Networks-on-Chip based Deep Neural Networks Accelerators for IoT Edge Devices

Giuseppe Ascia*, Vincenzo Catania*, Salvatore Monteleone*, Maurizio Palesi*, Davide Patti*, John Jose†
*University of Catania, Italy
Email: firstname.lastname@dieei.unict.it
†Indian Institute of Technology, Guwahati, India
Email: †johnjose@iitg.ac.in

Abstract—**The need for performing deep neural network inferences on resource-constrained embedded devices (*e.g.*, Internet of Things nodes) requires specialized architectures to achieve the best trade-off among performance, energy, and cost. One of the most promising architectures in this context is based on massive parallel and specialized cores interconnected by means of a Network-on-Chip (NoC). In this paper, we extensively evaluate NoC-based deep neural network accelerators by exploring the design space spanned by several architectural parameters including, network size, routing algorithm, local memory size, link width, and number of memory interfaces. We show how latency is mainly dominated by the on-chip communication whereas energy consumption is mainly accounted by memory (both on-chip and off-chip). The outcome of the analysis, thus, pushes toward a research line devoted to the optimization of the on-chip communication fabric and the memory subsystem for performance improvement and energy efficiency, respectively.**

Index Terms—**Deep Neural Network accelerator, Network-on-Chip, Performance evaluation, Energy analysis, Design space exploration, IoT edge devices.**

I. Introduction

During the past few years, there has been a trend inversion from offloading as much as possible toward the cloud to moving back to the edge and terminal nodes. The main motivation of such inversion is related to the communication cost. If we consider a typical Internet of Things (IoT) scenario characterized by the three main tasks, namely, sensing, analyzing and classifying, and transmitting, we can observe as follows: sensing usually involves several kinds of sensors, including, MEMS Inertial Measurement Units (IMUs), MEMS microphones, ULP imagers, and EMGs/ECGs/EITs which work in the range of 100 μW to 2 mW; the analysis and classification task is usually implemented by means of microcontrollers working in the range of 1 to 10 mW at 1 to 25 MOPS; Finally, short range transmission at medium bandwidth or long rang transmission at low bandwidth require active power levels in the order of 50 mW. Based on this, the current trend is that of enriching the functionality and the duties of the local node to reduce the frequency and/or the amount data to be transmitted for being processed in the cloud.

Many current applications in the context of agriculture, health monitoring, surveillance, structural monitoring, *etc.*, are more and more often implemented by means of machine learning approaches based on the use of deep learning techniques. Deep neural networks (DNNs) represent one of the most widespread solutions applied in the aforementioned domains. Although the training phase of a DNN is the most time consuming and computational demanding task as compared to the inference phase, the latter is still over the computational capabilities provided by traditional microcontroller based devices. To tackle with this gap, several neural network accelerators have been proposed to work in-tandem with the microcontroller/microprocessor aimed at improving performance and power metrics, including, latency, throughput, and energy efficiency while executing DNN inferences [1].

As the complexity of DNNs increases, one can expect the emergency of scalable DNN accelerator platforms in which many DNN accelerators are connected into the same chip by means of an on-chip communication network [2]. This paper presents an experimental analysis aimed at identifying the main elements of a NoC-based DNN accelerator which mostly impact its performance and energy metrics. The analysis is focused on exploring the design space spanned by a set of architectural elements, including, network size, routing algorithm, number of memory interface, local memory size, and links size. Two of the most common neural networks architectures, namely, LeNet-5 [3] and VGG-16 [4] are considered in the experiments as they are representative of two different level of complexity (approx 60 K parameters for LeNet and 138 M parameters for VGG-16). The outcome of the analysis is that the on-chip communication account for a relevant faction of the total inference latency whereas energy consumption is dominated by the memory sub-system (both on-chip and off-chip).

The rest of the paper is organized as follows. The next section reviews works related to the design and implementation of accelerators for deep neural network inferences. Sec. III discusses how a convolutional neural network is mapped on a NoC-based architecture. The experimental analysis is discussed in Sec. IV. Finally, Sec. V concludes the paper.

II. Related Work

In the last few years, many works have been conducted for accelerating DNNs by means of GPGPU [4], FPGAs [5], and ASIC implementations [6]. The need for scalability has suggested the use of the NoC paradigm as backbone for interconnecting specialized DNN accelerator cores. An analytical evaluation and comparison of different configurable interconnect architectures for NNs is presented in [7]. The authors found that multicast mesh NoC provides the highest performance/cost ratio and consequently it is the most suitable

This work was supported in part by the Piano per la Ricerca 2016/2018 DIEEI Universitá degli Studi di Catania.

interconnect architecture for a configurable neural network implementation. A large-scale DNN accelerator which uses a mesh based interconnection network for data transfer is used in DianNao accelerator [8]. A hierarchical NoC architecture for spiking NN is proposed in [9] to overcome the lack of modularity and poor connectivity shown by traditional neuron interconnect implementations based on shared bus topologies. At the extreme spectrum of communication fabric, IBM Truenorth [10] employs a 256 $\times$ 256 crossbar into the neurocore and a mesh NoC for inter-core communication. Communication latency and throughput have a strong impact on the overall performance of the NN. Scatters, gathers, and local communications which characterize the traffic generated in the different layers of a DNN have been considered for generating a network tailored to optimize such communication patterns [11]. The proposed architecture outperform conventional NoC architectures in terms communication latency, throughput, energy and cost. Traditional NoC architecture is considered in [2] to implement NoC based DNN accelerators in which the communication traffic is reduced by adopting a new sparsified parallelization technique that exploits the noise-tolerance feature of deep learning algorithms to enable the NN to learn a configuration that is very suitable to be parallelized on a NoC.

