *Pudiannao: A polyvalent machine learning accelerator*, ASPLOS, 2015.
[17] P. Lotfi-Kamran *et al.*, *Noc-out: Microarchitecting a scale-out processor*, ISCA, 2012.
[18] M. Martins *et al.*, *Open cell library in 15nm freepdk technology*, ISPD, 2015.
[19] S. Murali *et. al*, *Sunmap: a tool for automatic topology selection and generation for nocs*, DAC, 2004, pp. 914–919.
[20] R. Nikhil, *Bluespec system verilog: efficient, correct rtl from high level specifications*, MEMOCODE, 2004, pp. 69–70.
[21] U. Ogras *et al.*, *Application-specific network-on-chip architecture customization via long-range link insertion*, ICCAD, 2005.
[22] H. Sharma *et al.*, *From high-level deep neural models to fpgas*, MICRO, 2016.
[23] C. Szegedy *et al.*, *Going deeper with convolutions*, CVPR, 2015.
[24] T. Theocharides *et al.*, *A generic reconfigurable neural network architecture implemented as a network on chip*, SOC, 2004.
[25] D. Vainbrand *et al.*, *Network-on-chip architectures for neural networks*, NOCS, 2010, pp. 135–144.