3

A scalable and flexible interconnect-based dataflow architecture for Edge AI Inference

Rohit Prasad and Hana Krichene

Université Paris-Saclay, CEA-List, France

Abstract

The scalable approach of edge artificial intelligence (AI) inference, especially Convolutional Neural Network (CNN) for computer vision and image recognition functions, has increased its computational complexity due to the involvement of multiple properties, i.e., input image size, choice of the filter size, zero padding, and strides. Dataflow architectures based on many Processing Elements (PEs) are considered promising solutions to execute CNNs, efficiently offering high parallelism and bandwidth. However, the existing dataflow architectures are generally specialised with difficulty in achieving scalability and flexibility. This work proposes an interconnect-based dataflow architecture to overcome such problems. The proposed architecture can efficiently handle convolutions featuring different input image/feature-map shapes and filters, with data reuse and communication-computation overlap. It is scalable and configurable to adapt to different CNN layers. The experimental results show that the proposed architecture can accelerate LeNet5 convolution layers by up to 71.2× in latency performance w.r.t. a RISC-V-based CPU and that it also accelerates MobileNetV2 convolution layers by up to 2.07× in latency performance w.r.t. a dataflow architecture featuring row-stationary execution style.

Keywords: dataflow architecture, data reuse, CNN accelerator, interconnect, hardware accelerator.

3.1 Introduction

CNNs used for Deep Learning (DL) are increasingly achieving higher accuracy in processing modern Artificial Intelligence (AI) applications such as computer vision [4] and image recognition [6]. Existing CNNs face problems of high computational complexity and large amounts of data to process. The problem becomes more critical in the context of Edge AI due to the availability of limited resources. Dataflow architectures [10] are presented as a promising solution to provide high parallelism with high throughput while facilitating the movement of shared data via Network-on-Chip (NoC). These architectures are based on massive PEs performing an elementary function, e.g., Multiply-ACcumulate (MAC) operation in the case of CNN processing. Due to the large amount of data in CNN processing, NoC [8] ensures data exchange among PEs, and between PEs and memories. On the one hand, existing dataflow architectures are specialised and not very flexible for a given CNN. Ensuring the flexibility of the architecture and data transfer with the least delay without compromising the computation are the keys to improving the performance of those dataflow architectures.

We propose an interconnect-based dataflow architecture in this work. The proposed architecture comprises distributed memories and an array of PEs interconnected via a mesh interconnect network. The underlying interconnect network provides high bandwidth and injection rate to achieve simultaneous data transfers via parallel routing paths. The interconnect network also provides flexible data transfers with different directions of multicast and broadcast. Thanks to this interconnect network, the proposed architecture can scale and fit the ever-increasing size of ever-evolving neural networks and accommodate the amount of data generated. The employed interconnect network is configurable, allowing the proposed architecture to handle different CNN shapes and layers of shapes of the same CNN and optimise the PE utilisation by generating a suitable design configuration for a given convolution processing. This benefits the proposed architecture to maximise the reuse of shared data and the communication-computation overlap.

The remainder of the paper is organised as follows, section 1.2 discusses the related work, section 1.3 presents the background work, section 1.4 describes the sub-system of the proposed architecture, section 1.5 explains the execution model of the proposed architecture, section 1.6 presents the evaluation methodology, details the experiments, and analyses the performance results. Finally, section 1.7 presents a conclusion.

3.2 Related Work

Typically, in dataflow architecture, a program is seen as a collection of data nodes where a data flow node is executed when all its inputs are ready. Once a result is ready at a data node, copies of the result are distributed to their destination operators. This chain of operations is performed in sequential order until the end of execution of the program is reached. There is no separate control flow, and operations scheduling requires in-depth knowledge of the underlying algorithm of the program. The scope of this section is limited to the dataflow architectures that target the acceleration of CNNs only.

In [16], a dataflow architecture called Eyeriss is presented that aims to save energy consumption by minimising the data movement on a PE array. Eyeriss exploits the reuse of filter weights and feature map pixels in the convolutions to minimise the data movement due to the accumulations of partial sums. Eyeriss limits the dimensions of input data due to its fixed PE array size.

The DianNao series, i.e., DianNao [13], DaDianNao [14], and ShiDianNao [19] aim to increase the efficiency of the system by minimising the latency of memory accesses. DaDianNao minimises the main-memory access latency by implementing a large on-chip embedded Dynamic Random Access Memory (eDRAM). DaDianNao is targeted at the data center solution. ShiDianNao targets the acceleration of CNN applications by mapping the parameters onto a smaller on-chip Static Random Access Memory (SRAM). ShiDianNao has a better energy efficiency than DaDianNao by 60$\times$ because the former avoids memory access to DRAM by storing data in SRAM.

Maeri [7] is another dataflow architecture that augments multiplication and addition operators with tiny switches, and communication is available to them using a reconfigurable interconnect network. Maeri can execute convolution, Long Short-Term Memory (LSTM), pooling, and fully connected layers. It supports cross-layer mapping and also addresses sparsity in the network.

