NoC-based DNN Accelerator: A Future Design Paradigm

Special Session Paper

Kun-Chih (Jimmy) Chen
National Sun Yat-sen University,
Kaohsiung, Taiwan
kcchen@mail.cse.nsysu.edu.tw

Masoumeh Ebrahimi
KTH Royal Institute of Technology,
Stockholm, Sweden
mebr@kth.se

Ting-Yi Wang
Yuch-Chi Yang
National Sun Yat-sen
University,Kaohsiung, Taiwan

ABSTRACT

Deep Neural Networks (DNN) have shown significant advantages in many domains such as pattern recognition, prediction, and control optimization. The edge computing demand in the Internet-of-Things era has motivated many kinds of computing platforms to accelerate the DNN operations. The most common platforms are CPU, GPU, ASIC, and FPGA. However, these platforms suffer from low performance (*i.e.*, CPU and GPU), large power consumption (*i.e.*, CPU, GPU, ASIC, and FPGA), or low computational flexibility at runtime (*i.e.*, FPGA and ASIC). In this paper, we suggest the NoC-based DNN platform as a new accelerator design paradigm. The NoC-based designs can reduce the off-chip memory accesses through a flexible interconnect that facilitates data exchange between processing elements on the chip. We first comprehensively investigate conventional platforms and methodologies used in DNN computing. Then we study and analyze different design parameters to implement the NoC-based DNN accelerator. The presented accelerator is based on mesh topology, neuron clustering, random mapping, and XY-routing. The experimental results on LeNet, MobileNet, and VGG-16 models show the benefits of the NoC-based DNN accelerator in reducing off-chip memory accesses and improving runtime computational flexibility.

CCS CONCEPTS

• **Computing methodologies** → *Artificial intelligence*; • **Computer systems organization** → *Architectures*;

KEYWORDS

Network-on-Chip (NoC), Deep Neural Network (DNN), CNN, RNN, Accelerators, Routing Algorithms, Mapping Algorithms, Neural Network Simulator

ACM Reference format:
Kun-Chih (Jimmy) Chen, Masoumeh Ebrahimi, Ting-Yi Wang, and Yuch-Chi Yang. 2019. NoC-based DNN Accelerator: A Future Design Paradigm. In *Proceedings of International Symposium on Networks-on-Chip, New York, NY, USA, October 17–18, 2019 (NOCS '19),* 8 pages.
https://doi.org/10.1145/3313231.3352376

1 INTRODUCTION

Deep Neural Networks (DNNs), as a subset of Neural Networks (NNs), have shown enormous advantages in various domains. The number of applications benefiting from DNNs is increasing with the most popular ones in the area of pattern recognition, image classification, and computer vision [1][2]. The main advantage of DNN is the offered high accuracies that come at the cost of high computational complexity and power consumption. The recent DNN models, such as AlexNet [3] and VGG-16 [4], consist of hundreds of layers and millions of parameters that are too complex to be efficiently run on the existing hardware platforms. There are several algorithmic and architectural solutions to reduce the number of parameters and computational complexity. Pruning [5] and bit-width reduction [6] are in this category.

Another solution is to exploit parallelism through specialized DNN accelerators and ASIC design. In spatial architectures, the accelerators are composed of DNN processing units, arranged in a 2D array to flow the data from one PE to another [7]. To be integrated with other algorithmic and architectural solutions, these ASIC-based architectures demand new processor designs and connectivity patterns. This trend has led to several heuristic solutions and specific architectures. However, these architectures lack flexibility, and this trend impose huge design cost. In addition, these accelerators are optimized for a specific DNN application such as image classification while showing a poor performance under differing applications such as speech recognition. Another major problem is the long data communication latency mainly due to frequent read/write accessed from/to off-chip memory.

FPGAs are an alternative for the fast prototype of DNN accelerators due to their re-programmable attribute [8]. Compared with the ASIC-based DNN accelerators, FPGAs provide a flexible design space for various DNN accelerator designs. However, FPGA-based designs still suffer from low computational flexibility as they are configured for a specific DNN model and a particular application.

