A Latency-Optimized Reconfigurable NoC for In-Memory Acceleration of DNNs

Sumit K. Mandal, *Graduate Student Member, IEEE*, Gokul Krishnan, *Graduate Student Member, IEEE*, Chaitali Chakrabarti, *Fellow, IEEE*, Jae-Sun Seo, *Senior Member, IEEE*, Yu Cao, *Fellow, IEEE*, and Umit Y. Ogras, *Senior Member, IEEE*

***Abstract*— In-memory computing reduces latency and energy consumption of Deep Neural Networks (DNNs) by reducing the number of off-chip memory accesses. However, crossbar-based in-memory computing may significantly increase the volume of on-chip communication since the weights and activations are on-chip. State-of-the-art interconnect methodologies for in-memory computing deploy a bus-based network or mesh-based Network-on-Chip (NoC). Our experiments show that up to 90% of the total inference latency of a DNN hardware is spent on on-chip communication when the bus-based network is used. To reduce the communication latency, we propose a methodology to generate an NoC architecture along with a scheduling technique customized for different DNNs. We prove mathematically that the generated NoC architecture and corresponding schedules achieve the minimum possible communication latency for a given DNN. Furthermore, we generalize the proposed solution for edge computing and cloud computing. Experimental evaluations on a wide range of DNNs show that the proposed NoC architecture enables 20%-80% reduction in communication latency with respect to state-of-the-art interconnect solutions.**

***Index Terms*— In-memory computing, deep neural networks, neural network accelerator, network-on-chip, interconnect.**

I. Introduction

In RECENT years, deep neural networks (DNNs) have shown tremendous success in recognition and detection tasks such as image processing, health monitoring, and language processing [1], [2]. Higher accuracy in DNNs is achieved by using larger and more complex models. However, such models require a large number of weights, and consequently, traditional DNN hardware accelerators require a large number of memory accesses to fetch the weights from off-chip memory, leading to a large number of off-chip memory accesses lead to higher latency and energy consumption. On average, a single off-chip memory access consumes 1,000× the energy of a single computation [3]. Therefore, there is a strong need to minimize the latency and energy consumption due to the off-chip memory accesses in DNN accelerators.

Manuscript received May 30, 2020; revised July 21, 2020; accepted August 2, 2020. Date of publication August 11, 2020; date of current version September 21, 2020. This work was supported in part by the Center for Brain-inspired Computing (C-BRIC), one of the six centers in JUMP, the Semiconductor Research Corporation Program sponsored by the Defense Advanced Research Projects Agency (DARPA) through the NSF CAREER Award under Grant CNS-1651624 and in part by the Semiconductor Research Corporation under Grant ID 2938.001. This article was recommended by Guest Editor K.-C. Chen. *(Corresponding author: Sumit K. Mandal.)*

The authors are with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85287 USA (e-mail: skmandal@asu.edu; gkrish19@asu.edu; chaitali@asu.edu; jseo28@asu.edu; yu.cao@asu.edu; umit@asu.edu).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JETCAS.2020.3015509

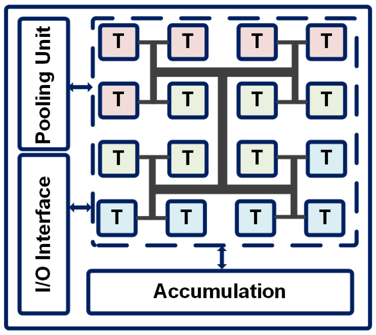


Fig. 1. Multi-tiled IMC architecture with bus-based H-Tree interconnect [11].

In-Memory Computing (IMC) techniques reduce memory access related latency and energy consumption through the integration of computation with memory accesses. A prime example is the crossbar-based IMC architecture which provides a significant throughput boost for DNN acceleration. Prior works have shown that both SRAM- and ReRAM-based crossbar architectures effectively improve performance and energy efficiency [4]–[7]. Such advantages stem from the efficiency of matrix multiplication implementation in crossbar architectures. At the same time, crossbar-based in-memory computing dramatically increases the volume of on-chip communication, when all weights and activations are stored on-chip. Emerging DNNs with higher accuracy, such as those derived through Neural Architecture Search (NAS) [8]–[10], further exacerbate the problem of on-chip communication due to larger model size and more complex connections. Therefore, designing an efficient on-chip communication architecture is crucial for the in-memory acceleration of DNNs.

State-of-the-art IMC architectures usually deploy a bus-based H-Tree interconnect [6], [12]. Figure 1 shows such a multi-tiled IMC architecture with bus-based H-Tree interconnect [11]. In this figure, the tiles in different layers shown in different colors, are connected through a bus-based H-Tree interconnect. We evaluate a range of DNNs for such an architecture using the NeuroSim [11] benchmarking tool. Figure 2 shows that up to 90% of the total inference latency of a DNN

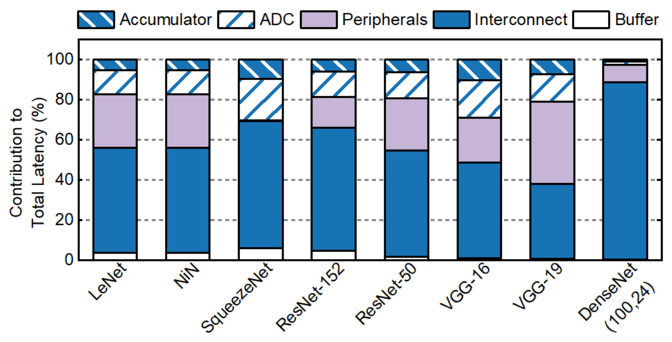


Fig. 2. Percentage contribution of different components of the IMC hardware to total latency. 40%-90% of the total latency is spent on on-chip communication when bus-based H-Tree interconnect is used.

hardware is spent on on-chip communication when the H-Tree interconnect is used.

In order to reduce on-chip communication latency, NoC-based interconnects are employed for conventional SoCs [13] and DNN accelerators [4], [14]. Eyeriss-V2 [14] proposes to use three different NoCs for weights, activations, and partial sums. Such an architectural choice allows for higher performance at the cost of both area and energy consumption. ISAAC [4] employs a concentrated-mesh (cmesh) NoC at the tile-level of the IMC accelerator. These empirical approaches demonstrate the necessity of NoC for in-memory computing of DNNs. However, a regular NoC does not guarantee minimum possible communication latency for DNN architectures. The performance of the NoC depends on both the NoC structure and the underlying workload.

In this work, we first propose an optimization technique to determine the optimal number of NoC routers required for each layer of the DNN. Next, we propose a methodology to generate a latency-optimized NoC architecture along with a scheduling technique customized for different DNNs. We prove, through induction, that the proposed NoC architecture achieves minimum possible communication latency using the minimum number of links between the routers. These two techniques together generate a custom NoC for IMC acceleration of a given DNN. We show that the proposed custom IMC architecture achieves 20%-80% improvement in overall communication latency and 5%-25% reduction in end-to-end inference latency with respect to state-of-the-art NoC based IMC architectures [4].

Constructing a custom NoC for each DNN is not a practical choice as fabricating custom hardware is expensive. The optimum DNN configuration changes due to on-line adaptation of algorithmic pruning ratio [15] and accuracy vs speed/power trade-off [16]. Consequently, communication patterns between different layers also change with the DNN configuration. Hence, the NoC needs to be configured to maintain optimality. Moreover, new DNNs are being designed at a fast rate due to the large research volume in this domain. Therefore, an exhaustive design-time exploration that considers all possible DNNs is not feasible. As a result, the NoC designed considering only the known DNNs will not be optimal for new DNNs. Hence, it must be configured at run-time to maintain the optimality. To this end, we propose a reconfigurable solution for two categories of DNNs namely, edge computing-based and cloud computing-based DNNs. We categorize the DNNs based on its application. For example, authors in [17] show the extensive usage of LeNet, SqueezeNet, VGG in various edge devices, and [18] use ResNet-152 DNN for cloud-based applications such as video analytics. However, there exist multiple factors which differentiate these DNNs, namely, number of layers, number of parameters, connection density, etc. In this article we consider LeNet, NiN, SqueezeNet, VGG-16, and VGG-19 as edge-computing based DNNs and ResNet-50, ResNet-152, and DenseNet (100,24) as cloud computing-based DNNs. We construct separate NoC architectures for these two categories offline. When a new DNN that was not known at design-time is to be executed, the NoC architecture is reconfigured to accommodate the DNN. Through leave-one-out experiments, we show that the proposed reconfigurable NoC has less than 5% performance degradation on average with respect to the customized solution. Overall, the proposed methodology generates both custom and reconfigurable NoC-based IMC hardware architectures that provide better performance than state-of-the-art IMC hardware for DNNs.

The major contributions of this work are as follows:

- A methodology to construct an NoC architecture along with a scheduling technique that provides minimum communication latency for a given DNN. We prove by induction that the proposed NoC achieves minimum communication latency.
- Reconfigurable NoC architecture for edge computing-based and cloud computing-based DNNs.
- Experimental evaluation of the proposed NoC-based IMC architecture showing up to 80% reduction in communication latency with respect to state-of-the-art interconnect solution for IMC hardware of DNNs.

II. Related Work

IMC-based hardware architectures have emerged as a promising alternative to conventional von-Neumann architectures. Prior works have proposed IMC hardware based on both SRAM and nanoscale non-volatile memory (e.g. resistive RAM or ReRAM) [4]–[6], [12], [19]. Authors in [4] proposed a ReRAM-based IMC architecture for DNN inference. The architecture utilizes a crossbar of size 128×128 to perform the Multiply-and-Accumulate (MAC) operations. They employ a parallel read-out method to accelerate the MAC computations in the analog domain. In addition, a Digital-to-Analog Converter (DAC) and an Analog-to-Digital Converter (ADC) is employed to switch between digital and analog domains. In contrast, [6] utilized spike-based computation to perform MAC operations in the time domain. Such an architecture does not need DAC and ADC units. An atomic computation-based ReRAM IMC architecture for both training and inference of DNNs is proposed in [20]. Authors in [12] proposed a systolic array-based ReRAM IMC design for DNN inference. Other works proposed in the past have explored SRAM-based IMC [5], [19].

All of the prior works focus on accelerating the computation while giving less importance to inter-layer data movement.