These architectures feature a different approach for mapping and executing convolution layers. In addition to these architectures, the proposed architecture features the overlap between communication and computation, and a scalable PE array to adapt to different input data sizes.

3.3 Background: dataflow execution models

In DNN, the same data is used at multiple input data locations. If repeated accesses of the temporal data from memory are performed, such repetition

can degrade the performance of a sub-system in terms of latency and energy consumption. Dataflow architectures can stand out in such situations because these architectures can efficiently exploit data reuse to avoid repeated memory accesses of temporal data. There are four ways dataflow architectures exploit data reuse, which is defined in the existing convolution neural networks, i.e., (1) Weight Stationary (WS), (2) Input Stationary (IS), (3) Output Stationary (OS), and (4) Row Stationary (RS). Below is a short description of each dataflow model:

1. *WS dataflow model* exploits the filter weight data reuse. NeuFlow [3] implements such a dataflow model.
2. *IS dataflow model* exploits the data reuse by distributing image/input feature maps (ifmaps) to multiple processing elements (PEs). SCNN [1] implements such a strategy to handle sparse CNNs, where multiple filter weight data are zeros.
3. *OS dataflow model* exploits data reuse by broadcasting filter weights, and ifmaps are reused throughout the PE array. ShiDianNao [19] implements such a strategy, where each PE produces the outputs by sharing ifmap data from the neighbouring PEs.
4. *RS dataflow model* reuses ifmaps, filter, and partial sum (intermediate result) to accelerate convolutions. Eyeriss [15] implements such a strategy. Data reuse is performed by sending filter data horizontally and ifmap data diagonally. Once all the PEs in the array have received their respective data, execution begins. Partial sums are accumulated by moving them vertically in each PE column. The dimension of the PE array is determined by filter size and output feature map (ofmap) size for a particular layer. The proposed architecture also uses the RS dataflow model to exploit data reuse in the execution of convolutions.

3.4 Interconnect-based dataflow architecture

Figure 3.1 (a) represents the proposed architecture, which includes (1) a Neural Global Controller (NGC), (2) a Neural Processing Element (NPE) array connected through (3) an Artificial Intelligence Network on Chip (AINoC), and (4) Global Buffers (GB) for storing input and output data. The rows of the filter and rows of the ofmap determine the size of the NPE array, e.g. if the filter size is 6×6, then the number of rows in the NPE array would be 6, and the number of columns would be equal to the number of rows in ofmap,

which is computed using the equation below:

$$ofmap\_rows = (((ifmaps\_rows - (filter\_rows + padding\_start + padding\_end))/stride) + 1)$$

3.4.1 NGC: Neural Global Controller

The NGC is a five-stage Finite-State Machine (FSM) shown in Figure 3.1 (b). Following is the description of each stage:

1. *IDLE* represents the idle state of the NGC.
2. *LOAD_config* loads and decodes the configuration line for the current layer, including the size and number of filters and ifmaps for the current layer.
3. *LOAD_filter* starts sending the filter data (i.e., payload) and the control word from the Input GBs (IGBs) to the connected routers. Depending on the opcode, routers either unicast, multicast, or broadcast the incoming payload data to the AINoC.
4. *LOAD_ifmaps* starts sending the ifmaps data and the control word from the IGBs to the connected router. Depending on the opcode, routers either unicast, multicast, or broadcast the incoming payload data to the AINoC. LOAD_filter and LOAD_ifmaps states can be interchanged to provide some flexibility in the data loading order in the proposed architecture.
5. *COMPUTE* starts the MAC operations in the NPEs. Once a Partial Sum (PSum) is computed in the bottom row NPEs, the results are sent to their respective north NPEs along with their control words. The NPEs in the upper row then add the incoming results with their locally computed PSum and send the computed result to their north NPEs. This chain of operations is executed until it reaches the top NPE row, where the final PSum is stored in the Output GBs (OGBs) to be used in the next layer.

3.4.2 NPE: Neural Processing Element

The NPE is a simple operator controlled by the FSM of the NGC, as shown in Figure 3.1 (c). It remains in the idle state until the NGC triggers the execution. It proceeds in this way according to the control signals sent by the NGC:

- When the load_filter signal is received, it stores the incoming payload into the filter Register File (RF). Once all the NPEs have received their

corresponding filter data, the top right NPE in the NPE array sends a signal to NGC to jump to the next state.

- When the load_ifmap signal is received, it stores the incoming payload into the ifmaps RF. Once all the NPEs have received their corresponding ifmaps data, the top right NPE in the NPE array sends a signal to NGC to jump to the next state.

When the start_compute signal is received, it begins the MAC operation on the filter and ifmaps data and generates PSum. Then, it either (i) sends the PSum to the north NPE (if bottom row NPE) or (ii) adds the incoming PSum from the south NPE with local PSum and sends the result to the north NPE or OGB (if top row NPE). In this phase, communication-computation overlap is also performed by the NPEs, which received their required data.

