Activation in Network for NoC-based Deep Neural Network Accelerator

Wenyao Zhu, Yizhi Chen, and Zhonghai Lu
Department of Electrical Engineering, KTH Royal Institute of Technology, Stockholm, Sweden
Email: wenyao@kth.se, yizhic@kth.se, zhonghai@kth.se

***Abstract*—Network-on-Chip (NoC) based Deep Neural Network (DNN) accelerators are widely adopted, but their performance is still not satisfactory as the network congestion may enlarge the inference latency. In this work, we leverage the idea of in-network processing and propose a computation-while-blocking method to conduct activation in network that improves inference latency for NoC-based DNN accelerators. Our approach offloads the non-linear activation from processing elements (PEs) to network routers. Based on a cycle-accurate NoC-DNN simulator, we experiment on a popular neural network model LeNet. The proposed approach can achieve up to 12% speedup in the first layer, and an overall around 6% decrease in total cycles compared to the baseline.**

***Index Terms*—Deep neural networks, Network-on-Chip, DNN accelerator, In-network processing**

I. Introduction

Modern Deep Neural Network (DNN) models usually contain millions of parameters and run billions of computations for a single inference process, therefore hardware accelerators are designed to reduce its latency. As each neuron performs simple multiply-accumulation (MAC) operations in a large amount, most DNN accelerators employ hundreds of processing elements (PEs) [1] in parallel to increase throughput.

Network-on-Chip (NoC) based DNN accelerators (NoC-DNN) have recently been an active research area. The majority of such implementations perform careful co-design with PE and NoC to optimize data traffic aiming for specific DNN layer structures [2]. In this case, most efforts on NoC-DNN have been put into topology designs to support flexible communication, and reduce throughput and power/energy overheads [3]–[6]. On the other hand, their performance is still not optimized from the NoC perspective due to the data communication delay, especially in the network congestion situation.

With the inspiration of in-network processing [7], some of the necessary operations in DNNs, such as non-linear activation functions, can be conducted in network routers with minimal influence on transmission delay to decrease the overall latency. In this paper, we propose a computation-while-blocking technique to conduct activation in network (AiN) that can be seamlessly integrated with current NoC-based DNN accelerator designs to improve the inference speed. We summarize our contributions as follows.

- We propose a novel in-network processing technique, specifically, in-network activation for NoC-DNN designs.
- We modify the typical NoC virtual channel (VC) router architecture to offload the non-linear activation function from PEs to routers.
- We evaluate our technique on a cycle-accurate NoC-DNN simulator and discuss the possible performance improvement for different DNN layers.

We discuss the related work in Section II. Then we describe the in-network activation approach and evaluation results in Section III and IV, respectively. We conclude in Section V.

II. Related work

NoC-based hardware accelerators are common for general-purpose deep learning applications with myriad layer configurations [2]. Recent NoC-DNN works have exerted great effort into energy efficiency optimization. Chen *et al.* present the famous Eyeriss V2 [4] DNN accelerator architecture with a hierarchical 2D mesh NoC to increase data reusability and PE utilization. Zhu *et al.* implement a NoC-based DNN accelerator with the sparse skipping technique to reduce memory access overhead [6]. Other researchers focus on increasing computing flexibility and throughput for different traffic patterns used in NoC-DNNs. Multiply-Accumulate Engine with Reconfigurable Interconnect (MAERI) [3] employs tree-based topology to support various DNN partitions and mappings via tiny configurable switches. However, as the tree topology increases the time complexity for data communication, the scalability of MAERI is not as good as mesh-based NoC. In [5], a reconfigurable mesh-based NoC-DNN accelerator is proposed to cover different kernel sizes for convolution operation. It also adopts the channel-based weight-wise convolution and input-oriented approach to reuse the input and weight, which reduces memory access and improves throughput.

These hardware designs have mainly focused on multiply-accumulation (MAC) operations in the spatial PE array architecture. Other operations, such as the non-linear activation, are overlooked though they appear as an important part of the DNN inference process. These operations are simply attached in PE in their optimized NoCs. The concept of in-network processing [7] points out the feasibility of offloading such computations to the network to reduce the end-to-end latency. Lu presents in-network packet generation to engage NoC actively in computation to reduce lock coherency overhead [7]. Zheng *et al.* develop IIsy for in-network classification using off-the-shelf programmable switches [8]. It can run small models such as Random Forest purely inside the network without any back-end cores. But for DNNs, it is not applicable.