Crossbar-based IMC hardware designs for DNNs significantly increase the volume of on-chip communications, making the role of on-chip interconnect crucial. Different on-chip interconnect solutions have been used for IMC-based DNN accelerators in the literature. Bus-based H-Tree interconnect is proposed in [11], [21]. However, bus-based interconnect contributes up to 90% of the total inference latency, as shown in Figure 2. Hence, a bus-based interconnect does not provide a scalable solution for DNN accelerators. Shafiee *et al.* employs a concentrated mesh (cmesh)-based NoC to connect multiple tiles on-chip [4]. An IMC-based DNN accelerator for high-precision training is proposed in [22]. Ni *et al.* proposed a distributed in-memory computing architecture with a binary RRAM-based crossbar [23]. However, all these techniques assume a fixed interconnect architecture for different DNNs, i.e. these techniques do not cater to the communication needs of different DNNs.

The DNN accelerators presented in MAERI [24] and Eyeriss-v2 [14] use a flexible interconnect for DNN accelerators. In MAERI [24], a fat-tree-based programmable interconnect is used to support various sparse and non-sparse DNN dataflows. The work in [14] proposes a hierarchical mesh-NoC for DNNs with different operating modes to support different levels of data reuse in different dataflow patterns. However, these works do not consider the non-uniform weight distribution of different DNNs [25], DNN graph structure, and the computation-to-communication imbalance of the DNNs. A communication-centric IMC architecture is proposed in [26]. In this work, an optimization-based technique is incorporated to construct the schedules of a given DNN on mesh interconnect. Nonetheless, this architecture does not guarantee minimum possible communication latency for different DNNs.

In contrast to prior works, we propose an NoC architecture along with a scheduling technique that achieves the minimum possible communication latency for a given DNN. Specifically, our proposed approach takes different DNN parameters such as graph structure and weights distribution as input and constructs an NoC architecture with schedules ensuring minimum possible communication latency. This NoC is customized to a given DNN and does not inherit any of state-of-the-art topology (e.g. tree, mesh, torus, etc.). Furthermore, we propose a reconfigurable NoC architecture for two representative class of DNNs, namely, edge computing-based and cloud computing-based DNNs. The reconfigurable architecture assumes that the IMC resources (IMC tiles and associated peripheral circuits) are available on-chip. Then, based on the DNN, the NoC architecture is reconfigured to obtain the lowest possible communication latency.

III. Background and Overview

A. Background of In-Memory Computing

Conventional architectures separate the data access from memory and the computation in the computing unit. In contrast, IMC seamlessly integrates computation and memory access in a single unit such as the crossbar [4]–[7]. It has emerged as a promising method to address the memory access bottleneck. Both SRAM and NVM-based (e.g. ReRAM) IMC

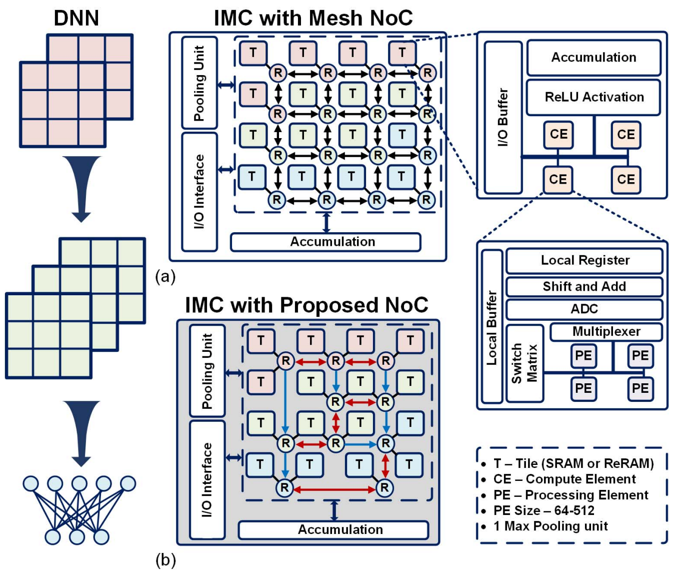


Fig. 3. IMC architecture with an arbitrary DNN mapped onto different tiles with, (a) the mesh-NoC and (b) the proposed latency-optimized NoC.

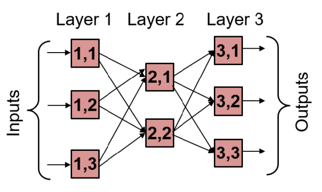
hardware architectures provide a dense and parallel structure to achieve high performance and energy efficiency. This localizes computation and data memory in a more compact design and enhances parallelism with multiple-row access, resulting in improved performance [4], [6].

The IMC architecture consists of Processing Elements (PE) or crossbar arrays built with IMC cells which hold the weights of the DNN. The size of the PE can vary from 64×64 to 512×512. Along with the computing unit, peripheral circuits such as ADC, Sample and Hold circuit (S&H) and accumulator circuits are used to obtain each DNN layer's computation result. The Word-Line (WL) is activated by the input activation which allows for the MAC operation to be performed along the Bit-Line (BL) via analog voltage/current accumulation. The analog MAC result is converted to digital values using an ADC, and subsequently accumulated using a shifter and adder circuit.

B. Overview of the Proposed IMC Architecture With Latency-Optimized NoC

Figure 3 shows a representative IMC hardware architecture for DNN inference acceleration [11]. The proposed IMC system utilizes a hierarchical architecture where each tile has computing elements (CEs) and each CE employs processing elements (PEs). We assume all the weights and activations are stored on-chip. Each tile consists of four CEs, input-output (IO) buffers, accumulator circuits, and a ReLU activation circuit. An H-Tree-based interconnect is used to connect the different hardware units in the tile. In addition, there is a global pooling and accumulator unit to perform pooling and inter-tile accumulation, respectively.

Each CE in a tile consists of four PEs that communicate through a bus-based interconnect. The PEs represent the crossbar structure (SRAM or ReRAM) which performs the computation. Each CE further consists of a read-out circuit that converts the MAC results from analog to the digital domain.



(a)

Layer 1 – Layer 2		Layer 2 – Layer 3	
Link	Cycle	Link	Cycle
$L_{1,1} - L_{2,1}$	1	$L_{2,1} - L_{3,1}$	4
$L_{1,2} - L_{2,2}$	1	$L_{2,2} - L_{3,2}$	4
$L_{1,1} - L_{2,2}$	2	$L_{2,1} - L_{3,2}$	5
$L_{1,3} - L_{2,1}$	2	$L_{2,2} - L_{3,1}$	5
$L_{1,2} - L_{2,1}$	3	$L_{2,2} - L_{3,2}$	6
$L_{1,3} - L_{2,2}$	3	$L_{2,1} - L_{3,3}$	6

(b)

Fig. 4. (a) Communication between layers in a DNN and (b) The schedules for obtaining minimum communication latency between layers. Without loss of generality, it is assumed that the computation time in the tile is 0 cycles.

The read-out circuit consists of a sample and hold circuit, flash ADCs, and a shift and add circuit. The choice of ADC stems from the precision of the partial sums required for the best accuracy and the physical footprint and performance of the ADC. A multiplexer circuit is employed to share the read-out circuit among different columns of the IMC crossbar. We multiplex 8 columns for each read-out circuit in a time-multiplexed manner to reduce both area and energy for the peripheral circuitry. Finally, the architecture does not utilize a DAC; it assumes sequential input signaling, i.e., an n-bit input signal is fed over n clock cycles in a bit-serial manner.

Figure 3(a) shows the IMC architecture with a regular mesh NoC at the tile level. The regular mesh NoC has one router-per-tile and employs the standard X-Y routing algorithm. Figure 3(b) shows the IMC architecture with the proposed latency-optimized NoC. The router has the architecture described in Section IV-E. Section IV details the methodology proposed to generate the latency-optimized NoC for IMC acceleration of DNN inference. The proposed latency-optimized NoC architecture utilizes the optimal number of routers and links to perform on-chip communication. Such an architecture results in reduced energy and latency.

IV. Methodology

A. Determining Minimum Communication Latency

We aim to construct an NoC architecture, customized for different DNNs, which achieves minimum possible communication latency. To this end, we first show how the minimum possible communication latency between two consecutive layers of a given DNN for one round of communication is achieved. In one round of communication, each source router of a layer sends one packet to each destination router in the next layer. Since each router sends/receives maximum one packet per cycle, the minimum possible communication latency to finish all the transactions is $\max(N_k,\ N_{k+1})$, where N_k is the number of routers in k^{th} layer.

Figure 4(a) represents an IMC hardware of a neural network with one hidden layer, where each square represents a tile. There are three tiles in the input layer (Layer 1), two tiles in the hidden layer (Layer 2), and three tiles in the output layer (Layer 3). We assume that there is an NoC router associated with each tile, and all routers have one input and one output port. We also assume layer-by-layer operation for the DNN, i.e. after all packets reach layer 2 from layer 1, then the communication between layer 2 and layer 3 will start. If each router of a layer is connected to every other router of the next layer (as shown in Figure 4(a)), then the minimum possible communication latency is achieved. We utilize below key insights to construct the schedules for obtaining minimum communication latency.

TABLE I
Summary of the Notations Used in This Article

A_k	Number of output activations in the k^{th} layer
N_k	Number of routers in the k^{th} layer
$R_{k,n}$	n^{th} router in the k^{th} layer
$R_{k_1,n_1} - R_{k_2,n_2}$	The packet between R_{k_1,n_1} and R_{k_2,n_2}
$L_{k_1,n_1} - L_{k_2,n_2}$	The link between R_{k_1,n_1} and R_{k_2,n_2}

Key insight 1: A single router can not send/receive more than one packet in a cycle. However, multiple routers can send/receive packets in parallel.
Key insight 2: Since a router can send/receive only one packet in a cycle, the congestion is minimum i.e., no more than a transaction is scheduled through a particular link.

Figure 4(b) shows the schedules for each round of communication between two layers to achieve the minimum possible communication latency. With these schedules, there is no congestion in the NoC, since no link is scheduled to carry more than one packet in a particular cycle. With this NoC topology, the number of links required (to achieve minimum possible communication latency) between the k^{th} layer and $(k+1)^{\text{th}}$ layer is $N_k \times N_{k+1}$. Thus the number of links is $O(N^2)$, where N is the number of routers in a layer. The total number of links can be very large for DNNs with a large number of routers per layer. For example, with this NoC topology, more than 1.5×10^4 links are required for DenseNet (100,24). Since the number of links increases the NoC area, this NoC topology is impractical to implement. To overcome this challenge, we first propose a technique to determine the optimal number of routers required for each layer of a DNN. In addition, we propose an NoC architecture that achieves the minimum possible communication latency between two consecutive layers of DNNs with minimum number of links.