In the above works, the aspects related to the NoC as communication fabric for supporting the data movement among the NN processing elements is often only marginally investigated. In this work, we explore several architectural parameters of the NoC, including, network size, number of memory interfaces, routing algorithm, local memory size, to assess their impact on NN inference latency end energy consumption of the NoC based NN accelerator.

III. NoC-based Deep Neural Network

A. *Reference Architecture*

We consider as reference NoC-based DNN accelerator architecture a mesh-based NoC in which a node can be either a memory interface (MI) or a processing element (PE). The PE is specialized in performing the computational kernels used by the three types of layers which form the DNN, namely, convolutional layer, max/avg layer, and fully connected layer. The next sub-sections will discuss how each type of layer is mapped into the NoC focusing on identifying the data flows both intra-NoC (among the PEs) and off-NoC (among the NoC-based DNN accelerator and the main memory).

B. *Convolutional Layer*

A convolutional layer takes in input the input feature map and a set of filter. Each filter is convolved with the input feature map to generate a channel of the output feature map. Fig. 1 shows the traffic generated for processing a convolutional layer with six filters. The example shows a 4×4 mesh with fifteen PEs and one MI. The top part of the figure shows the traffic generated to load the input feature map from the main memory. The MI sends the input feature map to the PEs involved in this layer. We consider a $1 : n$ mapping between PEs and filters, that is, a PE can process multiple filters. In the example, each

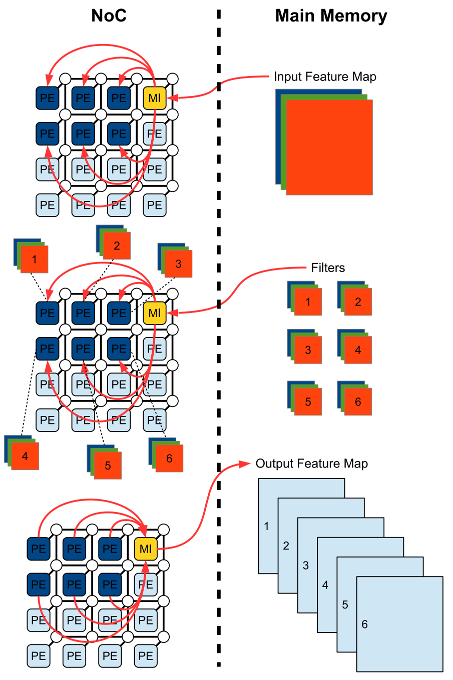

Fig. 1. Traffic generated for processing a convolutional layer. Loading of the input feature map (top), loading of the filters (middle), storing the output feature map (bottom).

filter is sent to a specific PE (middle part of the figure). Finally, each PE has all the ingredients to compute a channel of the output feature map that is store back to the main memory (bottom part of the figure). The output feature map, will be the input feature map for the next layer of the NN.

It should be pointed out that, the last phase shown in Fig. 1 can be skipped. In fact, each of the PEs active at a generic layer has in its local memory a number (one in the example) of channels of the output feature map. Thus, the PE involved in the subsequent layer, can obtain the input feature map from the PEs that computed the previous layer. That is, the input feature map for layer i is spread over the PEs used for processing the layer $i-1$. Based on this, with exception of the first layer, the access to the main memory for loading the input feature map can be avoided. This case is shown in Fig. 2 for layer i and layer $i+1$. The output feature map of layer i is spread over the six PEs, *i.e.*, each channel of the output feature map is stored into the local memory of a PE. In layer $i+1$ there are eight filters and each of them has to be applied to the input feature map whose six channels are stored into the local memory of the six PEs (one channel per PE) shown in blue in the figure. In order to apply the filter, each the eight PEs needs the complete input feature map (all the six channels). Thus each of the six PEs (represented in blue in the figure) sends its feature map channel to the other seven PEs. Doing so, all the eight PEs can apply their respective filter.