At the end of the computation, an end_compute signal will be sent by the last NPE of the array to inform the NGC of the end of the execution, in order to move on to the next execution and the loading of new data.

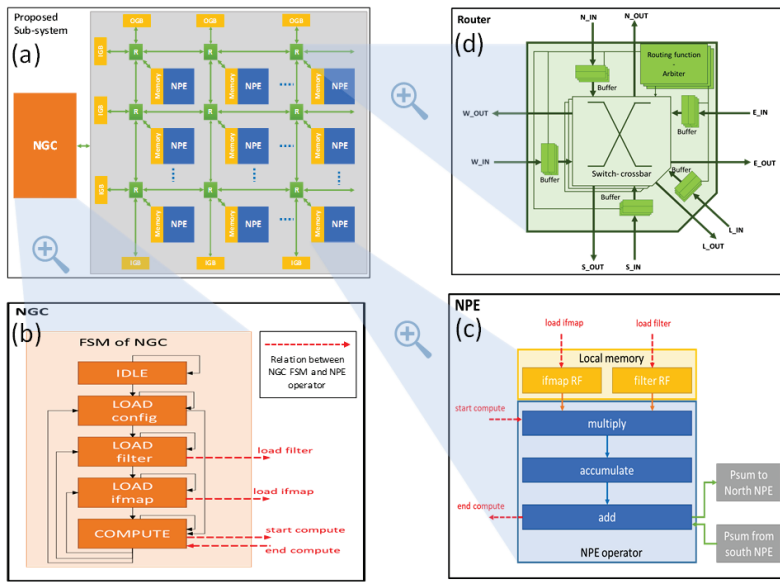

Figure 3.1 (a) The proposed interconnect-based dataflow architecture sub-system, (b) Neural Global Controller (NGC), (c) Neural Processing Element (NPE), (d) Router in Artificial Intelligence Network on Chip (AINoC).

3.4.3 AINoC: Artificial Intelligence Network-on-Chip

The AINoC [9] consists of routers optimised for parallel dataflow processing with minimal data transfer cost to achieve energy-efficient CNN processing without compromising accuracy and application performance.

As shown in Figure 3.1 (d), the routing device is composed of several parallel routing paths, each including a buffer, a communication controller, an arbiter, and a switch. All these paths are designed to guarantee a large bandwidth and flexible communication. Indeed, through several buffering modules, e.g. First-In-First-Out (FIFO), different communication requests received in parallel can be stored without any loss. These requests are then processed simultaneously in several control modules. These modules ensure a deterministic control of the data transfer according to a static X-Y (X-direction priority) routing algorithm and management of different communications (unicast, multicast, and broadcast). Parallel arbitration of the processing order of incoming data packets according to the Round-Robin Arbitration (RRA) [5] based on scheduled access allows for better collision management, i.e., a request that has just been granted, will have the lowest priority on the next arbitration cycle. Parallel switching comes next to simultaneously route data to the right outputs according to the Wormhole switching [11], i.e. the connection between one of the inputs and one of the outputs of a router is maintained until all the elementary data of a message packet are sent and this in a simultaneous way through the different switching modules.

The data packet format is shown in Figure 3.2. A data message consists of two packets: a control packet followed by a data packet. A packet is composed of a header (flit code) and a payload. In the control packet, the payload is a destination or source address, while in the data packet, the payload is a set of data flits. The packet size is 32-bit. However, the size of the header and the payload are variable. It depends on the size of the interconnection network, as the number of routing devices increases, more bits are needed to encode the addresses of the receivers or senders. Similarly, the flit size and number vary with the size of the payloads (filter weights, activation inputs, or PSums) to be passed through the network. The value of the header determines the communication to be provided by the router. There are three possible types of communication inter-PEs: unicast, multicast (horizontal, vertical, and diagonal), and broadcast. For memory access, the