B. Determining the Optimal Number of Routers

The optimal number of routers in each layer of a DNN is a function of the number of packets that are sent from one layer to the next. Since a higher number of packets between two source-destination router pairs increase communication latency (due to higher congestion), determining the optimal number of routers for each DNN layer is crucial. We perform an analysis which shows that the inefficient distribution of routers can cause higher communication latency. In Figure 5(a), we show the number of packets between two routers for each layer of different DNNs when one router is allocated per tile. The number of packets is normalized with respect to the highest number of packets (4×10^5), which occurs between two routers in the 4th and 5th layers of SqueezeNet. High number of packets between two routers increases congestion in the NoC, resulting in high communication latency.

Therefore, we propose a technique to determine the optimal number of routers required for each layer of the DNN.

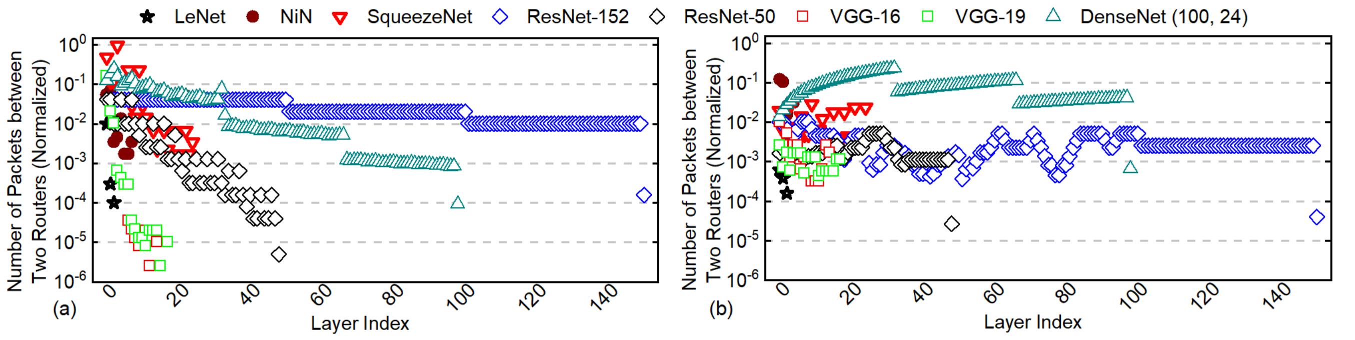


Fig. 5. The number of packets between two routers with (a) one router per tile, and (b) the proposed technique. All the numbers are normalized with a factor of 4×10^5 (the highest number of packets per router between two layers with one router per tile, which occurs between 4th and 5th layer of SqueezeNet). The number of packets between routers decreases significantly with the proposed approach for all DNNs except DenseNet (100,24) and ResNet-50. However, the number of routers used for DenseNet with our proposed technique (300) is less than the number of routers required (1088) if one router is allocated per tile. Our technique achieves around 72% reduction in the NoC area. Similar improvement is seen for ResNet-50 as well.

The objective function is shown in Equation (1). It is a function of the number of routers in each layer which is denoted by $\bar{\mathbf{N}}$, where $\bar{\mathbf{N}} = \{N_1, N_2, \ldots, N_K\}$. The number of activations between the k^{th} and the $(k+1)^{\text{th}}$ layer is denoted by A_k. Therefore, if we divide A_k by the product of N_k and N_{k+1}, we obtain the number of activations between a pair of routers.

$$L(\bar{\mathbf{N}}) = \sum_{k=1}^{K-1} \left(\left\lceil \frac{A_k}{N_k N_{k+1}} \frac{Q}{W} \right\rceil \right) \max(N_k, N_{k+1}) \qquad (1)$$

$$\underset{\bar{\mathbf{N}}}{\text{minimize}} \; L(\bar{\mathbf{N}})$$

$$\text{subject to } N_k > 0; \quad k = 1, \ldots, K,$$

$$\sum_{k=1}^{K} N_k < N. \qquad (2)$$

Furthermore, after multiplying the expression by the bit precision (Q) and dividing it by the NoC-bus width (W), we obtain the number of packets between a pair of routers ($\frac{A_k}{N_k N_{k+1}} \frac{Q}{W}$). To convert the value to integer we take the ceiling of the expression. The minimum possible latency to finish each round of communication is $\max(N_k,\ N_{k+1})$ cycles as shown in Section IV-A, and the number of rounds equals the number of packets between a pair of routers. Therefore, multiplying these two terms we obtain the total communication latency of the DNN with the proposed NoC architecture (Equation (1)). At this point, the number of routers for each layer is unknown. Therefore, we minimize the objective function by setting all elements of $\bar{\mathbf{N}}$ positive and a user-defined upper bound on the total number of routers (N) for the DNN, as shown in Equation (2). We solve the optimization problem using the gradient-based interior point algorithm. A sub-gradient based methodology is incorporated in the region where the function is not differentiable. In this work, we assume that packet transmissions happen only in two consecutive layers. Therefore ensuring local minimum (minimum number of links between two consecutive layers) is sufficient to ensure global minimum latency with a minimum number of links.

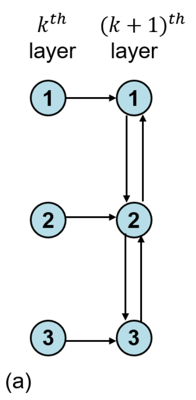
Next, we show how the minimum communication latency for the configuration shown in Figure 4(a) is achieved. For each round of communication, one packet is communicated between a pair of routers in two consecutive layers. Therefore, $\left\lceil \frac{A_k}{N_k N_{k+1}} \frac{Q}{W} \right\rceil = 1$ in Equation 1. For the configuration shown in Figure 4(a), $N_1 = 3, N_2 = 2, N_3 = 3$. Putting this in Equation 1 we obtain, $L(\bar{\mathbf{N}}) = \max(3, 2) + \max(2, 3) = 3 + 3 = 6$. Therefore, minimum communication latency for this configuration is 6 cycles, which supports the schedules shown in Figure 4(b).

Figure 5(b) shows the number of normalized packets between two routers for different DNNs with our proposed tile-to-router mapping methodology. We observe that with the proposed mapping methodology, the number of packets between two routers for different layers of different DNNs is always less than 0.25. We observe an increase in the number of packets between two routers in the case of DenseNet and ResNet-50 due to a decrease in the number of routers with our proposed methodology. The number of routers used for DenseNet with our proposed technique (300) is less than the number of routers required (1088) if one router is allocated per tile. This provides 72% reduction in the NoC area. Similar improvement is seen for ResNet-50 as well.

C. Constructing the Custom NoC

In this sub-section, we construct our proposed latency-optimized NoC architecture. The optimal number of routers required for each layer of a given DNN is computed using the methodology described in Section IV-B. We analyze a given DNN layer-by-layer to obtain the latency-optimized NoC architecture and the corresponding schedules. We show, by induction, that the proposed NoC architecture along with the schedules achieves the minimum communication latency using the minimum number of links for one round of communication. Without loss of generality, let us assume N_k and N_{k+1} are the number of routers in two consecutive layers. We consider three cases: 1) $N_k = N_{k+1}$, 2) $N_k < N_{k+1}$ and 3) $N_k > N_{k+1}$. The minimum possible latency for one round of communication between the two layers is $\max(N_k,\ N_{k+1})$. For each of these cases, we develop the latency-optimized NoC architecture and the corresponding schedules. We also prove that the constructed NoC architecture achieves the minimum possible latency for all cases.

Case 1 ($\mathbf{N_k = N_{k+1}}$): We first consider a case where two consecutive layers of a DNN have three routers each, i.e.



(b)

Link	Cycle	Transaction
$L_{k,1} - L_{(k+1),1}$	1	$R_{k,1} - R_{(k+1),1}$
$L_{k,2} - L_{(k+1),2}$	1	$R_{k,2} - R_{(k+1),2}$
$L_{k,3} - L_{(k+1),3}$	1	$R_{k,3} - R_{(k+1),3}$
$L_{(k+1),1} - L_{(k+1),2}$	2	$R_{k,1} - R_{(k+1),2}$
$L_{(k+1),2} - L_{(k+1),3}$	2	$R_{k,2} - R_{(k+1),3}$
$L_{(k+1),3} - L_{(k+1),2}$	2	$R_{k,3} - R_{(k+1),2}$
$L_{(k+1),2} - L_{(k+1),1}$	2	$R_{k,2} - R_{(k+1),1}$
$L_{(k+1),2} - L_{(k+1),3}$	3	$R_{k,1} - R_{(k+1),3}$
$L_{(k+1),2} - L_{(k+1),1}$	3	$R_{k,3} - R_{(k+1),1}$

Fig. 6. (a) NoC architecture which achieves the minimum possible latency when two consecutive layers each have three routers, (b) Schedule to achieve the minimum possible latency.

$N_k = N_{k+1} = 3$. Since each router of $(k+1)^{\text{th}}$ layer can receive at most one packet per cycle, the minimum number of cycles required to receive packets from all routers in k^{th} layer is $N_k = \max(N_k, N_{k+1})$. Therefore, the minimum possible latency for one round of communication between each of the three source routers to each of the three destination routers is three cycles. Figure 6(a) shows our proposed NoC architecture and Figure 6(b) shows the corresponding schedule to finish all transactions in three cycles. We assume that the communication starts at cycle-1. In one round, each router in k^{th} layer sends the same packet (output activation) to all routers in $(k+1)^{\text{th}}$ layer. We also assume that when a packet reaches a router, the associated tile computes on the packets, and the transaction is considered to be completed. In the next cycle, the router can send the received packet to other routers if necessary. We denote the packet to be transmitted from i^{th} router of k^{th} layer to j^{th} router of $(k+1)^{\text{th}}$ layer as $R_{k,i} - R_{k+1,j}$ as shown in Table I. At cycle-1, all routers in k^{th} layer send the packet to a router in $(k+1)^{\text{th}}$ layer through the horizontal links as shown in Figure 6(a). First three rows in Figure 6(b) show the transaction in cycle-1. In the next cycle, each router in $(k+1)^{\text{th}}$ layer transmits the packet received in cycle-1 both through upward and downward vertical link if the links exist. In the subsequent cycles, each router in $(k+1)^{\text{th}}$ layer sends the packet received from north to south, and the packet received from south to north through the downward and upward vertical link, respectively. All transactions are finished in three cycles. Since no link is scheduled to transmit more than one packet at a particular cycle, there is no contention in the NoC. We note that if any of the links shown in Figure 6(a) is removed, then some of the transactions will not be possible. Therefore, the NoC architecture in Figure 6(a) achieves the minimum possible latency using the minimum number of links.