Please note that, the underlying assumption behind the above discussion is that, the local memory into the PEs is enough large to store at least one channel of the feature map. If this hypothesis is not met, the output feature map is stored

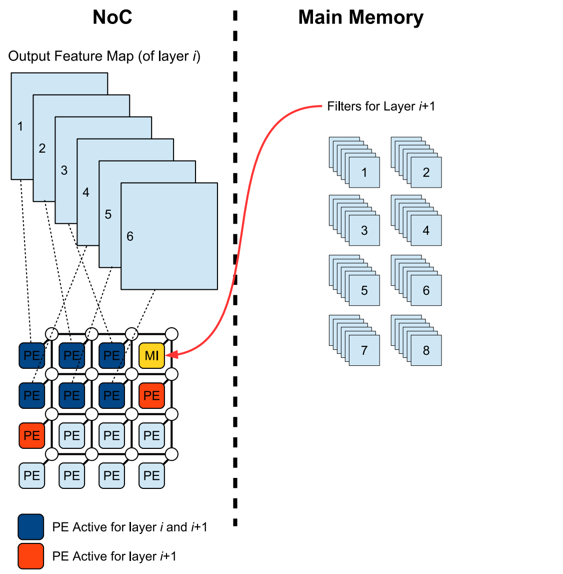

Fig. 2. The output feature map generated in layer i is spread over the blue PE. Each blue PE sends its feature map channel to the rest of the PEs active in the forthcoming layer $i + 1$.

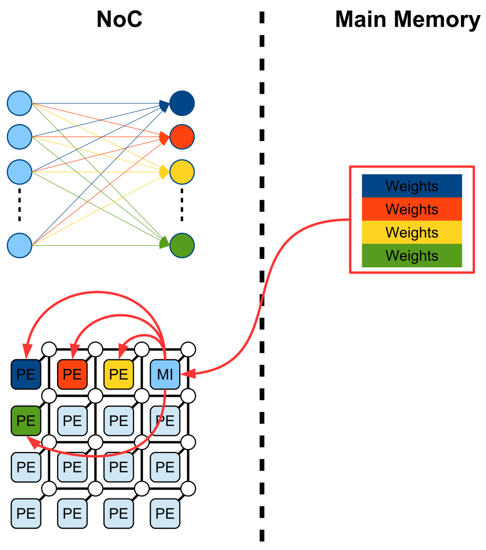

Fig. 3. A fully connected layer in which each neuron is mapped to a PE.

TABLE I
DESIGN SPACE CONSIDERED IN THE EXPERIMENTS.

Parameter	Values
Network size	3×3, 4×4, 5×5, ..., 10×10
Routing algorithm	XY, Odd-Even
Clock frequency	1 GHz
Local memory per core	1, 2, ..., 256 KB
No. of memory interfaces	1, 2, ..., 8
Link width	32, 64, 128, 256, 512 bit

back to the main memory.

From the computational point of view, the basic operations performed in this layer are Multiply And Accumulate operations (MAC operations) used to perform dot-products of the convolution among the filter mapped in the PE and data of the feature map. The amount of MAC operations depends on the filter size, the size of the input feature map and the stride used in the considered convolutional layer.

C. Pooling Layer

For a pooling layer, both average pool and max pool, there is a $1 : n$ mapping between PEs and feature map channels, that is, a PE can process multiple feature map channels. In this case, there is no PE to PE traffic as each PE works on the input feature map channel currently stored in its local memory. If the local memory is not large enough to store the entire feature map channel, the latter is fetched from the main memory resulting in memory to PEs traffic. From the computational point of view, the operations performed in this layer are either average or max operations among data in the input feature map. The amount of such operations depends on the size of the input feature map, the scale and stride parameters of the layer.

D. Fully Connected Layer

In a fully connected layer, the output neurons have a number of inputs equal to the size of the input feature map. We consider a $1 : n$ mapping between PEs and neurons, that is, a PE can process multiple neurons. Thus, each PE needs to fetch from the main memory a number of weights corresponding to the size of the input feature map as shown in Fig. 3.

From the computational point of view, the operations involved in this layer are mainly MAC operations used perform dot-products among data of the input feature map and the weights for each of the neuron mapped on the PE. The amount of MAC operations depends on the number of neurons of the layer and the size of the input feature map.

IV. EXPERIMENTAL ANALYSIS

A. Experimental Setup

The experimental platform is a simulated parameterized NoC-based Deep Neural Network that allows to assess different architectural configurations in terms of performance and energy. For the computational part, we have developed a simplified version of DianNao [8] which has been integrated into the processing element module of Noxim [12] cycle-accurate NoC simulator. The RTL models of the PE and router have been synthesized with Synopsis Design Compiler and mapped on a 45 nm CMOS LVT library from Nangate [13]. The links have been modelled with HSPICE and the parasitics extraction from layout has been made using Cadence Virtuoso. The power figures collected by the circuit level analysis have been used to back-annotate the simulator. The considered design space in the experiments is reported in Tab. I. For memory, both local and main memory, we used CACTI [14] to estimate the energy consumption (both leakage and dynamic) and timing information.

B. Experiments Overview

The analysis is organized as a per-layer analysis and a global analysis. The first one is aimed at analyzing how each specific layer affects the considered metrics, whereas the second one is focused on the overall behaviour of the network still as respect to the considered metrics. The considered metrics refer to the fraction of time and energy spent in communication, computation, and memory access, the memory traffic, and the PEs utilization. The analysis is carried out on a set of architectural configurations belonging to the design space reported in Tab. I.