To enhance the design flexibility of the current DNN accelerators and reduce the interconnection complexity, a modular structure similar to Networks-on-Chip (NoC) can be exploited. NoC is a packet-switched network which enables a large number of PEs to communicate with each other. NoC consists of routers and links, where each router is connected to a PE (or a group of PEs), and links connect the routers to each other. Topology determines the overall arrangement of routers and links which can be in the form of mesh and torus. In NoCs, resources are shared and distributed among PEs, *e.g.*, memory is distributed among PEs, and packets may utilize distributed network resources in their paths toward the destination. The superiority of NoC has inspired researchers to utilize it in the modeling of large-scale spiking neural networks (such as SpiNNaker

NOCS '19, October 17–18, 2019, New York, NY, USA

ACM ISBN 978-1-4503-6700-4/19/10…$15.00
https://doi.org/10.1145/3313231.3352376

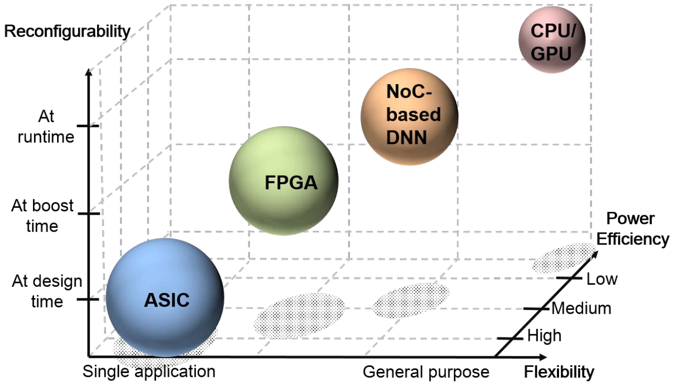

Figure 1: Flexibility and reconfigurablity of current DNN accelerator design paradigms.

[9][10] and CuPAN [11]) and a few DNN accelerators (such as Eyeriss-v2 [12] and Neu-NoC [13]).

Fig. 1 summarizes the aforementioned DNN accelerator design paradigms. CPUs and GPUs offer a very high reconfigurability feature at runtime, which enable them to support diverse DNN applications. However, these platforms suffer from high power consumption and data transfer latency between PEs and off-chip memory. In contrast, the ASIC-based DNN accelerators are specifically designed for a particular DNN model in order to achieve optimal performance and power efficiency. In return, these designs limit the computational flexibility; and as a result, the ASIC-based DNN accelerators are not reconfigurable at design time. As shown in Fig. 1, although the FPGA-based designs improve the design flexibility, the computational flexibility is not still sufficient to support reconfigurability at runtime. For example, the data path is fixed for a particular RNN or DNN model at design time, and no further reconfiguration can be done at runtime.

Among different DNN accelerator design methodologies, a NoC-based DNN accelerator could be an appropriate choice, which offers power efficiency, computational flexibility, and reconfigurability. The reason is that the NoC-based design methodology decouples the DNN operation into computation and data transmission. Regarding the data transmission part, the NoC interconnection can efficiently process various data flows for different DNN models. Regarding the computation part, different DNN computing models can be executed independent of the data flow. Furthermore, flexible communication reduces the frequent accesses to the off-chip memory by handling the data transfer from one core to another on the chip. Reducing memory accesses leads to significant power saving. The main contributions of this paper are summarized as follows:

- We investigate the state-of-the-art DNN accelerator design methodologies and analyze the pros and cons of each.
- We suggest Network-on-Chip as a new design paradigm to implement future flexible DNN accelerators.
- We analyze the number of memory accesses in the conventional and NoC-based designs under different DNN models .
- We analyze the performance of the NoC-based DNN accelerator under different design parameters.

2 CONVENTIONAL DNN COMPUTING PLATFORMS

In recent years, deep neural networks (DNNs) have become the foundation for many modern artificial intelligent (AI) applications. While DNN delivers stunning accuracy in many AI applications, it comes at the cost of intensive communication and extensive computation. In this section, we review the conventional hardware platforms to execute and accelerate DNN operations.

2.1 Common Hardware Platforms for Neural Network Computing

Central Processing Units (CPUs) typically consist of multi-core processors (usually from 2 to 10 cores), employed on desktop and mobile devices. Advanced CPUs, such as 48-core Qualcomm Centriq 2400 [14] and 72-core Intel Xeon Phi [15], are composed of more cores to improve the computation efficiency. CPUs are basically designed to compute general-purpose computations, so they have the main advantage of high computational flexibility, and they are capable of executing complex operations. Nevertheless, they are not suitable for DNN computing, which involves intensive parallel while simple computations (such as multiply and accumulate in convolution). While it has been some attempts to exploit CPU clusters (such as Intel BigDL [16]) to optimize deep learning libraries [17], they cannot still satisfy the efficiency demands in DNN computing.