Next, we prove (by induction) that, if the NoC architecture with $N_k = N_{k+1} = N-1$ achieves minimum latency, then the architecture with $N_k = N_{k+1} = N$ also achieves minimum latency. Figure 7 shows the architecture with $N_k = N_{k+1} = N$. The dotted box shows the architecture which is assumed to achieve minimum possible latency, i.e. all transactions will

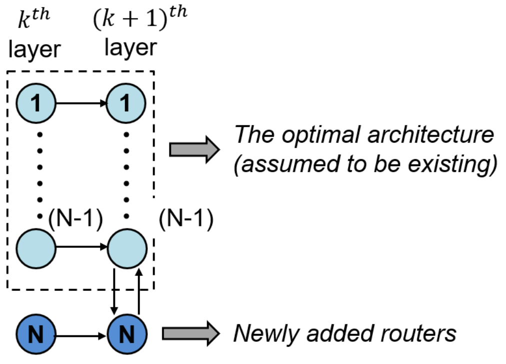


Fig. 7. NoC architecture which achieves the minimum possible latency when two consecutive layers consist of N routers each.

TABLE II
SCHEDULES FOR $N_k = N_{k+1}$

Link Type	Cycle	Link	Transaction	Range
Horizontal	1	$L_{k,n}-$ $L_{(k+1),n}$	$R_{k,n}-$ $R_{(k+1),n}$	$n = 1 \dots N$
Upward Vertical	m	$L_{(k+1),n}-$ $L_{(k_1),(n-1)}$	$R_{k,(n+m-2)}-$ $R_{(k+1),(n-1)}$	$n = 2 \dots N$
Downward Vertical	m	$L_{(k+1),n}-$ $L_{(k+1),(n+1)}$	$R_{k,(n-m+2)}-$ $R_{(k+1),(n+1)}$	$n = 1 \dots (N-1)$

finish at $(N-1)$ cycles. The dark blue circles indicate the newly added routers (N^{th}) in each layer. By adding the new routers and corresponding links, new transactions are introduced. Our goal is to schedule the new transactions in a way that there is no contention with any transaction scheduled with the architecture having $(N-1)$ routers in each layer. This can be achieved by scheduling the new transaction in the links in the manner shown below.

- Horizontal Link: $(L_{k,N} - L_{(k+1),N})$ carries the packet $R_{k,N} - R_{(k+1),N}$ in cycle-1.
- Upward vertical link: New transactions occur for the packet sent by the router N of the $(k+1)^{\text{th}}$ layer. The link $(L_{(k+1),N} - L_{(k+1),(N-1)})$ carries this packet at cycle-2. All other upward vertical links carry this packet after the last transaction through the links with the architecture consisting of $N-1$ routers in each layer. The packet reaches the 1st router of the $(k+1)^{\text{th}}$ layer at cycle-N.
- Downward vertical link: New transactions occur only through the link $(L_{(k+1),(N-1)} - L_{(k+1),N})$. Since this is a newly added link, the transaction does not contend with any other link. Through this link, the transaction which occurs last is $R_{k,1} - R_{(k+1),N}$ at cycle-N.

Therefore, all transactions for the architecture with N routers finish at cycle-N, which is the minimum possible latency with N routers in each layer. Table II shows the schedules for this case.

Case 2 ($\mathbf{N_k < N_{k+1}, N_k = N}$): Next, we consider the case where $N_k < N_{k+1}$. Without loss of generality, we assume that $N_k = N$. The latency-optimized NoC architecture for this case is shown in Figure 8(a). The proof that the architecture achieves minimum communication latency is shown in Appendix A. Table III shows the schedules for this case. The transactions in the horizontal link and upward vertical

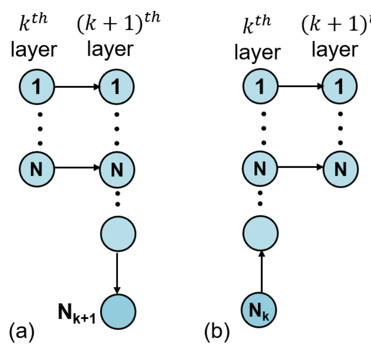


Fig. 8. NoC architecture which achieves minimum possible latency when (a) $N_k < N_{k+1}$ and (b) $N_k > N_{k+1}$.

TABLE III
SCHEDULES FOR $N_k < N_{k+1}$

Link Type	Cycle	Link	Transaction	Range
Horizontal	1	$L_{k,n}-$ $L_{(k+1),n}$	$R_{k,n}-$ $R_{(k+1),n}$	$n = 1 \dots N_k$
Upward Vertical	m	$L_{(k+1),n}-$ $L_{(k+1),(n-1)}$	$R_{k,(n+m-2)}-$ $R_{(k+1),(n-1)}$	$n = 2 \dots N_k$
Downward Vertical	m	$L_{(k+1),n}-$ $L_{(k+1),(n+1)}$	$R_{k,(n-m+2)}-$ $R_{(k+1),(n+1)}$	$n = 1 \dots N_{k+1} - 1$

TABLE IV
SCHEDULES FOR $N_k > N_{k+1}$

Link Type	Cycle	Link	Transaction	Range
Horizontal	1	$L_{k,n}-$ $L_{(k+1),n}$	$R_{k,n}-$ $R_{(k+1),n}$	$n = 1 \dots N_{k+1}$
Horizontal	m	$L_{k,N_{k+1}}-$ $L_{(k+1),N_{k+1}}$	$R_{k,(N_{k+1}+m-1)}-$ $R_{(k+1),N_{k+1}}$	-
Upward Vertical	m	$L_{k,N_{k+1}+n}-$ $L_{k,N_{k+1}+n-1}$	$R_{k,(N_{k+1}+n+m-1)}-$ $R_{(k+1),N_{k+1}}$	$n = 1 \dots$ $(N_k - N_{k+1})$
Upward Vertical	m	$L_{(k+1),n}-$ $L_{(k_1),(n-1)}$	$R_{k,(n+m-2)}-$ $R_{(k+1),(n-1)}$	$n = 2 \dots N_{k+1}$
Downward Vertical	m	$L_{(k+1),n}-$ $L_{(k+1),(n+1)}$	$R_{k,(n-m+2)}-$ $R_{(k+1),(n+1)}$	$n = 1 \dots$ $(N_{k+1} - 1)$

link are same as the Case 1, since there is no change in the configuration of these links. The downward vertical links in the $(k+1)^{\text{th}}$ layer carries a packet in each cycle till all the transactions are finished.

Case 3 ($\mathbf{N_k > N_{k+1}, N_{k+1} = N}$): Next, we consider the case where $N_k < N_{k+1}$. Without loss of generality, we assume that $N_{k+1} = N$. The latency-optimized NoC architecture for this case is shown in Figure 8(b). The proof that the architecture achieves minimum communication latency is shown in Appendix B. Table IV shows the schedules for this case. The transactions in the horizontal links $L_{k,n} - L_{(k+1),n}$ to $L_{k,(N-1)} - L_{(k+1),(N-1)}$ happens in cycle-1. Apart from carrying a packet in cycle-1, the link $L_{k,N} - L_{(k+1),N}$ also carries packets from routers $R_{k,(N+j)}$ to $R_{(k+1),N}$ in subsequent cycles, where $j = 1 \dots (N_k - N)$.

In Appendix C, we show the operation of the proposed NoC architecture for densely connected DNNs such as DenseNet [27].

D. Constructing a Reconfigurable NoC

So far, we have discussed our proposed methodology to construct a latency-optimized NoC customized for a given DNN. However, an NoC architecture customized for a specific DNN is not practical due to the lack of reconfigurability. Since DNNs are ever-evolving, we can never guarantee that the set of DNNs considered at design time is exhaustive. Therefore, at run-time the NoC might need to execute a DNN which was not considered at design time. To overcome this challenge, we propose a technique to construct two reconfigurable NoCs for two categories of DNNs, namely, edge-based and cloud-based DNNs. There are two steps involved in constructing the reconfigurable architecture. *First,* we set the number of layers to be supported by the NoC architecture. *Second,* we set the number of routers per layer for the NoC architecture.

Algorithm 1 Proposed Algorithm to Reconfigure the NoC at Runtime

```
1  Input: Number of layers of the DNN ($K$), number of input activations for each layer ($A_k$), precision bit ($Q$), NoC bus width ($W$), Number of routers available on-chip for each layer ($N_k^{max}$)
2  Output: Number of routers required for each layer ($N_k$) and the optimal schedule ($sched_{out}$)
3  Initialization: $sched_{out} \leftarrow [\ ]$
4  Obtain $N_k$ following Equation 1 and Equation 2
5  for $k$ = 1: $K$ do
6      $N_k = \min(N_k, N_k^{max})$
7  end
8  for $k$ = 1: K-1 do
       /* Number of packets */
9
10     $P_k = \left\lceil \frac{A_k}{N_k N_{k+1}} \frac{Q}{W} \right\rceil$
       /* Constructing the schedules */
11
12     for $p = 1 : P_k$ do
13         if $N_k == N_{k+1}$ then
14             $sched_p$ = schedule constructed by following Table II
15             $sched_{out} \leftarrow [sched_{out}; sched_p]$
16         end
17         if $N_k < N_{k+1}$ then
18             $sched_p$ = schedule constructed by following Table III
19             $sched_{out} \leftarrow [sched_{out}; sched_p]$
20         end
21         if $N_k > N_{k+1}$ then
22             $sched_p$ = schedule constructed by following Table IV
23             $sched_{out} \leftarrow [sched_{out}; sched_p]$
24         end
25     end
26 end
```

1) Setting Number of Layers: For each category, we set the number of layers to be the maximum number of layers among all DNNs available at design time in that particular category. Specifically, if $D^{(i)}$ is the number of layers of a DNN i and the number of DNNs considered in that category is I, then the number of layers the NoC architecture can accommodate is $D = \max(D^{(1)}, D^{(2)}, \ldots, D^{(I)})$. For the DNNs we consider in the edge computing category, SqueezeNet has the maximum number of layers ($D = 26$). Similarly, DNNs we consider in the cloud computing category, ResNet-152 has the maximum number of layers ($D = 152$).