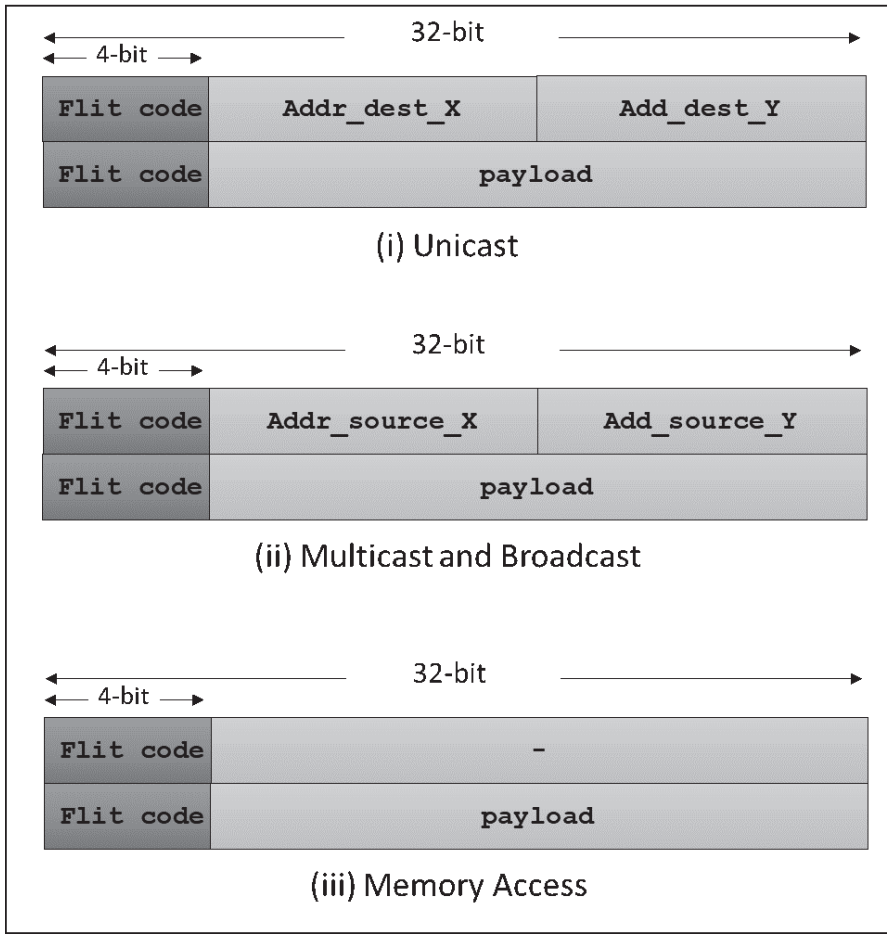

Figure 3.2 Packet format

reading from the IGB is a multicast communication; however, the writing to the OGB is a communication type that processes a direct parallel unicast from the first NPEs rows, and the OGB. The routing device first receives the control packet containing the type of communication and the source or destination address. The routing device decodes this control packet and then allocates the communication path to transmit the data packet that arrives at the cycle following the control packet. Once the data flits are transmitted, the allocated path will be released for further transfers.

3.4.4 Global Buffers

GBs are dual-port Random Access Memory (DPRAM) that are used to store the input data i.e., filter and ifmaps or output data i.e., PSum from top row NPEs. The size of each GB type is determined according to the data size requirement for each layer, such that the overhead due to GB is minimised.

3.5 Execution Model

The data movement and execution pattern in the proposed architecture are presented in this section. Once the data is ready in IGB, the execution in the proposed architecture can be divided into three phases, i.e., (1) load ifmap data into their respective NPEs, (2) load filter data into their respective NPEs, and (3) perform execution on the available data in each NPE. These steps are explained below:

1. *Load ifmap data*: In this phase, ifmap data are loaded into their respective NPEs. Data from IGBs are diagonally loaded into the NPEs, which have connections with them through a single router, and then data reuse is performed by moving the data diagonally to the target NPEs.
2. *Load filter data*: In this phase, filter data are loaded into their respective PEs. Data from IGBs are horizontally loaded into the NPEs, which have connections with them through a single router, and then data reuse is performed by moving the data horizontally to their respective NPEs. During this phase, the overlap between communication and computation is also performed. The NPEs, which receive the required data to compute the partial sum, begin the computation phase. Particularly, the first column of the NPE array gets all the required data and jumps from the communication (i.e., data receiving) phase to the computation phase while other columns still wait for input data.
3. *Execute MAC operation*: When an NPE receives all required data, it jumps from the communication phase to the computation phase. Each column is locally synchronised, where the bottom NPE sends the computed PSum to the north NPE. Each NPE (except the bottom NPE) adds the PSum received from their south NPE with the locally computed PSum before sending the result to their north NPE. This chain of receiving, adding, and sending data is performed until the data reaches the top NPE, where the computed result is stored back into the OGB. NPE array is executing in the Globally Asynchronous Locally

Synchronous (GALS) pattern to enable overlap between communication and computation in the proposed architecture.

3.6 Experiments and Results

3.6.1 Evaluation Methodology

In this work, different CNN algorithms from state-of-the-art were used as case studies. They have different sizes and include different types of layers and shapes. LeNet5 [18] and MobileNetV2 [12] were chosen to have a collection of data resulting from a range of small to large CNN and using a set of layers including classical 2D convolution (CONV2D) and fully connected layers (FC) but also point-wise (PW) and depth-wise (DW) convolution layers in MobileNetV2. Table 3.1 details the characteristics of all these CNN algorithms, including the types of layers they have and the number of each layer type. The values in the proposed architecture configuration are obtained by following the calculation rule presented in section 1.4. In our experimental study, we chose to test the key convolution layers that emphasise different filter sizes and ifmaps and the fully connected layers that require a linear spatial representation of the proposed architecture. We also note that a configuration for the proposed architecture must be generated for each