With regard to the considered neural networks, we selected two convolutional neural networks, namely, LeNet-5 [3] and VGG-16 [4]. LeNet-5 counts approx 60 K parameters and 6 layers, whereas VGG-16 count approx 138 M parameters and 20 layers. Their selection has been guided by the fact that can be considered as representative in the complexity spectrum.

C. Per-layer Analysis

Fig. 4 shows the fraction of time spent in each layer for LeNet-5 and VGG-16. The platform is configured as a 8×8 mesh in which four MIs are located into the four corners of the NoC, links are 256 bits wide and PE local memory is 32 KB. The fraction of time is broken into its three components, namely, communication, computation, and memory. As it can be observed, the communication dominates the latency. This true for every layer for LeNet-5 whereas, for VGG-16, the communication latency is low in the inner convolutional layers. This is due to the fact that, the feature size after the second max-pool layer drastically decreases. Thus, it results in a reduction of the PE to PE traffic for reconstructing the input feature map starting from the partial output feature maps that are spread over the active PEs in the previous layer. The fully-connected (FC) layer in LeNet-5 dominates the latency as the amount of weights to be fetched from the memory and dispatched to the PEs is much larger than the amount of traffic generated by the fetch of the filters and the feature maps in the other layers. In VGG-16, although FC layers account for a significant fraction of the latency, this is dominated by the second and third layer where the size of the feature maps is much larger than that of filters. Thus, the traffic generated to deliver the input feature maps to the PEs and that to store the output feature maps to the main memory (because it does not fit the local memory), result in a major impact on the latency.

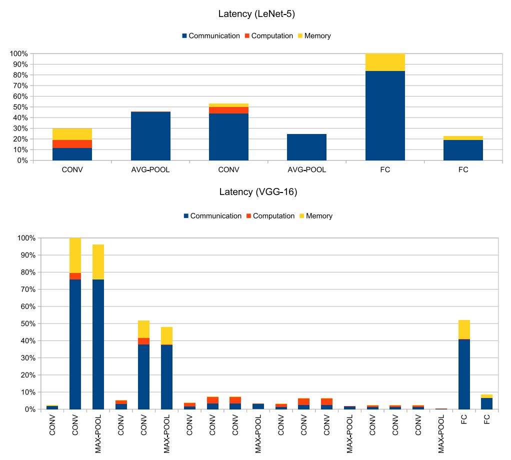

Fig. 4. Fraction of time spent in each layer.

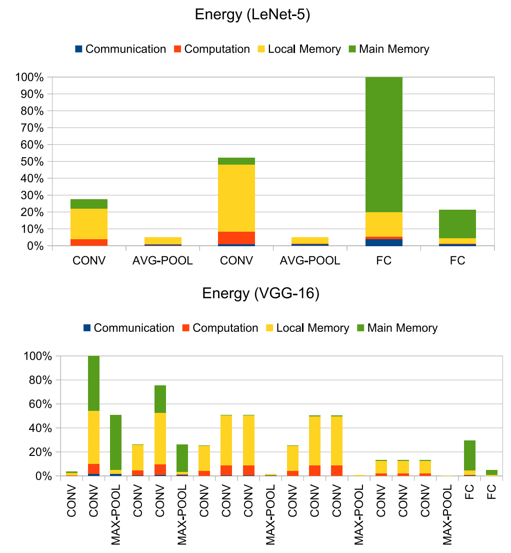

Fig. 5. Fraction of energy consumed in each layer.

Fig. 5 shows the fraction of energy consumed in each layer for LeNet-5 and VGG-16. Here the energy consumption is dominated by the memory, both local memory and main memory. For LeNet-5 the main memory contribution is mainly located to the FC layer due to the great amount of weights to be fetched as respect to the filters used in CONV layers. Further, the small size of the feature maps can be stored into the local memory reducing the traffic to the main memory. This explains how the second energy contribution is due to the local memory. In VGG-16, energy spent in local memory dominates in CONV layers. The main memory energy contribution is localized in the first layers in which the feature size is too big to fit the local memory size. After the second MAX-POOL layer, the amount of memory to store the feature map drastically reduces limiting the amount for traffic to and from the main memory.

Fig. 6 shows the normalized memory traffic in each layer for LeNet-5 and VGG-16. As expected, FC layers are the main responsible for the traffic with the main memory.

Fig. 7 shows the number of active PEs and the normalized load of PEs per layer. For LeNet-5 all the available 60 PEs are used only in FC layers whereas VGG-16 utilizes all the available PEs in each layer. As expected, the highest PE utilization is in correspondence of CONV layers and it is low in FC layers.