2) Setting Number of Routers per Layer: Next, we compute the number of routers to be allocated for each layer. To this end, for each DNN of that category, we evaluate the optimal number of routers required for each layer following the methodology described in Section IV-B. Then, for that particular layer, we allocate the maximum number of routers obtained across all DNNs of the category. If $N_k^{(i)}$ is the number of routers required for the k^{th} layer of a DNN i, then the number of routers allocated in the NoC architecture for k^{th} layer is $N_k^{max} = \max(N_k^{(1)}, N_k^{(2)}, \ldots N_k^{(I)})$, where $k = 1, 2, \ldots D$ and $N_k^{(i)} = 0$ if $D^{(i)} < k$.

At runtime, we reconfigure the NoC to determine how many routers need to be used and determine the schedules of the packets between each pair of layers of the new DNN. Algorithm 1 shows the proposed algorithm which is executed on-chip to reconfigure the proposed NoC architecture. The algorithm takes different DNN parameters (number of layers, number of input activations, precision bit), NoC bus width, and number of routers available on-chip for each layer as input. The number of routers required for each layer and the optimal schedules are obtained as outputs.

The number of routers required for each layer (N_k) of DNN is determined by following the procedure described in Section IV-B (shown in line 4 of Algorithm 1). If the required number of routers (N_k) are not available on-chip, then the maximum available routers (N_k^{max}) are utilized for that particular layer of the DNN (shown in line 5–7 of Algorithm 1). After that, we compute the schedules between two consecutive layers of the DNNs. First, we compute the number of packets between each source to each destination (line 9) and construct the schedules for the packets. In order to construct the schedules, we consider the three cases described in Section IV. Depending on the case, the schedules are constructed following Table II or Table III or Table IV. Second, the constructed schedule is then appended to the list of optimal layer-wise schedules ($sched_{opt}$) (shown in line 10–20 of Algorithm 1). The same procedure is repeated for all layers to obtain the complete schedule of the reconfigurable NoC.

E. Router Architecture of the Proposed NoC

Figure 9 shows the router architecture of the proposed NoC. The router has three input ports: one input port (I_P) connects with a router in the previous layer and the other two input ports are connected with routers of the same layer. We assume that all the packets to be sent in the next layer are stored in a buffer inside the compute elements in the previous layer. The input port I_N gets the input from the router situated to the north and the input port I_S gets the input from the router situated to the south. The router has two output ports: O_N sends the output to the router situated to the north and O_S sends the output to the router situated to the south. Inside the router, there are two multiplexers: M_A and M_B. M_A selects between the inputs coming from I_P and I_S and sends it to O_N. M_B

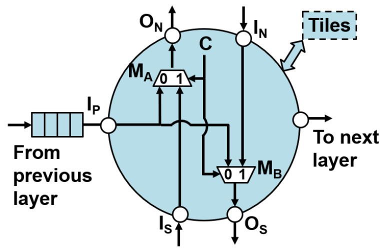


Fig. 9. Router architecture of the proposed NoC.

selects between the inputs coming from I_P and I_N and sends it to O_S. As discussed in Section IV, at cycle-2 of each round of the communication, the input from the previous layer is sent to both the routers situated to the north and south. In all other cycles, the input from the south is sent to the north and vice-versa. Therefore, M_A and M_B are controlled by the cycle-index (C) in each round of communication as shown in Figure 9. The router is interfaced with the tiles in the current layer. Upon receiving a packet, the router sends it to the corresponding computing tile. The tile computes on the packet and sends it back to the router, which then forwards it to the next router.

V. Experimental Evaluation

A. Experimental Setup

We evaluate our proposed latency-optimized NoC for IMC architecture for a wide range of DNNs. We consider LeNet [28] on MNIST dataset, NiN [29], ResNet-152 [30], VGG-16, VGG-19 [31], and DenseNet(100,24) [27] on CIFAR-100 dataset [32], and SqueezeNet [33] and ResNet-50 [30] on ImageNet dataset [34]. The DNNs we consider have parameters that range from 0.28M for LeNet to 45M for VGG-19 and the number of layers ranges from 5 for LeNet to 152 for ResNet-152. Moreover, the DNNs we chose have different connection patterns; linear (LeNet, NiN, SqueezeNet, VGG), residual (ResNet) and dense (DenseNet). These DNNs are a combination of fully connected (FC) and convolutional (Conv) layers. Therefore, our proposed methodology is applicable to fully connected layers as well as convolutional layers of a DNN.

1) Benchmarking Simulator: We developed an in-house simulator to evaluate the IMC architecture with the proposed latency-optimized NoC for different DNNs. The circuit part and interconnect part of the simulator are calibrated with NeuroSim [11] and BookSim [35], respectively. The inputs of the simulator include the DNN structure, technology node, NoC bus-width, type of IMC technology (ReRAM, SRAM, etc.), the number of bits per IMC cell, and frequency of operation. The circuit simulator performs the mapping of the entire DNN to a multi-tiled IMC architecture [4] and reports performance metrics, such as area, energy, and latency of the computing logic. The interconnect performance is evaluated using the interconnect simulator. The circuit simulator provides the number of tiles per layer, activations, and number of layers as output. These are used to construct the latency-optimized NoC, which are then fed to the interconnect simulator to compute the area, energy, and latency for the interconnect.

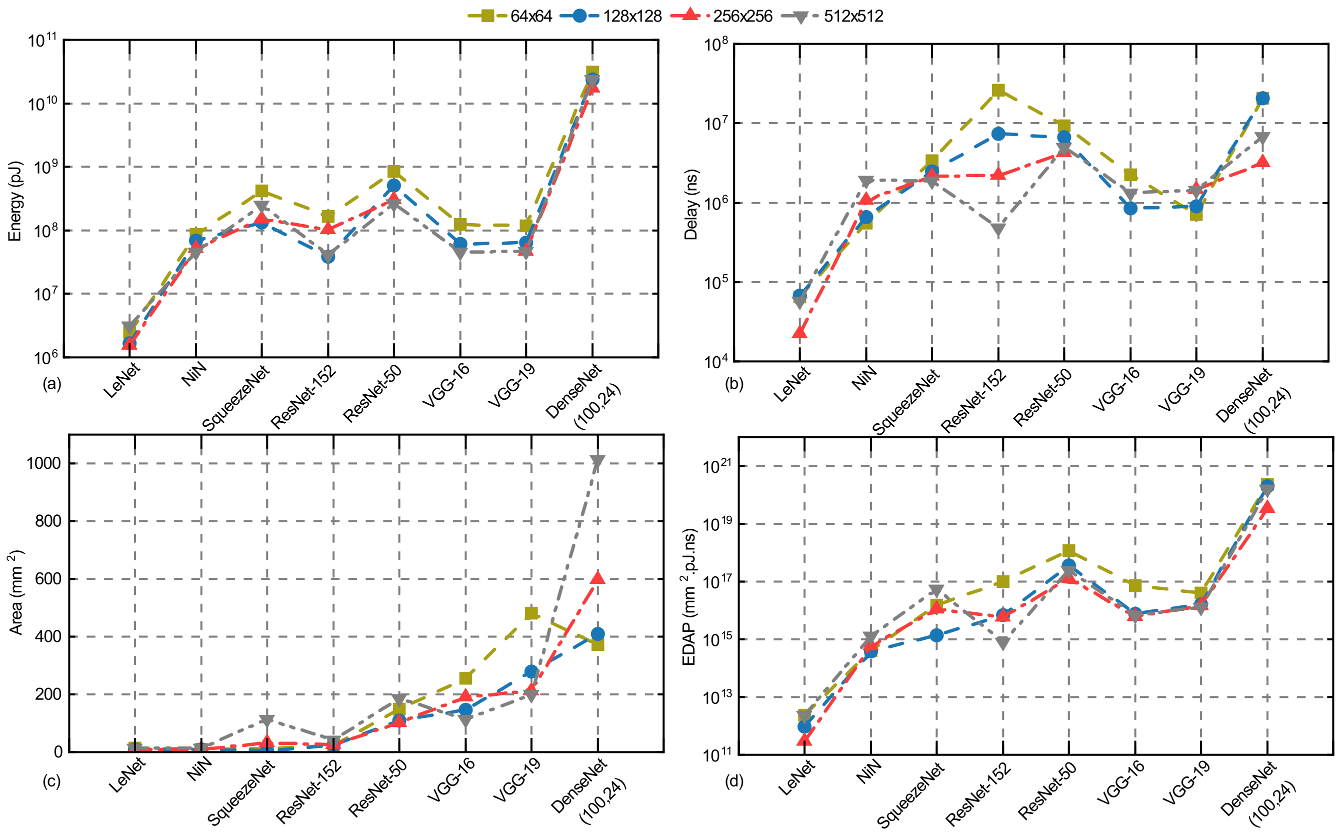


Fig. 10. (a) Energy, (b) Delay, (c) Area, and (d) EDAP of the baseline SRAM-based IMC architecture for different DNNs with different crossbar size. We observe that a crossbar size of 128 × 128 or 256 × 256 performs better (with mesh-NoC) than other crossbar sizes.

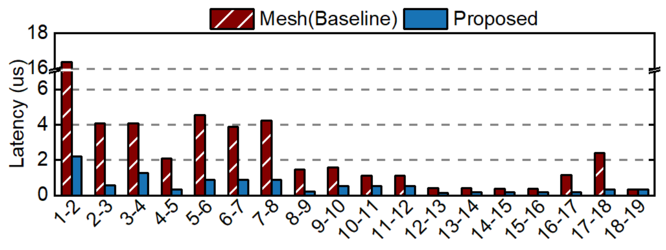


Fig. 11. Improvement in communication latency for each layer of VGG-19.