Nowadays, Graphics Processing Units (GPUs) are popular devices to compute DNN operations. Originally, GPUs were designed for computer graphic tasks. Due to the intrinsic parallel computing features in GPUs, they can efficiently execute the essential DNN operations in parallel, such as matrix multiplication and convolution. Also, thanks to the powerful parallel computing platforms (such as Tesla V100 by NVIDIA) and application programming interfaces (such as Compute Unified Device Architecture (CUDA)), GPUs are being extensively used for DNN acceleration in many applications [18]. Although GPUs provide high parallel computing capability to speed up the DNN computing, there is a general concern about the increasing power consumption [19]. Furthermore, GPUs practically require additional communication channels (such as PCIe or NVLink) to connect with the sampling devices. Therefore, the data transfer latency is the other issue in this kind of computing platforms [8].

2.2 Neural Network Accelerators

Because of the intrinsic characteristic of parallel computation in DNN operations, it is intuitive to exploit parallelized multicore hardware to accelerate the operations. However, due to the bandwidth limitation between off-chip memory and processing cores, it is not power and performance efficient to frequently transfer highly parallelized data into the parallelized multicore hardware. To achieve a better trade-off between performance and power consumption in DNN computing, Application Specific Integrated Circuits (ASIC) are used to realize the DNN hardware. In ASIC-based designs, to reach high performance and power efficiency, on-chip computing units are optimized for a particular application [20][21][22][23]. As a result, ASIC-based accelerators achieve superior performance and power advantages over their CPU and GPU counterparts.

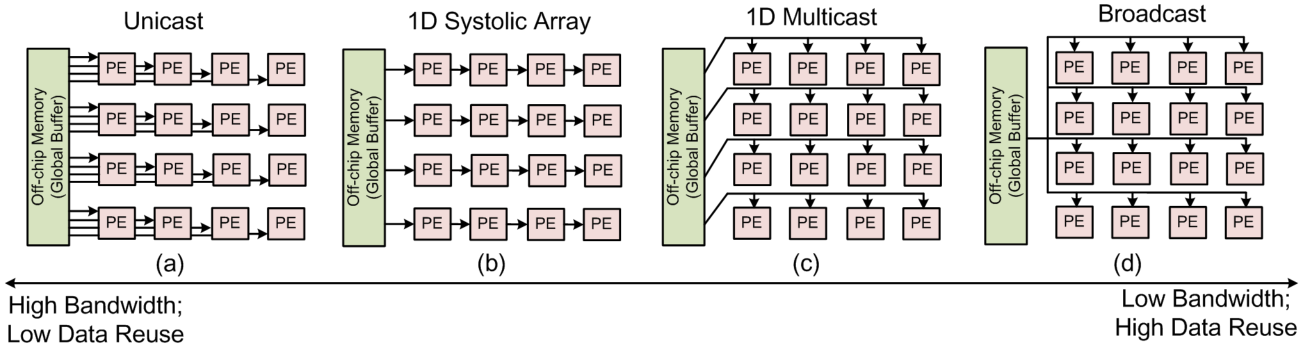
Unicast
1D Systolic Array
1D Multicast
Broadcast
Off-chip Memory (Global Buffer)
PE
(a)
(b)
(c)
(d)
High Bandwidth; Low Data Reuse
Low Bandwidth; High Data Reuse

Figure 2: A PE array design usually adopts the data reuse feature in DNN operations.

As mentioned before, the massive data communication latency between the off-chip memory and the processing units has become the system performance bottleneck in advanced DNN models. Because the DNN operation can be treated as a matrix multiplication [7], many ASIC-based DNN designs have improved the conventional unicast or systolic array architecture (Fig. 2(a) and (b)) and developed the multicast or broadcast architecture (Fig. 2(c) and (d)). In these architectures, the DNN computation (*i.e.*, the matrix multiplication) is accelerated using the characteristics of data reuse and data sparsity. In [24], Esmaeilzadeh *et al.* proposed a hardware called Neural Processing Unit (NPU) to approximate program functions with approximate algorithmic transformation. DianNao series, including DianNao [25], DaDianNao [26], and ShiDianNao [27], use Neural Functional Unit (NFU) to perform both neuron and synapse computations. Different from these platforms, PuDianNao [28] employs Functional Unit (FU) as the basic execution entity. Unlike the accelerators using NFU, PuDianNao can support additional machine learning algorithms such as k-means, linear regression, multi-layer perceptron (MLP), and support vector machine (SVM). In [29], Yin *et al.* proposed a hybrid-NN processor that is composed of two types of PEs and can support configurable heterogeneous PE arrays. By exploiting the characteristic of data reuse in the conventional CNN models, Chen *et al.* proposed the Eyeriss processor [7] that can optimize the neural network computation for a specific dataflow.