D. Impact of Local Memory Size

The local memory into the PEs has a strong impact on both latency and energy metrics. Fig. 8 shows the normalized latency per inference for different local memory sizes for LeNet-5 and VGG-16. We use the same platform configuration as for

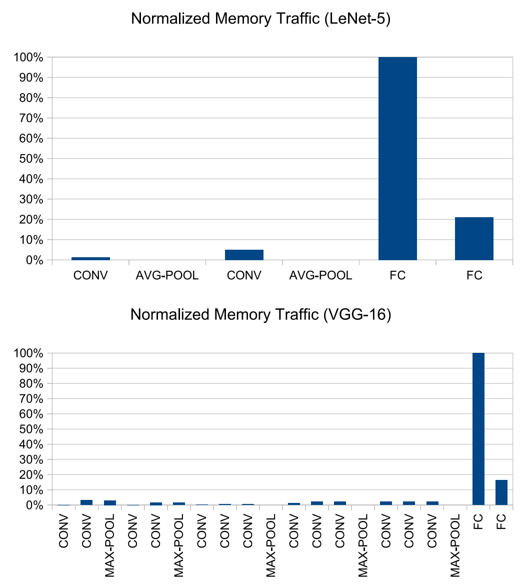

Fig. 6. Normalized memory traffic per layer.

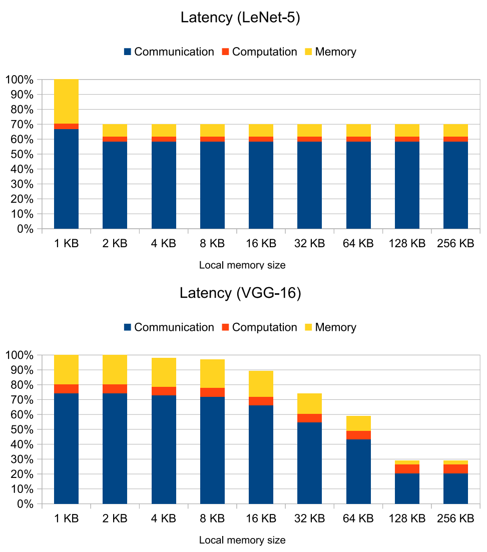

Fig. 8. Normalized latency per inference for different local memory size.

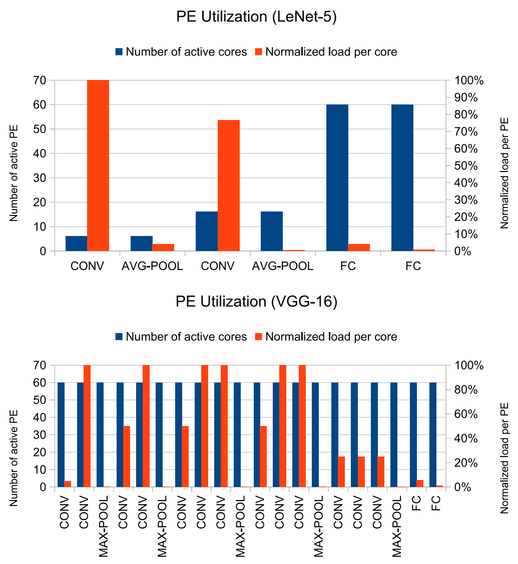

Fig. 7. PE utilization: number of active PEs and normalized load of PEs per layer.

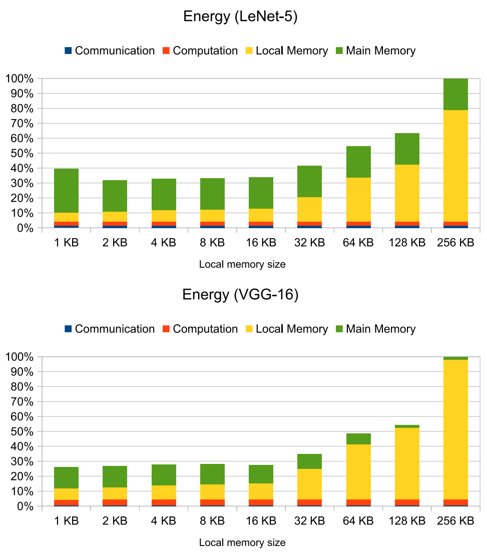

Fig. 9. Normalized energy per inference for different local memory size.

the per-layer analysis above in which the local memory size is made to vary from 1 KB to 256 KB. As it can be observed, the low complexity of LeNet-5 does not show any latency improvement for local memory size larger than 2 KB. On the contrary, the local memory requirement demanded by VGG-16 is larger than 16 KB to obtain important latency improvements but the latter becomes not relevant above 128 KB.

The energy trends are shown in Fig. 9. As the local memory size increases, the total energy consumption decreases till an inversion point after that it start to increase. The inversion point is at 2 KB for LeNet-5 and 16 KB for VGG-16. It is due to the fact that, although as local memory size increases the main memory accesses decrease, the energy per access to the local memory increases. Thus there is a optimal local memory size beyond which the main memory energy saving is less than the local memory per access energy.