In contrast to the above works, we propose a new technique to improve the NoC-based DNN accelerator design by offloading part of the DNN calculations from PEs to routers.

III. Activation in Network Design

A. NoC-DNN architecture overview

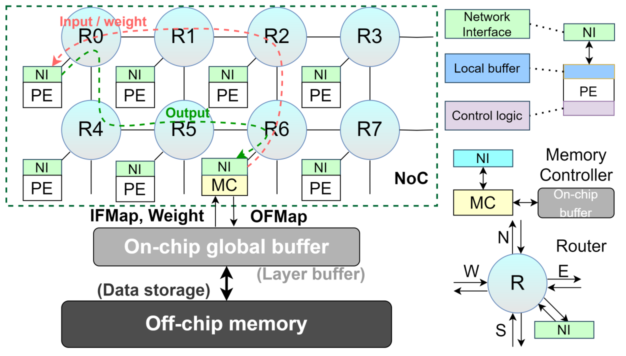

Fig. 1: NoC-DNN accelerator architecture.

NoC-based DNN accelerator mainly involves three components, including memory hierarchy, network, and processing cores. **Fig.** 1 illustrates the NoC-DNN accelerator architecture, which is interconnected via a 2D mesh network. There are generally two types of cores. PE cores consist of the control logic, arithmetic logic units (ALUs), and local buffers, which are used for mathematical computations. Memory controller (MC) cores let NoC access the on-chip global buffer. Routers are linked with cores through the network interface (NI).

B. Non-linear activation offloading motivation

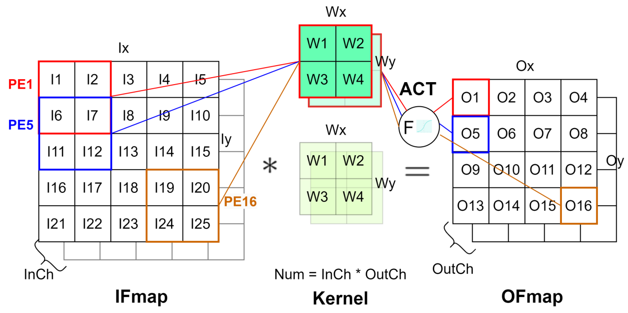

Fig. 2: Output stationary convolution process.

Fig. 2 shows a typical output stationary process for a convolutional layer in DNN, where each PE is mapped with one output neuron at a time. Therefore, the operation inside PE can be formulated as $O = F(\sum(I \times W))$. Here I represents the input features, and W represents the kernel weights. Once the MAC operation is done, the non-linear activation function F, such as ReLU or Sigmoid, is applied to get the neuron output. Recent state-of-the-art NoC-based DNN accelerators normally place an additional non-linear activation function block inside each PE or alongside the PE array in a computation tile [3]–[5]. Both placements accommodate the computation process that non-linear activation happens at the end of processing one neuron output to ensure correctness.

In NoC-DNN, the inference latency consists of the processing time in PE, transmission time in NoC, and memory access time. Modern NoCs may use Quality of Service (QoS) mechanisms such as priority-based services to speed up critical packets so as to improve application performance [9], [10]. However, network congestion still exists, particularly in hot spot regions, e.g., near MC cores for gathering data, leading to higher transmission latency and likely forcing PE to stall. Inspired by the in-network processing concept, we can actually exploit such latency-induced opportunity by moving part of the computation from PE to the network. In this case, we propose offloading the non-linear activation functions to network routers, considering that these functions are the last step of neuron computation and can be efficiently implemented using approximate computing logic or lookup tables.

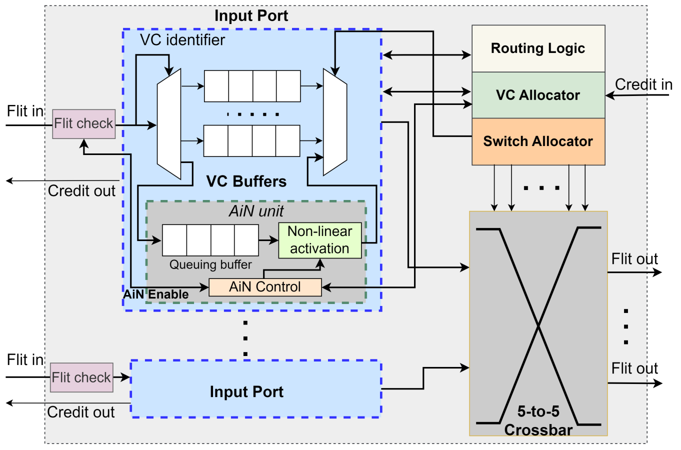

Fig. 3: Proposed AiN VC router architecture.