In spite of the fact that the ASIC-based DNN accelerators can achieve better performance with high power efficiency, the design flexibility is very low as the design is optimized based on the specification of one or few DNN models. To mitigate this design challenge, the FPGA-based DNN design methodology has become attractive in recent years [8]. FPGA is an integrated circuit that allows reconfiguration of interconnects between essential components, such as simple logic gates and memory. In any aspect of the application domain, it can properly reconfigure the programmable gates and find a specific architecture to reach the design constraint within a short period. Although the design flexibility of the FPGA-based designs is better than the ASIC-based designs, the computational flexibility is still limited during the runtime operation (*e.g.*, data paths are fixed at design time).

3 NOC-BASED NEURAL NETWORK DESIGN

3.1 NoC in Deep Learning

As mentioned before, CPU, GPU, ASIC, and FPGA can accelerate DNN operations, but each comes with a drawback when dealing with large-scale and hybrid DNN models. CPUs and GPUs are power-hungry and require additional complexity around their compute resources to facilitate software programmability [30]. ASIC and FPGA devices are more power-efficient, but they are configured and optimized for a specific model, and thus they lack computational flexibility at runtime.

To solve the problems mentioned above, Network-on-Chip (NoC) can be integrated into the DNN accelerator [13]. The NoC-based DNN accelerator separates the platform into data computation and communication. Such separation brings the advantages of better computational flexibility (*e.g.*, different DNN models can be efficiently executed on the platform) and design simplicity and scalability (*e.g.*, computation and computation can be separably designed and optimized) [12]. In the NoC-based design, after executing the neuron operations by a processing element, the obtained result is propagated through packets to the next PE. This process is repeated until the final result is obtained. In this way, no specific dataflow is required to be considered in order to run the target DNN model in the DNN accelerator efficiently. The important point is that the flexible communication between PEs enables handling different DNN models with varying data flows using an identical architecture. Some aspects of designing an NoC-based DNN accelerator have already been investigated in the literature. For example, an efficient mapping algorithm was proposed in [13], and a proper NoC topology was discussed in [31] and [32].

In summary, the offered high flexibility and regular structure make the NoC-based DNN accelerator a new and attractive design paradigm. With proper mapping and routing algorithms, the computational power and performance can be adjusted according to the underlying DNN models. As shown in Fig. 1, the NoC-based designs suggest a better trade-off between the reconfigurability feature and power consumption compared with the conventional design choices (*i.e.*, CPU, GPU, ASIC, and FPGA).

3.2 NoC-based ANN/DNN Simulator

NoC can reduce the interconnection complexity between PEs in DNN accelerators. To facilitate the NoC-based DNN design, a high-level NoC-based neural network simulator is needed. However, most of the current simulators focus either on the traffic behavior on the NoC [33][34][35] or the DNN computation [36][37].

One of the first tools to simulate the NoC-based ANN is proposed by Chen *et al.* [38] and is called NN-Noxim. This simulator facilitates the NoC-based ANN evaluation in the system level. NN-Noxim is an extension of Noxim [33], which is a popular NoC simulator to simulate the traffic behavior on the NoC system. To improve the hardware utilization, the computation of multiple neurons has been assigned into one Processing Element (PE). In other words, the original ANN is first clustered, and then the clustered ANN is mapped to the target NoC platform. NN-Noxim reports classification accuracy as well as the hardware results such as power and latency.

Although NN-Noxim has embedded the fundamental functions of ANN, it is not supporting convolution and pooling calculations. As a result, it cannot compute most CNN models, which include convolution and pooling operations. This issue has been addressed by the further development of the simulator in [39].