E. Impact of NoC Size

As the NoC size increases, the number of available PEs allows to parallelize the computation in each layer. However, the average communication distance increases with a consequent negative impact on the communication component. Fig. 10 shows the normalized latency per inference for different NoC size. We use the same platform configuration as for the per-layer analysis above in which the NoC is made to vary from

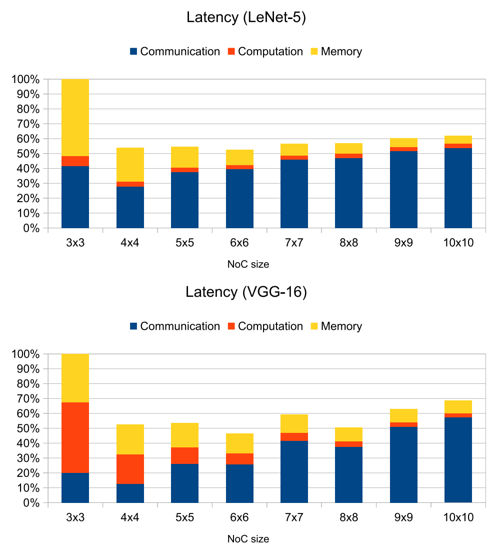

Fig. 10. Normalized latency per inference for different NoC size.

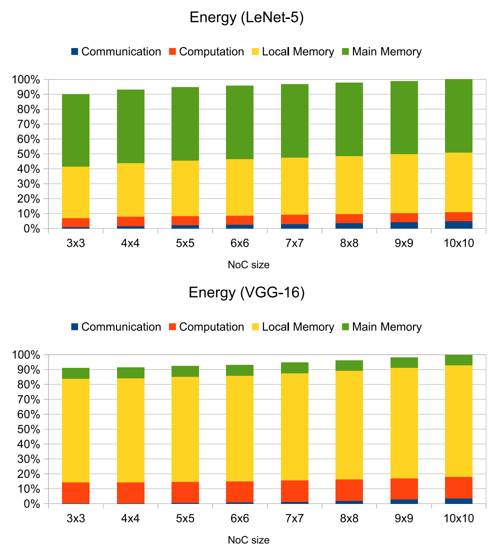

Fig. 11. Normalized latency per inference for different NoC size.

3×3 to 10×10. For LeNet-5 it has been shown that, with exception of FC layers, at most 16 PEs are used (see Fig. 7). For this reason, starting from 4×4 NoC no appreciable latency variation is observed. In fact, starting from 4×4 NoC, there is not any appreciable reduction of computation latency while the communication latency increases. For VGG-16, the minimum communication latency is observed for 4×4 NoC and the overall minimum latency is for 6×6 NoC. Starting from such NoC size, the decrease of the computation latency due to the higher number of available PEs is less than the increase of communication latency.

Fig. 11 shows the trend of energy as NoC size increases. For LeNet-5, the energy slightly increases due to the increase of the average communication distance which increases the communication component of the energy. In VGG-16, as NoC size increases, the greater availability of local memory slightly reduces the traffic to main memory but not enough to save energy as compared to the increase of the communication and local memory components. For LeNet-5 the local memory and main memory energy components are almost the same whereas for VGG-16 the local memory energy component dominates. This behaviour is due to the fact that, as we have observed in the per-layer analysis, in LeNet-5 the energy consumption is dominated by main memory during the FC layer whereas the energy component of local memory if moderate but present in all the remaining layers. Thus, the overall energy consumption of the two components roughly equalize. Conversely, in VGG-16, considering again the per-layer analysis, it can be observed that the local memory energy components is relevant in all the CONV layers and due the higher number of layers than LeNet-5, the energy contribution of the local memory overall dominates.

F. Impact of Routing Algorithm

There are primarily three kinds of traffic flows generated when a DNN is mapped into a NoC, namely, scatter, gather, and local. Scatter flow is the traffic distribution from memory to PEs to deliver input feature maps and weights. Based on the considered parallelization scheme, feature map distribution results in broadcast communication flows whereas weights distribution results in unicast communication flows. Gather flow is the traffic distribution from PEs to memory for storing back the output feature map. Thus, based on what we found in the previous subsection, the amount of local memory in each PE determines the gather traffic volume. The gather traffic is an hot-spot traffic as destinations are MIs which are few as compared to the number of PEs. Finally, local flow is the traffic distribution among PEs (broadcast traffic) due to dispatch the computed output feature map channels among PEs. Even in this case, the amount of local traffic depends on the local memory size. In fact, if the channels of the output feature map computed by a PE do not fit the local memory size, they are stored into the main memory and then distributed to the PEs through the scatter flow. It is known that the routing algorithm has a relevant impact on the communication latency. Further, the difference among routing algorithms are highlighted as network size and communication patterns change. To assess the impact of the routing algorithm in our analysis, in Fig. 12 we show the percentage latency reduction when we pass from a deterministic XY routing to an adaptive Odd-Even [15] routing. We consider different NoC sizes and different local memory sizes as the latter determines the distribution of the three traffic flows described above. In LeNet-5, as NoC size increases and exceeds 5×5 size and for local memory size greater than 2 KB, the use of Odd-Even starts to improve the latency. In VGG-16, the impact of routing algorithm is more evident (up to 30% of latency reduction) and, as in LeNet-5, starts for 5×5 NoC size. It is interesting to observe that,

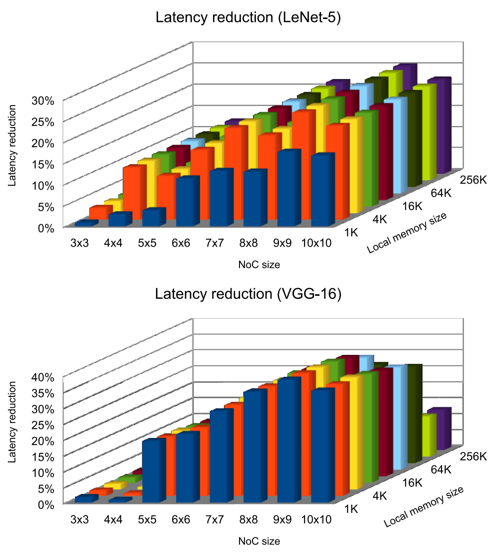

Fig. 12. Latency reduction when passing from XY to Odd-Even routing algorithm for different NoC sizes and different local memory sizes.