Table 3.1 CNN Layers type

CNNs	Layer Type	ifmap size	filter shape	Config. of proposed architecture
LeNet5	conv_1	1x32x32	1x5x5	5x28
	conv_2	6x14x14	6x5x5	5x10
	conv_3	16x5x5	16x1x1	1x5
	fc_1	1x1x120	1x120x84	1x84
	fc_2	1x1x84	1x84x10	1x10
MobileNetV2	conv_1	1x128x128	8x[3x3x3]	3x126
	conv_2	8x64x64	8x3x3	3x62
	conv_3	24x64x64	24x3x3	3x62
	conv_4	36x32x32	36x3x3	3x30
	conv_5	48x16x16	48x3x3	3x14
	conv_6	96x8x8	96x3x3	3x6
	conv_7	144x8x8	144x3x3	3x6
	conv_8	240x4x4	240x3x3	3x2
	conv_9	80x4x4	256x1x1	1x4

evaluated layer to respect the RS dataflow execution mode (section 1.3.2). However, the row width for the FC layer (i.e., 1000) of MobileNetV2 is too big for the proposed architecture, due to the limited space allotted to store the value of the number of channels in the configuration word, so this layer has been excluded from our experiments.

3.6.2 FPGA Implementation Results

The evaluation platform used for all tests is the Versal ACAP VCK190 kit [20] featuring an "XCVC1902-2VSVA2197" FPGA partition containing 899840 programmable LUTs, 899840 Flip-Flops, 1968 DSP58, and 158Mb of URAM and BRAM. The software tools used to implement and test different configurations of the proposed architecture are:

- QuestaSim or Questa Advanced Simulator (version 2021.4) from Mentor Graphics is provided to simulate and test the programming and debugging of FPGA chips.
- Vivado Design Suite (version 2021.2) is a software suite produced by Xilinx to synthesise and analyse hardware description language (HDL) designs.

3.6.2.1 Area

The different configurations of the proposed architecture include four main modules: the NGC, distributed memories (IGB & OGB), a given number of NPEs, and routers that are directly connected to the NPEs. All configurations of the proposed architecture are designed with the VHDL description language to be rapidly implemented on FPGA. The implementation results estimate the frequency of the proposed architecture, which is around 125 MHz. This frequency depends on the frequency of the longest critical path in the configuration. A good place and route for the modules of the proposed architecture is necessary to reduce the length of the critical path and accelerate the propagation of the signals. The synthesis results define the occupied area (logic elements, memory, Digital Signal Processing (DSP) blocks, etc.) and the hardware resources consumption of the proposed architecture according to the different configurations defined in Table 3.1.

The synthesis results of the different modules constituting the proposed architecture are given in Table 3.2. Due to the simple structure of the different modules, the consumption of logic and memory resources remains low. This allows generating a configuration of the proposed architecture with a large

grid of computing elements to process large convolution layers. For the GB memories, we opted for the use of Block Random Access Memory (BRAM) by forcing the synthesis tool to choose these memory blocks instead of the configurable logic blocks (CLB). We also notice that the size of the router is relatively larger than the NPE. This can be explained by NPE providing a simple convolution operation. At the same time, the router has multiple routing paths to provide parallel multicast and control of blocking areas in the communication network. These multiple routing paths mainly accelerate the data transfer and reduce the energy consumption during the execution of a convolution layer. It is then a trade-off between area and performance in the proposed architecture. Area can be treated as a small overhead to ensure a balance in the choice of the architecture and the objectives to be achieved.

Table 3.2 Breakdown of Versal ACAP VCK190 FPGA resources used by the modules of the proposed architecture after synthesis

Module	CLB	BRAM	Area occupancy (%)
NGC	12.21	0	0.02
GB	0	0.5	0.05
Router	76.78	0	0.13
NPE	39.10	0	0.07

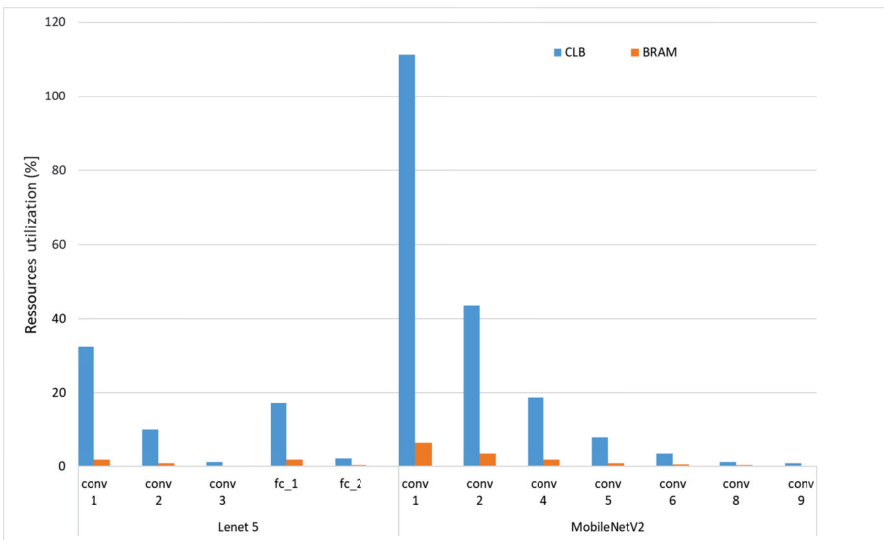

Figure 3.3 Synthesis results of different configurations of the proposed architecture