C. Non-linear activation-in-network (AiN) design

Our AiN technique utilizes the packet congestion time for calculating non-linear activation. To realize our idea, we modify a canonical VC router architecture from [11]. **Fig.** 3 depicts the new AiN design with two additional units in the router. First is the flit check unit for input flits that controls the multiplexer for VC buffer selection. Second is the AiN unit for flit buffering and non-linear activation operations.

1) Flit check: In a conventional VC router, one flit follows a four-stage pipeline in the order of Routing Computation (RC), VC Allocation (VA), Switch Allocation (SA), and Crossbar Traversal (CT) [11]. AiN will only be activated if the coming flit is blocked and it requires non-linear activation. We place the flit check unit into the routing pipeline at the VA stage. This is because the neuron output packet is normally a short single-flit packet and VA happens only for the head flit.

A one-bit status tag is added to the flit header to indicate the condition for AiN. If non-linear activation is required for its payload, the tag is set to 1, otherwise, it remains 0. Besides the tag, we check the VC buffer occupancy as the normal VA stage does. Assuming that the AiN queuing buffer takes one cycle to pass the flit, while the activation logic takes another cycle to compute, we expect at least 2 cycles of blocking in the input port as a suitable situation. Hence the flit check unit counts the existing flits in all VCs at its input port. If there are larger or equal to three flits waiting during VA stage and queuing buffer is not full, the flit check unit grants an AiN enable signal to all flits with a tag 1. Therefore the multiplexer can pass those flits to the AiN unit instead of VC buffers.

2) AiN unit: The AiN unit includes a queuing buffer, a non-linear activation function block, and a controller. The queuing buffer is set as an extra VC buffer with the same size. During each SA stage, it receives a valid flit from the flit check unit if an AiN enable signal is asserted. For each cycle, if there exists a flit in the queuing buffer, the controller takes one flit out and decodes its payload according to the status tag. Then the activation function block performs non-linear activation for the payload using one cycle. After that, its tag is set to 0 and its priority in VA stage is set to high for the next arbitration.

The flit check and AiN units are placed in each of the input ports that have a neighbor router, so we use four instances if there are four directions as shown in **Fig.** 1. In case there is no congestion in packet trajectory, the non-linear activation will be done at the destination router. Hence an additional set of AiN units is added to the input port connected to NI in MC routers to handle the computing near memory situation.

D. Theoretical performance improvement

We discuss the theoretical improvement of the end-to-end inference latency. Since NoC-DNN uses a layer-completing workflow, we focus on the analysis for a single layer. Convolution layers and fully connected layers are common layers that require non-linear activation. To unify the analysis process, we unfold the convolution layer where each element on the output feature map represents one neuron. We assume that neuron tasks are uniformly assigned on all PEs in NoC. Theoretical inference latency improvement ratio ρ is derived by

$$\rho = 1 - \frac{T_{ar} \times r + T_c + T_{tr}}{T_{ape} + T_c + T_{tr}}, \qquad (1)$$

where T_{ar} is the AiN processing time and r is the ratio of non-congested flits over all flits that require non-linear activation. T_{ape} represents the non-linear activation time in PE. T_c is MAC operation time in PE, and T_{tr} is the end-to-end latency for data transmission. For the same task on the NoC-DNN hardware, T_{ar}, T_{ape}, and T_c can be treated as constant. Then the upper bound of ρ appears when $r = 0$. However, $r = 0$ means AiN valid flits are all queuing in the router, which indicates a much longer T_{tr} than $r = 1$ has. In real NoC-DNN accelerator designs, one of the essential design objectives is to optimize the communication delay. Thus we expect a larger r to shorten T_{tr} for a well-designed NoC.

IV. Evaluation

A. Evaluation setup

To validate the proposed method, we build up a cycle-accurate behavior level NoC-DNN simulator in C++. The VC network design is adopted from the NoC simulator in [10], which is based on Garnet in Gem5 [12]. Then we create PE and MC cores and map the DNN workload as the baseline design. After that, we implement the flit check and AiN functions to VC routers following our scheme. We record the end-to-end latency in both setups for comparison. In this simulation environment, we support common DNN layers including convolutional, pooling, and fully connected layers. We also implement ReLU and Sigmoid functions in AiN units.