3.3 NoC Design Parameters in ANN/DNN Accelerators

Advanced DNN models usually consist of thousands of neuron computations, which results in huge computation, inter-neuron communication, and memory accesses. Designing optimized accelerators to solve these issues has become the core of many recent research efforts. Each PE has to execute neuron computations at high frequency, and thus fast and energy-efficient PE designs are highly demanded. The results of each PE should be delivered to another PE in the next layer. This data transmission requires massive inter-neuron communication, and thereby, an efficient interconnection platform is required to offer a flexible and scalable platform. On the other hand, PE outputs and partial sums are usually stored in the off-chip memory. Loading and offloading this data lead to massive memory accesses, which significantly increases the power consumption and data transfer latency. This issue also requires proper solutions in different design levels. In this section, we investigate the solutions that NoC can contribute to addressing.

Mapping and Clustering Algorithms: Mapping algorithms define the way in which neurons are allocated to the processing elements. Since a single neuron computation is very simple, it is a cost-efficient solution to assign multiple neurons to a PE [38]. So, clustering algorithms determine how neurons are grouped. The mapping and clustering algorithms should be designed in such a way that to reduce the overall data communication, packet latency, and power consumption. Different mapping and clustering algorithms have been introduced so far in the area of many-core systems [40][41]. However, it has been few contributions investigating the mapping of ANN applications into the NoC-based platform. To reduce traffic congestion and the data delivery latency, Liu *et al.* proposed an NN-aware mapping algorithm to map a neural network to the NoC platform [13]. By analyzing the communication flow in a neural network and the topology of the adopted NoC, authors apply an exhaustive search approach to find the optimal mapping results with the smallest data communication latency. This mapping algorithm reduces the average packet latency in comparison with sequential and random mapping. In another work, Firuzan *et al.* [42], proposed an approach to vertically or horizontally slice the neural network in order to map a large-scale neural network to the target NoC platform. In the vertical slicing, neurons in the same layer are grouped while in the horizontal slicing, neurons in different layers are grouped. Each group is then assigned to one PE in the NoC platform.

Topology: Topology determines the arrangement of PEs, routers, and links in the network. In [12], a hierarchical mesh topology is proposed to connect PEs. The proposed topology helps in reducing data communication in DNN computation. In [42], Firuzan *et al.* proposed a cluster-based topology with a reconfigurable architecture to construct an NoC-based neural network accelerator. Since a cluster handles most of the data communication locally, this kind of cluster-based NoC architecture can mitigate traffic congestion in the network. The clos topology is proposed by Yasoubi *et al.* [11] where PEs are connected through a direct path. The main motivation behind this work is to keep the data communication latency scalable with respect to the mesh size.

Routing Algorithms and Router Architecture: Routing algorithms define the possible routes to deliver a packet from a source to a destination. Routing algorithms in the NoC platform have been extensively investigated in literature [43][44]. However, the type of data flow in neural networks is rather different and is mostly one-to-many (*i.e.*, data multicasting/broadcasting) and many-to-one traffic (*i.e.*, data gathering), so proper routing algorithms are demanded to support these types of traffic patterns. Regarding the router architecture, some works have utilized traditional 5-port router with wormhole switching [26]. However, most of the current ANN designs are based on circuit-switching, where the paths are fixed at design time [7]. This is mainly to avoid the overhead of routing at runtime. Packet-switched NoC, on the other hand, brings an advantage of flexibility, which may contribute to reducing the memory accesses.

Memory Bandwidth: As mentioned before, load and off-loading data from memory leads to massive memory accesses that is counted as one of the leading design challenges in DNN accelerators. To alleviate this problem, Kwon *et al.* proposed a tree-based NoC interconnect to provide a higher bandwidth between memory and PEs [45]. Some data flow modes are also proposed to efficiently process different actions between the memory and the PEs such as scatter and gather. Due to employing the tree-based structure, the complexity of the data communication between the memory and PEs is reduced from $O(n^2)$ to $O(nlogn)$ in comparison with the mesh-based topology. As a result, this approach decreases the memory access latency.

4 NOC-BASED DNN ACCELERATOR DESIGN PARADIGM

To implement a DNN network, such as Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN), the network should be first flattened into an ANN-like network. As shown in