Figure 3.3 shows the percentages of FPGA resource utilization when executing the different layers of Lenet5 and MobilenetV2 given in Table 3.1. The processing of each type of layer requires a particular configuration of the proposed architecture. A configuration of the proposed architecture depends on the number of filter rows, ifmap rows, and ofmap rows. We observe a correlation between the variation in the size of the proposed architecture and the consumption of the CLBs. The larger the configuration, the greater the resource consumption. The consumption of the BRAM memory blocks depends on the size of the input image or the ifmaps. This means that the memory size remains fixed for a fixed input image/feature-map size, and the size of the filter. Particularly, for the conv_1 of MobileNetV2, we notice that the number of CLBs exceeds the maximum number of CLBs available in the FPGA targeted in these experiments. This representation shows that the proposed architecture remains flexible to support all convolution layer sizes. We just need to aim for a prototyping platform that provides the necessary logic resources for mapping all layers.

3.6.2.2 Latency

Figure 3.4 shows the latency performance of the proposed architecture for each convolution type. It can be observed that the proposed architecture is up to 71.2× (conv_1, Lenet5) faster w.r.t. single RISC-V CPU [2]. The total execution time for each convolution type for the proposed architecture is divided into ifmap loading time, filter loading time, data reuse, and overlap between communication-computation including time required for the PSum to traverse across their respective columns to store the computed ofmap. The breakdown of latency reports that data reuse and overlap between communication and computation significantly improve the overall execution time in the proposed architecture. For latency comparison of the proposed architecture with RISC-V CPU, the time required for access L2 to load data into IGBs is also considered for a fair comparison.

The overall speedup of MobileNetV2 convolution layers is up to 2.07× w.r.t. Eyeriss v2 [16, 17]. Here, Eyeriss v2 executes all layers of MobileNetV2 while the proposed architecture executes convolution layers (Table 3.1). These results are obtained through RTL simulations.

3.6.2.3 Energy consumption

Different hardware modules of the proposed architecture involved in different execution phases for each convolution type are shown in Table 3.3. The explanation of each phase is as follows: (1) Phase A represents data loading

from all IGBs, (2) Phase B represents data reuse, (3) Phase C represents data loading from row IGBs, and (4) Phase D represents computation in NPE array. The results in this section are obtained through hardware emulation.

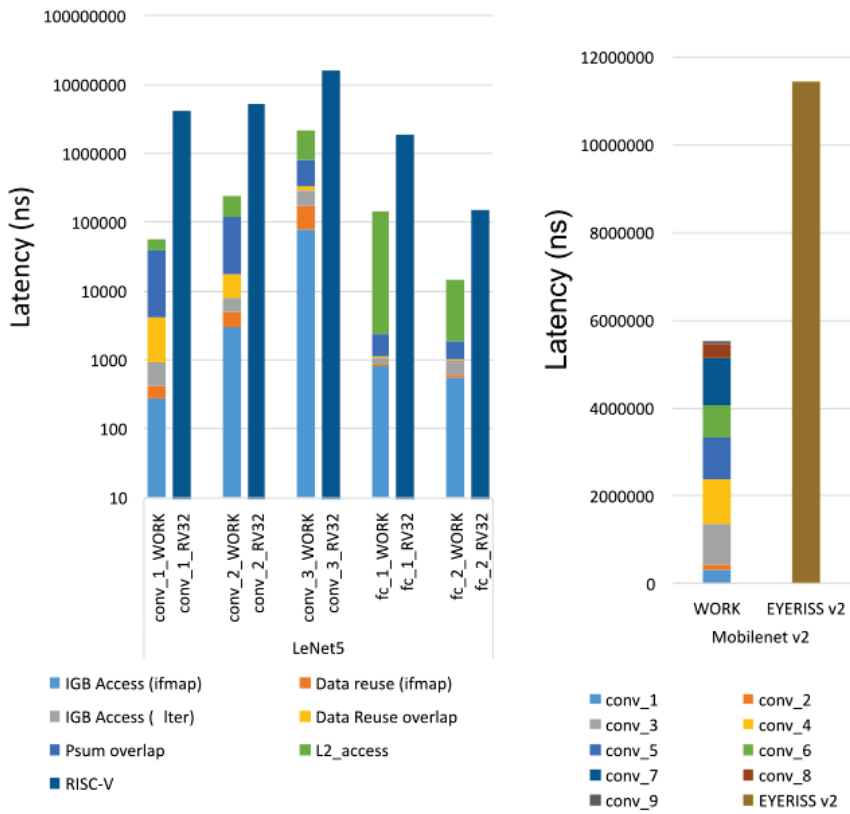

Figure 3.4 Breakdown of latency (ns). For the proposed architecture, the convolution layer includes memory accesses and computations. WORK = This Work, RV32 = RISC-V CPU.