1) NoC-based DNN accelerator configuration: The network architecture consists of an 8×8 2D mesh as summarized in **Tab.** I. We arrange cores in a row-major order with ID 0 to 63 and separate the whole network into four 4×4 regions. MC cores are positioned at the center of each region with IDs of (17, 18), (21, 22), (41, 42), and (45, 46). The AiN queuing buffer has a depth of 4 flits. We assume that the router operates at a higher speed than PE, with the NoC running at 2 GHz [10], and PE operates at 200 MHz as in [13].

TABLE I: NoC-DNN configuration

Item	Amount	Configuration
NoC	64 nodes	8×8 mesh network, 2 GHz. X-Y routing. 256-bit link bandwidth. Bidirectional link.
Router	64 routers	4 VCs per port, 4-flit buffers per VC.
Core	64 cores	56 PE cores and 8 MC cores, 200 MHz.

2) Simulator workflow: We use a row-major mapping for DNN neurons to PEs using output stationary data flow. Each PE is assigned one output neuron in a computation round. NoC-DNN requires multiple rounds to complete one layer computation. In each round, the NoC activity to compute one neuron output can be divided into three steps:

(a) In the first step, PE sends a request packet to MC, which involves the neuron ID to fetch weights and inputs from the global buffer.
(b) In the second step, MC sends the required data packet to PE. The packet length depends on the amount of data. For a neuron with one 5×5 convolution kernel, the payload consists of 25 inputs, 25 weights, and 1 bias. Suppose we use a 16-bit fixed-point format and the package header is 16 bits for routing information, we need $\lceil \frac{(2\times25+1)\times16+16\ (bit)}{256\ (bit/flit)} \rceil = 4$ flits.
(c) In the third step, PE sends the neuron output packet back to MC. Each PE is designed to calculate 25 MACs per PE cycle. In the baseline, the non-linear activation takes another PE cycle. In our AiN design, the activation step is offloaded to the router and it takes one router cycle.

TABLE II: LeNet structure for NoC-DNN

Layer	#Neuron	#Round	#OP	#ACT	ACT func.
Conv1	4704	84	$(5 \times 5) \times 4704$	4704	ReLU
MaxP1	1176	21	$(2 \times 2 \times 6) \times 1176$	-	-
Conv2	1600	29	$(5 \times 5 \times 6) \times 1600$	1600	ReLU
MaxP2	400	8	$(2 \times 2 \times 16) \times 400$	-	-
FC1	120	3	400×120	120	ReLU
FC2	84	2	120×84	84	ReLU
FC3	10	1	84×10	10	Sigmoid

3) DNN model: We choose a well-known CNN model, LeNet-5 [14] on MNIST classification, to demonstrate our approach. LeNet consists of 7 layers in total, comprising two convolutional (Conv) layers, two pooling (MaxP) layers, and three fully connected (FC) layers. As shown in **Tab.** II, we count the MAC (for Conv and FC layers) or Maximum (for Max. pooling layers) operations (OP), non-linear activation function (ACT), and computation rounds within each layer. Computation rounds are the number of output neurons divided by available PEs. In our case, the formula is #Round $=$ $\lceil$#Neuron/56$\rceil$. MaxP contains no non-linear activation, hence it is not relevant to our method.

B. Evaluation results

We first assess the classification results of LeNet through a comparative analysis between the outcomes from Python and our simulator. The layer output and final output consistency validate the correctness of our simulator. Then we record the end-to-end inference latency based on two setups: the baseline and our proposed approach.

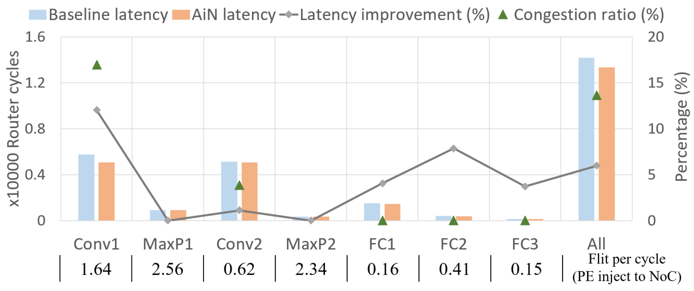

Fig. 4: NoC-DNN evaluation results on LeNet.