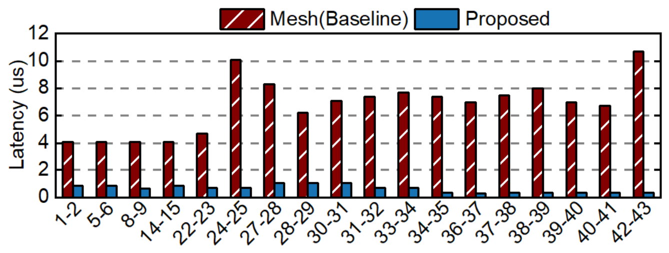


Fig. 12. Improvement in communication latency for each layer of ResNet-50.

The overall performance of the architecture is calculated by combining the circuit-level and interconnect-level performance.

2) Baseline Architecture: We utilize a crossbar-based multi-tiled IMC architecture to evaluate our proposed approach. Analog MAC computation is performed along the bitline, the analog voltage/current is digitized with a 4-bit flash ADC, a sample and hold circuit, and a shift and add circuit (read-out circuit) at the column periphery. 8 columns are multiplexed together to one read-out circuit to reduce chip area. Sequential input signalling is employed to do away with the DAC. Each tile consists of 4 compute elements (CEs) and each CE consists of 4 processing elements (PEs) or crossbar arrays [11]. We consider 32nm technology node [4], 1GHz frequency of operation, and a parallel read-out for the crossbar [6]. A mesh-based NoC with bus width of 32 bits and one router-per-tile is considered for the interconnect for the baseline architecture.

B. Optimal Size of Crossbar Array

In this section, we show the area, energy, and latency comparison of different DNNs with the baseline IMC architecture having different sized crossbars. The performance of IMC architecture for different DNNs vary with number of layers, connection density (number of connections per neuron) and number of parameters of the DNN. We consider SRAM-based bitcell/array design [11]. Figure 10 shows the comparison of energy, delay, area, and energy-delay-area product (EDAP) of crossbar size varying from 64 × 64 to 512 × 512. In each of the subfigures, the DNNs are presented in increasing order of area (LeNet has the lowest area and is at the left end while DenseNet(100,24) has the highest area and is at the right end). Figure 10(a) shows the energy consumption for different DNNs with different crossbar sizes. We observe that a crossbar size of 256×256 has the lowest energy consumption for 5 out of the 8 DNNs we evaluated. The IMC architecture

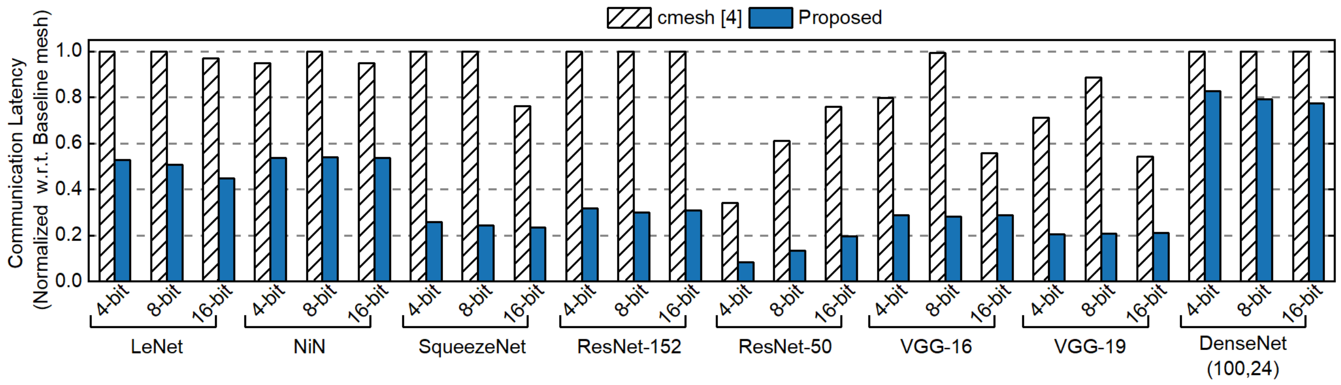


Fig. 13. Improvement in communication latency for different DNNs (with crossbar size of 256 × 256) and weights and activation precision with respect to mesh-NoC for cmesh [4] and the proposed approach.

with a crossbar size of 64 × 64 has poor performance in terms of energy consumption, inference latency, and area as shown in Figure 10(a), 10(b), and 10(c) respectively. Figure 10(d) shows the energy-delay-area product (EDAP) of different DNNs with different crossbar sizes. We observe that the architecture with a crossbar size of 128 × 128 and 256 × 256 has better performance in terms of EDAP for almost all DNNs. We observe a similar trend for IMC architecture with ReRAM-based crossbar arrays. Since larger crossbar size results in the chip being more compact, moving forward, we choose a crossbar size of 256 × 256 for all experiments. We also show some representative results with a crossbar size of 128 × 128.

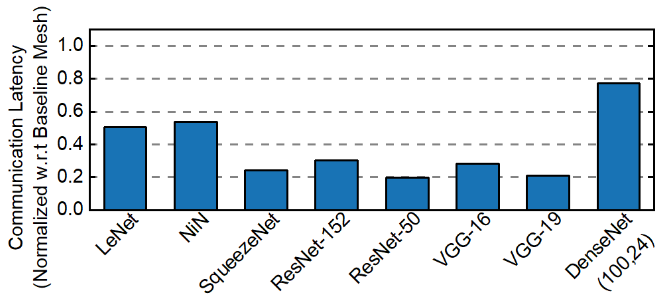


Fig. 14. Improvement in communication latency with proposed NoC with respect to mesh for crossbar size of 128 × 128.

C. Layer-Wise Comparison of Communication Latency

In this section, we show the improvement in communication latency with the proposed latency-optimized NoC (with an IMC architecture of 8-bit precision) for different layers of VGG-19 and ResNet-50. Figure 11 compares the communication latency of VGG-19 between the IMC architectures with the proposed NoC and the baseline mesh-NoC. We observe that the improvement in communication latency for the first 4 layers of VGG-19 is significant. Improvement in communication latency is highest between the first two layers of VGG-19, which is 86%.

We also observe significant improvement in communication latency for different layers of ResNet-50. Figure 12 shows the improvement for a few representative layers of ResNet-50 (we limit it for better visibility). The maximum improvement is seen between layer 42 and layer 43, which is 96%. The improvement in communication latency for each layer contributes to the improvement in total communication latency as shown in the next section. Such high improvement stems from the proposed latency-optimized NoC and the efficient distribution of routers among layers as discussed in Section IV-B.

D. Overall Improvement in Communication Performance

Next, we evaluate the proposed latency-optimized NoC-based IMC architecture for different DNNs. We compare the total communication latency of the proposed NoC with the cmesh interconnect proposed in [4]. Figure 13 shows the comparison of communication latency for three different data precisions (weights and activations): 4-bit, 8-bit, and 16-bit. In this case, we consider an IMC architecture with a crossbar size of 256 × 256. The communication latency values are normalized with respect to the communication latency values obtained with the baseline NoC. We observe that the communication latency for the cmesh interconnect is similar to that of the baseline NoC for LeNet, NiN, DenseNet (100,24), and ResNet-152. The cmesh-based IMC architecture performs significantly better for VGG-16, VGG-19, and ResNet-50. Specifically, for ResNet-50, on average, the cmesh interconnect achieves 57% improvement in communication latency with respect to the baseline mesh-NoC.

Our proposed latency-optimized NoC reduces the communication latency significantly both with respect to baseline NoC and cmesh-based NoC as shown in Figure 13. With respect to cmesh-based NoC, there is a 20%-80% improvement for different DNNs with different bit precisions. Highest improvement with respect to the baseline NoC is observed for ResNet-50 with a 4-bit data precision. On average, the proposed latency optimized NoC improves the communication latency by 62% with respect to mesh-NoC, and 57% with respect to cmesh interconnect. Since our proposed NoC architecture achieves minimum latency, there is a significant improvement in communication latency with respect to state-of-the-art works.

In Figure 14, we show the improvement in communication latency with the proposed NoC with respect to mesh-NoC for an IMC architecture with a crossbar size of 128 × 128 and

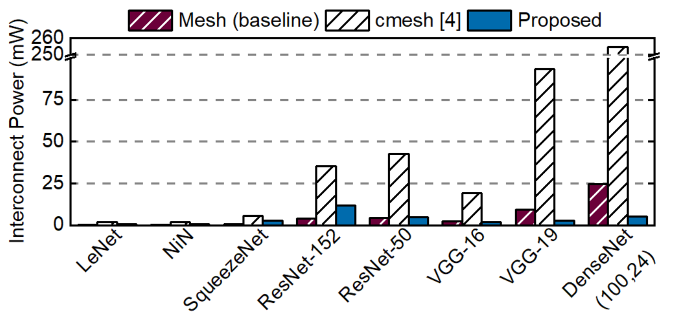


Fig. 15. Comparison of interconnect power consumption with different techniques.

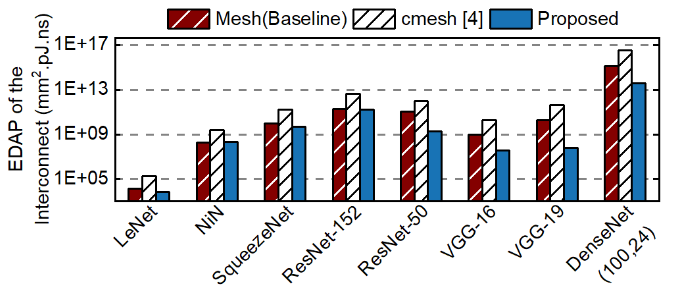


Fig. 16. Interconnect EDAP comparison for different DNNs.

8-bit precision. We observe that the improvement in communication latency follows a similar trend as the crossbar size of 256 × 256. Therefore, the improvement due to the proposed latency-optimized NoC is independent of crossbar size.

Figure 15 presents the comparison of interconnect power consumption for an IMC architecture with 8-bit precision for both weights and activations. Our proposed latency-optimized NoC achieves up to 4.6× improvement in interconnect power consumption with respect to baseline mesh-NoC. The power consumption with the proposed interconnect for ResNet-152 is higher than the baseline mesh-NoC due to the use of higher number of routers. However, this results in 3.32× improvement in interconnect latency with the proposed interconnect. We achieve 2.26×–47× improvement in power consumption as compared to the cmesh interconnect. The improvement is highest for DenseNet and least for SqueezeNet.