Table 3.3 Different execution phases in the proposed architecture

Execution Phase	NPE array	AINoC	IGB row	IGB column	OGB	NGC
A	X	X	X	X		X
B	X	X				X
C	X	X	X			X
D	X	X			X	X

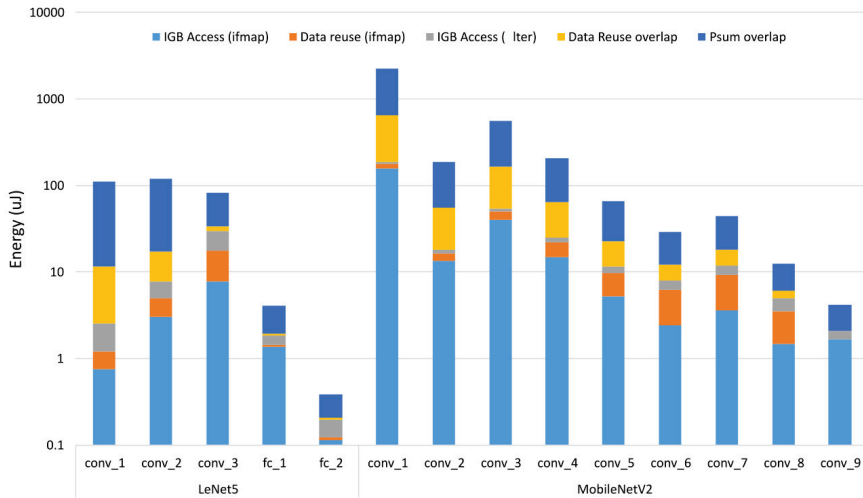

Figure 3.5 Energy consumption (uJ) of the proposed architecture

Using Table 3.3, energy consumption for each execution phase for the proposed architecture is computed. Figure 3.5 shows the total energy consumption of the proposed architecture per convolution layer. It can be observed that a significant energy saving is achieved because following input data loading into the NPEs, which have direct connections with IGBs, the proposed architecture applies data reuse by sending the loaded input data to target NPEs in the array. Notably, a significant energy saving can be observed during the loading of ifmap data because IGBs are not accessed during this phase (Phase B). Due to different design flows i.e., Eyeriss v2 is ASIC and the proposed architecture is FPGA, it is not a fair comparison between the two architectures, and also due to the unavailability of design flow scripts for RISCV CPU [2], we concluded to exclude the energy consumption comparisons for both architectures with the proposed architecture.

3.6.2.4 Energy efficiency

The proposed architecture can reach up to 2498 MOPS/W on an FPGA target for fc_1 of LeNet5 because of a single row with a large number of columns configuration, i.e., 1×84 (Figure 3.6). However, conv_2 of LeNet5 and conv_9 of MobileNetV2 have the lowest energy efficiency because in these layers there is less scope for optimized computation due to a low ratio

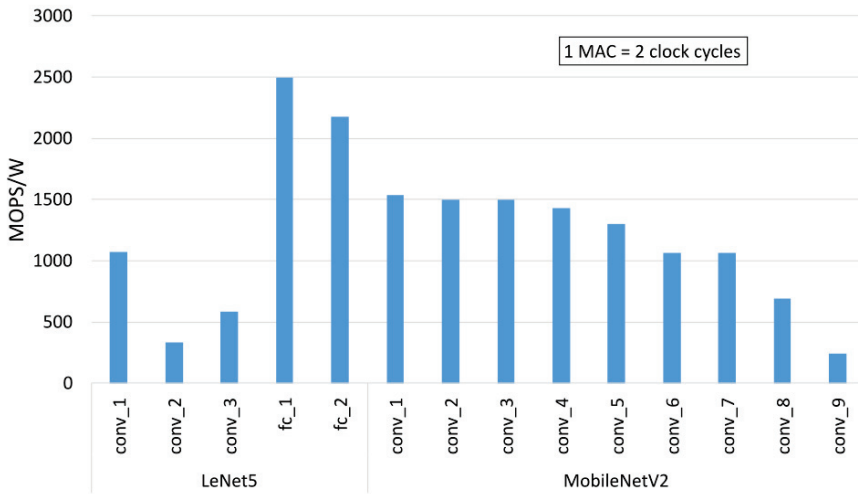

Figure 3.6 Energy efficiency (MOPS/W) of the proposed architecture

between ifmaps size and filter size (conv_2, LeNet5) or single row with few number of columns (conv_9, MobileNetV2).