We present the results in **Fig.** 4. The bar plot shows the end-to-end inference latency for different layers and the whole model. The line plot shows the latency improvements in percentage. Among the seven layers, two convolution layers constitute the major computational load in both tests. This observation aligns with the computational complexity of layers.

Our proposed approach demonstrates a notable 5.97% latency reduction for the entire LeNet model compared with the baseline. The maximum latency improvement per layer, illustrated in **Fig.** 4, reaches 12.02% in the Conv1 layer. Furthermore, the FC layers exhibit improvements ranging from 3.70% to 7.84%. Since pooling layers do not involve activation computations, there is no difference between the two tests.

C. Discussion

According to **Eq.** (1), r and T_{tr} determine the latency improvement ratio. The averaged congestion ratio $(1-r)$ that counts the congested packets among all third-step packets from PE to MC is plotted in green in **Fig.** 4. Since AiN doesn't apply to pooling layers, their congestion ratio is not shown.

DNN layers with higher network activities present greater opportunities for enhancing performance by our AiN method. As shown in **Fig.** 4, Conv layers have more PE to MC traffic than FC layers, which increases the back pressure to block flits in NoC and results in a lower r. Thus Conv layers can benefit more from AiN than FC. Though AiN is seldom activated in FC layers with nearly zero blocking situations, they still enjoy a modest speedup. This is attributed to routers running at a higher frequency than PEs. In our case, FC layers have $r \approx 1$, and T_{ape} is ten times of T_{ar}. Comparing FC2 to the other two FCs, it has a higher flit injection rate, which means a shorter $T_c + T_{tr}$. Hence it enjoys a greater improvement ρ in consistency to **Eq.** (1). Conv1 has a higher congestion ratio than Conv2. Additionally, since the second-step packet size in Conv2 is six times larger than Conv1, it prolongs overall T_c and T_{tr}. This can cause PE stalls and hinder the benefits of AiN, which leads to an even smaller latency improvement.

In summary, the observed latency improvement aligns with the theoretical equation, particularly benefiting DNN layers with frequent NoC accesses and shorter processing times, such as small-kernel convolution, through the proposed AiN.

V. Conclusion

In this paper, we present an in-network processing method for NoC-based DNN accelerators. A modified VC router micro-architecture is designed to realize the non-linear activation offloading approach for DNN inference latency optimization. We build a NoC-DNN simulator to evaluate our AiN approach. The experimental results show a near 6% latency reduction on LeNet. Besides, the theoretical and practical latency improvement using AiN are discussed. Our work reveals the opportunity of in-network processing in DNN accelerator designs. We believe that other operations can also be offloaded to the network to reduce inference latency. Their RTL implementations and tradeoff between timing and power/area will be investigated in the future.

Acknowledgment

The research has been supported in part by Vetenskapsrådet (Swedish Research Council) through the LearnPower project (2020-03494).

References

[1] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[2] S. M. Nabavinejad, M. Baharloo, K.-C. Chen, M. Palesi, T. Kogel, and M. Ebrahimi, "An overview of efficient interconnection networks for deep neural network accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, pp. 268–282, 2020.

[3] 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, 2018.

[4] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[5] K.-C. Chen and Y.-S. Liao, "A reconfigurable deep neural network on chip design with flexible convolutional operations," in *Proceedings of 15th International Workshop on NoC Architectures*, 2022, pp. 1–5.

[6] L. Zhu, W. Fan, C. Dai, S. Zhou, Y. Xue, Z. Lu, L. Li, and Y. Fu, "A NoC-based spatial DNN inference accelerator with memory-friendly dataflow," *IEEE Design&Test*, 2023.

[7] Z. Lu, "PiN: Processing in Network-on-Chip," *IEEE Design&Test*, 2023.

[8] C. Zheng, Z. Xiong, T. T. Bui, S. Kaupmees, R. Bensoussane, A. Bernabeu, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "IIsy: Practical in-network classification," *arXiv preprint arXiv:2205.08243*, 2022.

[9] S. K. Mandal, A. Krishnakumar, R. Ayoub, M. Kishinevsky, and U. Y. Ogras, "Performance analysis of priority-aware NoCs with deflection routing under traffic congestion," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[10] B. Wang and Z. Lu, "Flexible and efficient QoS provisioning in AXI4-based network-on-chip architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, pp. 1523–1536, 2021.

[11] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks*. Elsevier, 2004.

[12] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009, pp. 33–42.

[13] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, 2018.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.