To further understand the efficacy of the proposed NoC, we compare the energy-delay-area product (EDAP) with respect to the baseline mesh-NoC and cmesh interconnect for different DNNs. Figure 16 shows the comparison for an IMC architecture with 8-bit precision for both weights and activations. Our proposed latency-optimized NoC achieves up to 328× improvement in the EDAP of the interconnect with respect to baseline mesh-NoC. We achieve EDAP improvement for the interconnect in the range of 12×–6600× as compared to the cmesh interconnect. The improvement is highest for VGG-19 and least for NiN. Since cmesh interconnect uses additional number of routers and links to reduce latency, it results in higher area and energy. Therefore, the performance of cmesh interconnect is worse than mesh-NoC in terms of EDAP. The proposed latency-optimized NoC provides a large improvement in communication latency which results in reduced energy with reduced or comparable area. Therefore, our proposed latency-optimized NoC architecture performs significantly better in terms of area, energy and latency than both cmesh interconnect and the baseline NoC.

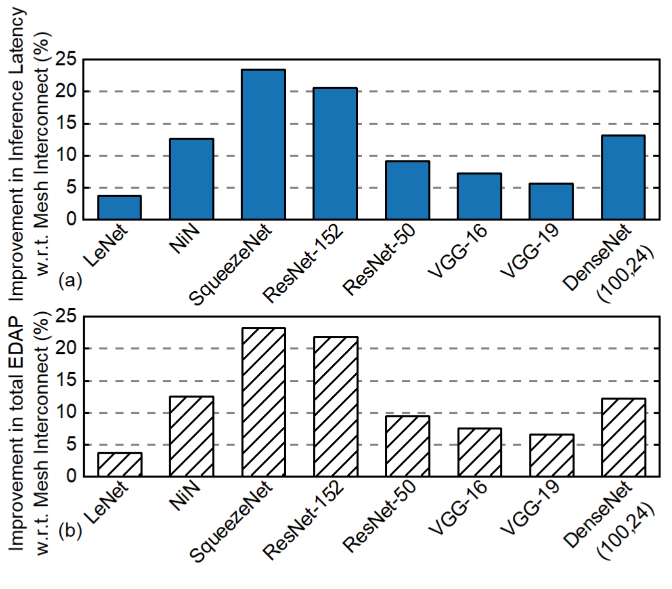


Fig. 17. Overall improvement in (a) total inference latency and (b) total EDAP of a SRAM-based IMC architecture with the proposed latency-optimized interconnect with respect to the baseline.

E. Overall Improvement

In this section, we discuss the overall improvement in inference performance for an SRAM-based IMC architecture with our proposed latency-optimized NoC architecture. Figure 17(a) shows the improvement in total inference latency with respect to the baseline architecture with mesh-NoC. The improvement is in the range of 5%-25% for different DNNs. We observe higher improvement for SqueezeNet and ResNet-152. These two DNNs have a higher number of activations between layers compared to other DNNs. Higher number of activation leads to higher communication volume, which in turn results in more congestion for mesh-NoC. In contrast, our proposed latency-optimized NoC schedules the packets in such a way that there is no congestion in the NoC leading to significant improvement over mesh-NoC. The efficiency of IMC architecture with mesh-NoC over [6], [20] is shown in [26]. Moreover, we observe that our proposed NoC architecture results in 13%-85% improvement in inference latency with respect to the IMC architecture with bus-based H-Tree interconnect [6], [12].

Figure 17(b) shows the improvement in total system EDAP for an SRAM-based IMC architecture with proposed latency-optimized NoC for all DNNs. Since improvement in inference latency is higher compared to the improvement in energy and area, we observe that the improvement in EDAP follows a similar trend as that of inference latency. On average, the proposed latency optimized NoC delivers 9.8% improvement in overall EDAP with respect to baseline architecture with mesh-NoC. Since interconnect plays an important role in overall performance of an IMC architecture, the proposed NoC architecture contributes to a considerable improvement in overall inference performance for different DNNs.

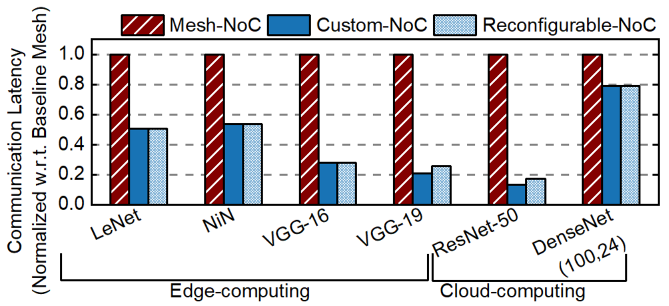


Fig. 18. Results of leave-one-out experiments with reconfigurable NoC for edge computing- and cloud computing-based DNNs.

F. Results With the Reconfigurable NoC

In this section, we show the results of our proposed reconfigurable NoC. We identify two broad class of DNNs, namely, edge-based and cloud-based DNNs. We categorize the DNNs based on its application on edge-computing or cloud-computing based devices. We consider LeNet, SqueezeNet, NiN, VGG-16 and VGG-19 in the category of edge-based DNNs and ResNet-50, ResNet-152 and DenseNet (100, 24) in the category of cloud-based DNNs. We assume that the circuit part of the IMC architecture is reconfigurable and supports the specific class of DNNs under consideration. We perform leave-one-out experiment to evaluate our proposed reconfigurable NoC. For example, while performing experiment for VGG-19 (edge computing-based DNN), we assume that information of VGG-19 is not available at design time.

The number of layers for a reconfigurable NoC for edge-computing is set at 26. For each layer, we set the number of routers as the maximum number of routers required for all DNNs for that particular layer. For example, for 1st layer, the optimal number of routers required for LeNet, NiN, SqueezeNet, and VGG-16 are 4, 2, 5, and 4, respectively. Therefore, we allocate 5 routers for 1st layer.

At runtime, on encountering the new DNN (VGG-19), we execute Algorithm 1 to generate the NoC schedules and execute VGG-19 with available resources on-chip. For fairness, we perform the same experiment with multiple DNNs as shown in Figure 18. We observe that there is <5% degradation in communication latency for VGG-19 and ResNet-50, while other DNNs have the same performance as that of the custom NoC. Since the optimal number of routers required for a few layers of the DNN may not be present on-chip, there might be a degradation in communication latency with respect to custom NoC. For example, for the experiment with VGG-19, 5 routers are allocated for the 1st layer. However, the custom NoC optimized for VGG-19 requires 6 routers. Since it can use up to 5 routers for the first layer, the communication latency of VGG-19 with reconfigurable NoC is more than the custom NoC. Still, for VGG-19 and ResNet-50, the proposed reconfigurable NoC performs significantly better than the baseline mesh-NoC as shown in Figure 18. We also observe that the runtime overhead of the proposed algorithm ranges from 0.049s (LeNet) to 49.93s (DenseNet). However, the overhead is negligible considering that the reconfiguration is a one-time effort for each DNN. Therefore, the proposed algorithm reconfigures the available NoC resources depending on the DNN being executed and provides significant benefit with respect to the baseline mesh-NoC.

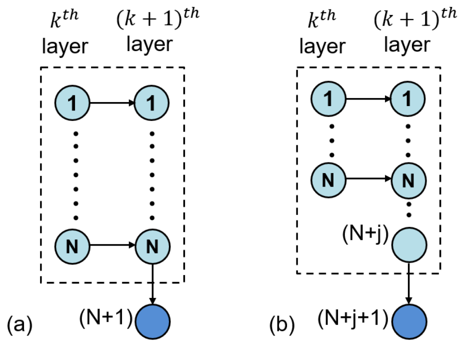


Fig. 19. NoC architecture to achieve minimum latency for Case 2. (a) shows the case when there is one more router in $(k + 1)^{th}$ layer than k^{th} layer. (b) shows the general case. The dotted box shows the optimal architecture (already proved) and the circles filled with dark color represent the newly added router.

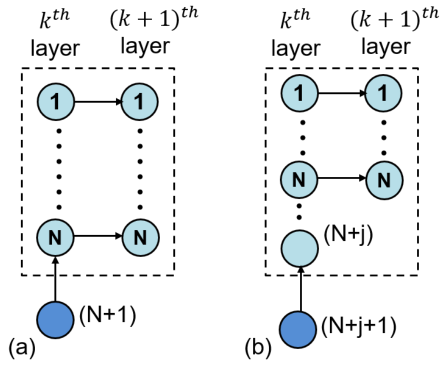


Fig. 20. NoC architecture to achieve minimum latency for Case 3. (a) shows the case when there is one more router in $(k + 1)^{th}$ layer than k^{th} layer, (b) shows the general case. The dotted box shows the optimal architecture (already proved) and the circles filled with dark color represent the newly added router.

VI. Conclusion

In this work, we present a latency-optimized reconfigurable NoC for in-memory acceleration of DNNs. State-of-the-art interconnect methodologies include bus-based H-Tree interconnect and mesh-NoC. We show that bus-based H-Tree interconnect contributes significantly to the total inference latency of DNN hardware and are not a viable option. Mesh-NoC based IMC architectures are better than bus-based H-tree but they too do not consider the non-uniform weight distribution of different DNNs, DNN graph structure, and the computation-to-communication imbalance of the DNNs. None of the architectures holistically investigated minimization of communication latency. In contrast, our proposed latency-optimized NoC guarantees minimum possible communication latency between two consecutive layers of a given DNN. Furthermore, we proposed reconfigurable NoC for two representative categories of DNNs, namely, edge computing-based and cloud computing-based DNNs. Experimental evaluations on a wide range of DNNs confirm that the proposed NoC architecture enables 60%-80% reduction in communication latency with respect to state-of-the-art interconnect solutions.

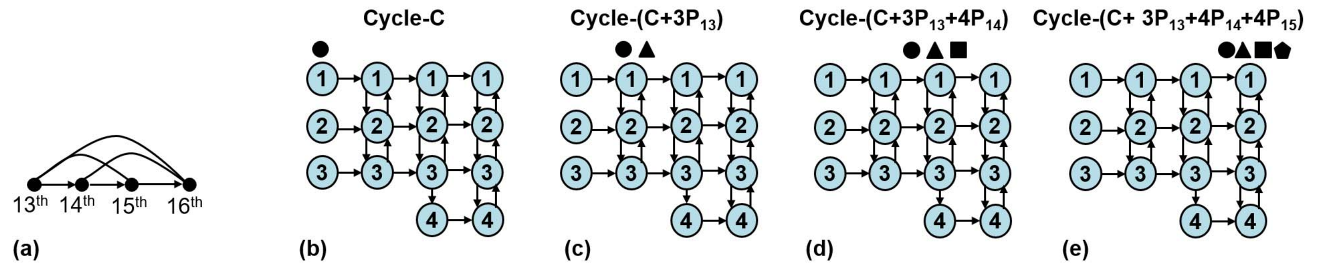


Fig. 21. Operation of the proposed NoC for a section of DenseNet (100,24) [27]. (a) A representative section of DenseNet (100,24). (b)–(e) show the communication between the layers of DenseNet (100,24).