3.7 Conclusion

This work presented a new flexible and scalable interconnect-based dataflow architecture, which can leverage data reuse and overlap between communication and computation to accelerate CNNs. We evaluated the proposed architecture results using LeNet5 and MobileNetV2 to show its adaptability to different types of DNNs. We then compared the latency results with state-of-the-art architectures. The proposed architecture is implemented (place and route) onto the Versal ACAP VCK190 kit featuring XCVC19022VSVA2197 FPGA partition. The experimental results show that the proposed architecture can speedup LeNet5 convolution layers by up to 71.2× in latency performance w.r.t. a RISC-V-based CPU and also speedup MobileNetV2 convolution layers by up to 2.07× in latency performance w.r.t. Eyeriss v2. We plan, in the future, to continue implementing optimisation techniques in the proposed architecture to better its energy efficiency and make the most of the underlying AINoC for accelerating complete Convolution Neural Networks execution.

Acknowledgements

This work was conducted within the scope of the European NEUROKIT2E project, funded by the European Union's Horizon Europe research and innovation program, under grant agreement number 101112268.

References

[1] A. Parashar et al., "SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks," in arXiv. 2017 https://doi.org/10.48550/ARXIV.1708.04485

[2] A. Pullini et al., "Mr. Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing," in IEEE JSSC, 2019, vol. 54, 7, pp. 1970–1981. https://doi.org/10.1109/JSSC.2019.2912307

[3] C. Farabet et al., "NeuFlow: A runtime reconfigurable dataflow processor for vision," in CVPR WORKSHOPS. IEEE, USA, 2011, pp. 109–116. https://doi.org/10.1109/CVPRW.2011.5981829

[4] D. Bhatt et al., "CNN Variants for Computer Vision: History, Architecture, Application, Challenges, and Future Scope," in Electronics: Ambient Assistive Methodologies/Frameworks for Internet of Medical Things, 2021, vol. 10, https://doi.org/10.3390/electronics10202470

[5] E. S. Shin et al., ń Round-robin Arbiter Design and Generation," in Proceedings of the 15th ISSS. Japan, 2002, pp. 243–248. https://doi.org/10.1145/581199.581253

[6] G. Sapijaszko et al., "An Overview of Recent Convolutional Neural Network Algorithms for Image Recognition," in 61st MWSCAS. 2018. Canada. https://doi.org/10.1109/MWSCAS.2018.8623911

[7] H. Kwon et al., "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects," in ACM SIGPLAN Notices. Vol. 53. 2018. pp. 461–475. https://doi.org/10.1145/3296957.3173176

[8] H. Krichene et al., "Analysis of on-chip communication properties in accelerator architectures for Deep Neural Networks," in 15th IEEE/ACM NOCS. USA, 2021. pp. 9–14. https://doi.org/10.1145/3479876.3481588

[9] H. Krichene et al., "AINoC: New Interconnect for Future Deep Neural Network Accelerators," in DASIP, 2023. pp 55–69.

[10] K. Sankaralingam et al., "Distributed Micro-architectural Protocols in the TRIPS Prototype Processor," in 39th MICRO'06. USA. 2006. https://doi.org/10.1109/MICRO.2006.19

[11] L. M. Ni et al. 1993. "A survey of wormhole routing techniques in direct networks," in IEEE Trans. Computer. 1993, Vol. 26, pp. 62–76. https://doi.org/10.1109/2.191995

[12] M. Sandler et al. 2018. "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in CVPR, 2018, USA, pp. 4510–4520. https://doi.org/10.1109/CVPR.2018.00474

[13] T. Chen et al., "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," in Proceedings of the 19^{th} ASPLOS, USA, 2014, pp. 269–284. https://doi.org/10.1145/2541940.2541967

[14] Y. Chen et al., "DaDianNao: A Machine-Learning Supercomputer," in 47^{th} Annual IEEE/ACM MICRO, UK, 2014, pp. 609–622. https://doi.org/10.1109/MICRO.2014.58

[15] Y. H. Chen et al., "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in ACM/IEEE 43^{rd} ISCA, Korea (South), 2016, pp. 367–379. https://doi.org/10.1109/ISCA.2016.40

[16] Y. H. Chen et al., "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," in IEEE JETCAS, 2019, vol. 9, pp. 292–308. https://doi.org/10.1109/JETCAS.2019.2910232

[17] Y. T. Chen et al., "Tile-Based Architecture Exploration for Convolutional Accelerators in Deep Neural Networks," in IEEE 3^{rd} AICAS, USA, 2021, pp. 1–4. https://doi.org/10.1109/AICAS51828.2021.9458540

[18] Y. Lecun et al., "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, 1998, vol. 86, pp. 2278–2324. https://doi.org/10.1109/5.726791

[19] Z. Du et al., "ShiDianNao: Shifting vision processing closer to the sensor," in ACM/IEEE 42nd ISCA, USA, 2015, pp. 92–104. https://doi.org/10.1145/2749469.2750389

[20] Xilinx. "User Guide UG1366" (v1.1). https://docs.xilinx.com/r/en-US/ug1366-vck190-eval-bd.