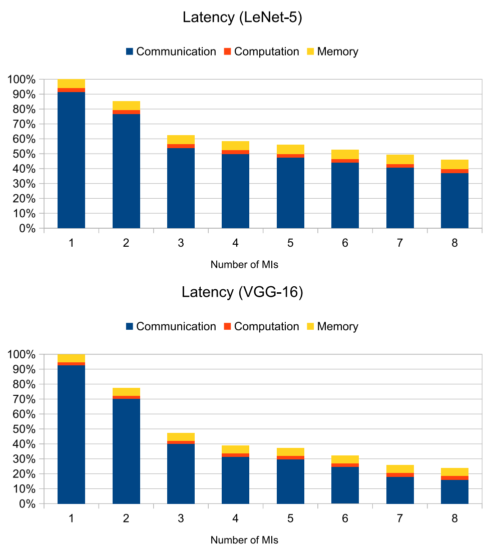

Fig. 13. Normalized latency per inference for different number of MIs.

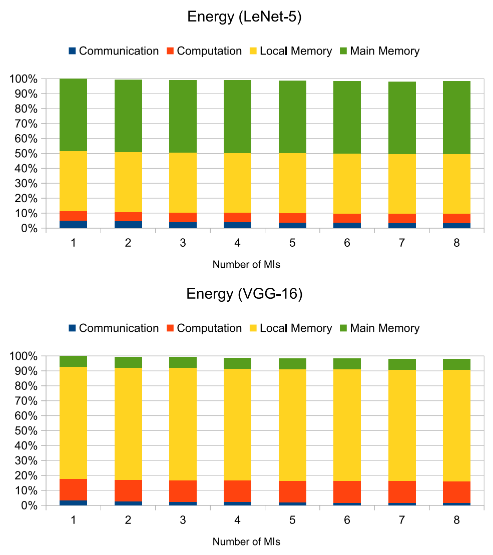

Fig. 14. Normalized energy per inference for different number of MIs.

after 64 KB of local memory the improvement due to the use of Odd-Even decay. This is due to the fact that, the amount of gather flow decreases thus eliminating the hot-spot traffic to the MIs. In fact, it is known that Odd-Even is effective in attenuating the impact of hot-spot traffic thus, as the hot-spot component of the traffic reduces, the difference among XY and Odd-Even attenuates.

G. Impact of Memory Interfaces

As the number of memory interfaces increases the average distance between PEs and MIs decreases. Thus, the communication component of the latency for memory accesses decreases. This trend is shown in Fig. 13 in which the number of MIs is made vary from 1 to 8 and the platform is configured as for the per-layer analysis. As it can be observed, the latency reduction passing from 1 MI to 8 MIs is almost 50% and 80% for LeNet5 and VGG-16, respectively. As LeNet-5 uses at most 16 PEs in all the layers with exception of the FC layers, a slowdown in communication latency reduction after 3 MIs is observed. For VGG-16, in which all the available PEs are used in any layer, the communication latency reduction is relevant up to 8 MIs. Further, even if no visible from the figure (as the latency is dominated by the communication component), there is a increase of computation latency due to the fact that, since the NoC size is fixed, as the number of MIs increases, the number of PEs decreases accordingly.

As the number of MIs increases, the energy consumption decreases as shown in Fig. 14. For LeNet-5 the energy consumption is equally dominated by local memory and main memory whereas local memory dominates in the case of VGG-16. This is due to the fact that, the number of active PEs for VGG-16 is higher than that for LeNet-5, thus local memory is much more utilized in VGG-16. For the same motivation the energy contribution of the computing components is higher in the case of VGG-16 than LeNet-5.

H. Impact of Link Width

As the link width of the NoC increases, the communication bandwidth increases with a consequent improvement of communication latency. Fig. 15 shows the normalized latency per inference when the platform is configured like in the case of the per-layer analysis and in which the link width is made to vary from 32 bits to 512 bits. Since the communication component is the major contributors to the global latency, as expected, as link width increases, the global latency decreases almost proportionally.