Appendix A

Proof for Case 2 ($\mathbf{N_k < N_{k+1}, N_k = N}$): First, we consider the configuration with $N_{k+1} = N+1$ as shown in Figure 19(a). Since each router of k^{th} layer can send at most one packet per cycle, the minimum number of cycles required to send packets to all routers in the $(k+1)^{\text{th}}$ layer is $N_{k+1} = \max(N_k, N_{k+1})$. The NoC configuration shown in the dotted box is optimal as it has an equal number of routers in both layers. By adding the router $R_{(k+1),(N+1)}$, we add the downward vertical link $L_{(k+1),N} - L_{(k+1),(N+1)}$. The new transactions due to the newly added router will only happen through this link. Specifically, the router $R_{(k+1),(N+1)}$ will receive one packet at each cycle starting from cycle-2 only from the router $R_{(k+1),N}$. The last transaction through this link is $R_{(k+1),1} - R_{(k+1),(N+1)}$ and that will happen in cycle-(N+1). Therefore, the NoC configuration shown in Figure 19(a) completes all transactions in minimum possible cycles and therefore it is optimal.

Next, we assume that the architecture with $N_{k+1} = N + j$ is optimal as shown in the dotted box in Figure 19(b). We will prove by induction that if the architecture with $N_{k+1} = N + j$ is optimal, then the architecture with $N_{k+1} = N + j + 1$ is also optimal which proves the general case. By introducing the router $R_{(k+1),(N+j+1)}$, the downward vertical link $L_{(k+1),(N+j)} - L_{(k+1),(N+j+1)}$ is introduced. The new transactions due to the newly added router will only happen through this link. Specifically, the router $R_{(k+1),(N+j+1)}$ will receive one packet at each cycle starting from cycle-2 only from the router $R_{(k+1),(N+j)}$. The last transaction through this link is $R_{(k+1),1} - R_{(k+1),(N+j+1)}$ and that will happen in cycle-(N+j+1). Therefore, the NoC configuration shown in Figure 19(b) completes all transactions in minimum possible cycles and therefore it is optimal.

Appendix B

Proof for Case 3 ($\mathbf{N_k > N_{k+1}, N_{k+1} = N}$): First, we consider the configuration with $N_{k+1} = N + 1$ as shown in Figure 20(a). Since each router of $(k+1)^{\text{th}}$ layer can receive at most one packet per cycle, the minimum number of cycles required to receive packets from all routers in k^{th} layer is $N_k = \max(N_k, N_{k+1})$. The NoC configuration shown in the dotted box is optimal as it has an equal number of routers in both layers. By adding the router $R_{k,(N+1)}$, we add the upward vertical link $L_{k,N} - L_{k,(N+1)}$. Specifically, the router $R_{k,(N+1)}$ will send one packet at cycle-1 to the router $R_{k,N}$. This packet will be sent to $R_{(k+1),N}$ through the link $L_{k,N} - L_{(k+1),N}$ at cycle-2. We note that this link is free at cycle-2 with the configuration shown in the dotted box. Subsequently, this packet will reach the router $R_{(k+1),1}$ at cycle-(N+1) which is the last transaction with this NoC configuration. Therefore, the NoC configuration shown in Figure 20(a) completes all transactions in minimum possible cycles and therefore it is optimal.

Next, we assume that the architecture with $N_k = N + j$ is optimal as shown in the dotted box in Figure 20(b). We will prove by induction that if the architecture with $N_k = N + j$ is optimal then the architecture with $N_k = N + j + 1$ is also optimal which proves the general case. By introducing the router $R_{k+,(N+j+1)}$, the upward vertical link $L_{k,(N+j)} - L_{k,(N+j+1)}$ is introduced. Specifically, the router $R_{k,(N+j+1)}$ will send one packet at cycle-1 to the router $R_{k,(N+j)}$. This packet will reach the router $R_{k,N}$ at cycle-(j+1). In cycle-j it will be sent to the router $R_{(k+1),N}$ and in the subsequent cycle the packet will traverse through the upward vertical link till it reaches the router $R_{(k+1),1}$. The last transaction that will happen in cycle-(N+j+1). Therefore, the NoC configuration shown in Figure 20(b) completes all transactions in minimum possible cycles and therefore it is optimal.

Appendix C

Execution of the proposed network on dense neural network: Figure 21 shows the operation of the proposed NoC on DenseNet [27]. We consider the 13th–16th layer of DenseNet which is a representative section of this DNN. In DenseNet, all neurons in a particular layer are connected with all other neurons in the subsequent layers as shown in Figure 21(a). The number of routers allocated with our proposed methodology for the 13th, 14th, 15th and 16th layers are 3,3,4,4 respectively which is shown in Figure 21(b). Different packets generated in different layers are shown in different markers in Figure 21(b)–21(e). We denote the number of packets to be communicated for each source to destination pair from k^{th} layer to $(k+1)^{\text{th}}$ layer as P_k. We assume that at cycle-C the packets (shown in ●) in the 13th layer is ready to be communicated to the 14th layer. Specifically, at each round of communication, each router in 13th layer will send one packet (marked with ●) to each router in 14th layer. According to our proposed approach, each round of communication between 13th and 14th layer will take 3 cycles ($\max(N_{13}, N_{14}) = \max(3, 3) = 3$). Therefore, one round of communication will be finished at cycle-(C+$3P_{13}$). After that, the computations are performed in the 14th layer. Without loss of generality, we assume that computations are performed in the same cycle the packets (activations) reach the layer. After computations, new packets are generated which are to be communicated to the next layer. The new packets are denoted by the triangular shaped marker. Each type of the packets marked with circular and triangular shaped marker are to be communicated to 15th layer. Each round of communication takes 4 cycles ($\max(N_{14}, N_{15}) = \max(3, 4) = 4$). Therefore, all the packets reach 15th layer at cycle-(C $+3P_{13}+ 4P_{14}$). After that, the computations are performed in the 15th layer and the new packets (shown in rectangular shaped marker are generated. Similarly, the packets will reach the 16th layer at cycle-(C $+3P_{13} + 4P_{14} + 4P_{15}$) and upon computation, new packets will be generated (shown in pentagon filled in black). Thus the proposed NoC architecture works seamlessly for densely connected networks.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[2] C. D. Manning, C. D. Manning, and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: MIT Press, 1999.

[3] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[4] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with *in-situ* analog arithmetic in crossbars," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 14–26.

[5] S. Yin, Z. Jiang, M. Kim, T. Gupta, M. Seok, and J.-S. Seo, "Vesti: Energy-efficient in-memory computing accelerator for deep neural networks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 1, pp. 48–61, Jan. 2020.

[6] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 541–552.

[7] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A 64-tile 2.4-Mb in-memory-computing CNN accelerator employing charge-domain compute," *IEEE J. Solid-State Circuits*, vol. 54, no. 6, pp. 1789–1799, Jun. 2019.

[8] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1284–1293.

[9] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*. [Online]. Available: http://arxiv.org/abs/1611.01578

[10] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.

[11] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 12, pp. 3067–3080, Dec. 2018.

[12] M. Mao, X. Peng, R. Liu, J. Li, S. Yu, and C. Chakrabarti, "MAX2: An ReRAM-based neural network accelerator that maximizes data reuse and area utilization," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 398–410, Jun. 2019.

[13] J. Jeffers, J. Reinders, and A. Sodani, *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*. San Mateo, CA, USA: Morgan Kaufmann, 2016.

[14] Y.-H. Chen, T.-J. Yang, J. S. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[15] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–12.

[16] L. Yang, Z. He, Y. Cao, and D. Fan, "Non-uniform DNN structured subnets sampling for dynamic inference," in *Proc. 57th ACM/IEEE Design Automat. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[17] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[18] Z. Fang, D. Hong, and R. K. Gupta, "Serving deep neural networks at the cloud edge for vision applications on mobile platforms," in *Proc. 10th ACM Multimedia Syst. Conf.*, Jun. 2019, pp. 36–47.

[19] W.-S. Khwa *et al.*, "A 65 nm 4 Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3 ns and 55.8 TOPS/W fully parallel product-sum operation for binary DNN edge processors," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 496–498.

[20] X. Qiao, X. Cao, H. Yang, L. Song, and H. Li, "AtomLayer: A universal ReRAM-based CNN accelerator with atomic layer computation," in *Proc. 55th ACM/ESDA/IEEE Design Automat. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[21] X. Peng, R. Liu, and S. Yu, "Optimizing weight mapping and data flow for convolutional neural networks on RRAM based processing-in-memory architecture," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[22] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "FloatPIM: In-memory acceleration of deep neural network training with high precision," in *Proc. 46th Int. Symp. Comput. Archit.*, Jun. 2019, pp. 802–815.

[23] L. Ni, H. Huang, Z. Liu, R. V. Joshi, and H. Yu, "Distributed in-memory computing on binary RRAM crossbar," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, pp. 1–18, May 2017.

[24] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 461–475, Nov. 2018.

[25] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 7, pp. 1354–1367, Jul. 2018.

[26] G. Krishnan, S. K. Mandal, C. Chakrabarti, J.-S. Seo, U. Y. Ogras, and Y. Cao, "Interconnect-aware area and energy optimization for in-memory acceleration of DNNs," *IEEE Des. Test. Comput.*, early access, Jun. 11, 2020, doi: 10.1109/MDAT.2020.3001559.

[27] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.

[28] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[29] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*. [Online]. Available: http://arxiv.org/abs/1312.4400

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[31] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[32] A. Krizhevsky, "Learning multiple layers of features from tiny images," M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.

[33] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*. [Online]. Available: http://arxiv.org/abs/1602.07360

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[35] N. Jiang *et al.*, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2013, pp. 86–96.