For what it concerns energy, Fig. 16 shows that there is no appreciable variation in energy consumption as the link width increases. In fact, as link width increases, the energy consumption of the router increases but the energy per port per

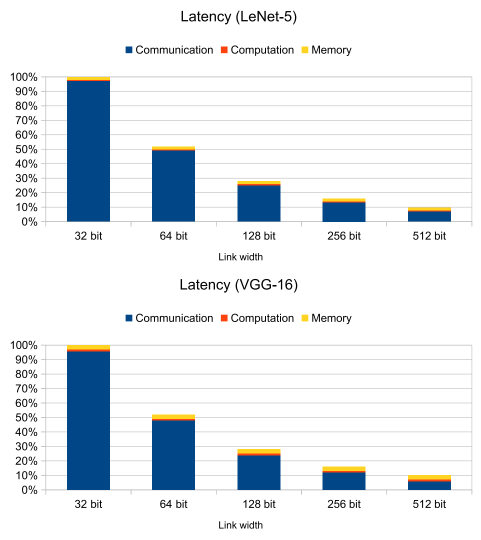

Fig. 15. Normalized latency per inference for different link width.

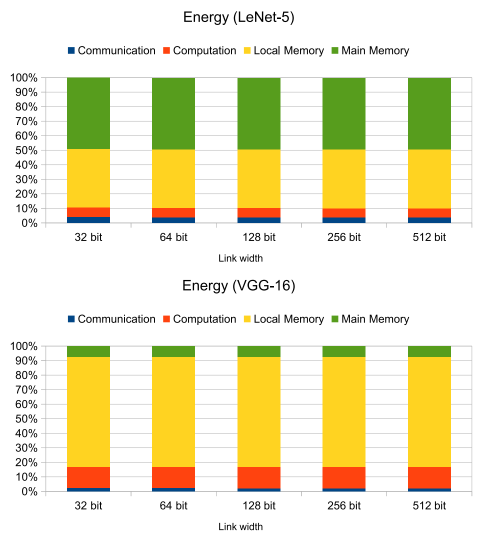

Fig. 16. Normalized energy per inference for different link width.

bit decreases. Thus, the communication energy decreases but its low contribution as respect to the other energy contributions is marginal.

V. Conclusion

In this paper, we have evaluated NoC-based deep neural network accelerators in terms of inference latency and energy consumption for two convolutional neural networks, namely, LeNet-5 and VGG-16, when several architectural parameters, including, NoC size, routing algorithm, number of memory interfaces, link width, and local memory size, are made to vary. Overall, we found that, the NoC is the main responsible for inference latency whereas memory (both local and main memory) is the main contributor for energy consumption. Although the analysis has been carried out at a relative high level of abstraction, the following conclusions can be drawn. To improve the performance in terms of inference latency, it is essential to focus on the communication sub-system. To improve the energy efficiency of the system, particular effort should be devoted on the memory sub-system by reducing the memory traffic and/or making memory access more energy efficient.

References

[1] F. Conti, P. D. Schiavone, and L. Benini, “Xnor neural engine: A hardware accelerator ip for 21.6-fj/op binary neural network inference,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2940–2951, Nov 2018.

[2] K. Zou, Y. Wang, H. Li, and X. Li, “Learn-to-scale: Parallelizing deep learning inference on chip multiprocessor architecture,” in *IEEE/ACM Proceedings of Design, Automation and Test in Europe conference (DATE)*. -: IEEE, March 2019, pp. 1172–1177.

[3] 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, Nov 1998.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, Jun 2017.

[5] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, “Deepburning: Automatic generation of fpga-based learning accelerators for the neural network family,” in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC ’16. New York, NY, USA: ACM, 2016, pp. 110:1–110:6. [Online]. Available: http://doi.acm.org/10.1145/2897937.2898003

[6] L. Cavigelli and L. Benini, “Origami: A 803-gop/s/w convolutional network accelerator,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 11, pp. 2461–2475, Nov 2017.

[7] D. Vainbrand and R. Ginosar, “Network-on-chip architectures for neural networks,” in *ACM/IEEE International Symposium on Networks-on-Chip*. -: IEEE, May 2010, pp. 135–144.

[8] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, “Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” in *International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: ACM, 2014, pp. 269–284.

[9] S. Carrillo, J. Harkin, L. J. McDaid, F. Morgan, S. Pande, S. Cawley, and B. McGinley, “Scalable hierarchical network-on-chip architecture for spiking neural network hardware implementations,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 12, pp. 2451–2461, Dec 2013.

[10] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. N. ans Pallab Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, “Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, Oct 2015.

[11] H. Kwon, A. Samajdar, and T. Krishna, “Rethinking nocs for spatial neural network accelerators,” in *IEEE/ACM International Symposium on Networks-on-Chip*. New York, NY, USA: ACM, 2017, pp. 19:1–19:8.

[12] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, “Cycle-accurate network on chip simulation with noxim,” *ACM Transactions on Modeling and Computer Simulation*, vol. 27, no. 1, pp. 4:1–4:25, Nov. 2016.

[13] I. NanGate, “NanGate 45nm open cell library,” 2008. [Online]. Available: http://www.nangate.com

[14] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, “Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0,” in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Washington, DC, USA: IEEE Computer Society, Dec 2007, pp. 3–14.

[15] G.-M. Chiu, “The odd-even turn model for adaptive routing,” *IEEE Transactions on Parallel Distributed Systems*, vol. 11, no. 7, pp. 729–738, 2000.