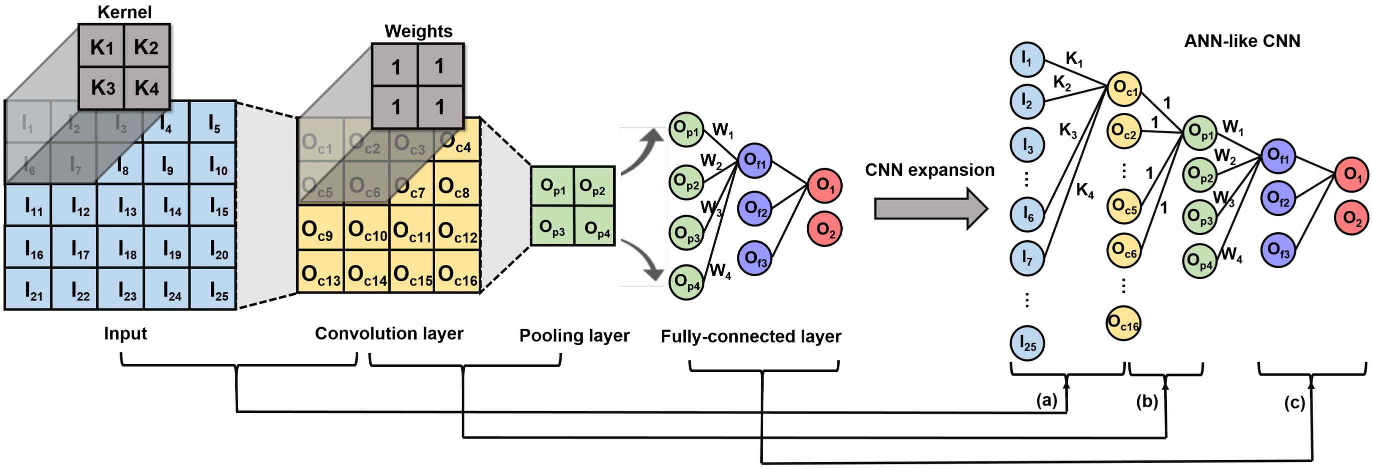

Figure 3: A CNN is flattened into ANN-like CNN: (a) the convolution layer and (b) the pooling layer can be expanded into a partially-connected layer; (c) the interconnection structure of a fully-connected layer is maintained.

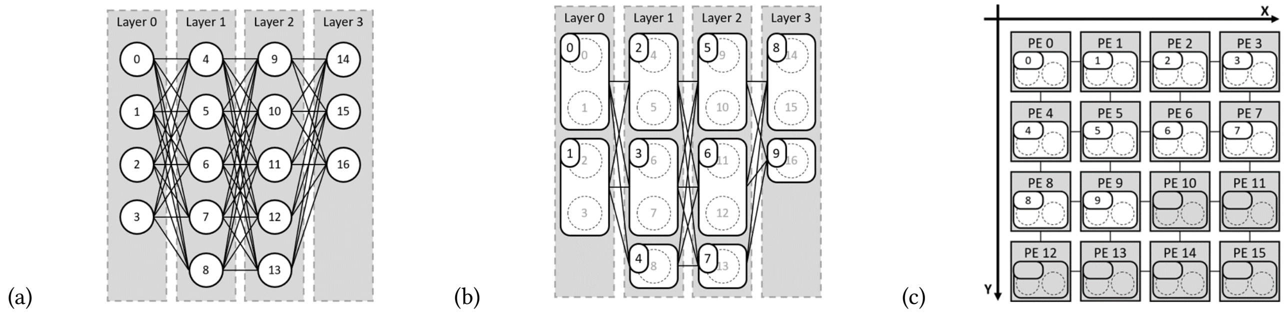

Figure 4: (a) The flattened DNN model (*e.g.*, ANN-like CNN); (b) clustering neurons; (c) mapping the clustered ANN model to the target NoC platform [38].

Fig. 3, a DNN network is usually composed of different types of layers, such as convolution layer, pooling layer, and fully-connected layer. In order to have a flexible platform, capable of executing different DNN models, we need to reach a unified computation model. For this purpose, similar to [39], we express the operations in different types of layers by Multiply-And-Accumulate (MAC) operations as follows:

In the ***convolution layer***, an output map is produced by performing a convolution on the input data and by using a kernel. Thereby, a partial convolution output (O_{c1}) can be expressed by MAC operations:

$$\begin{aligned} O_{c1} &= K_1 \times I_1 + K_2 \times I_2 + K_3 \times I_6 + K_4 \times I_7 \\ &= \sum_{l=1,2,3,4;j=1,2,6,7} (K_l \times I_j). \end{aligned} \tag{1}$$

By extending this equation to calculate other partial outputs, a semi fully-connected layer can be obtained, as illustrated in Fig. 3(a).

In the ***pooling layer***, the popular maximum pooling method is applied to capture the most important features [46]. To reuse the MAC operations and increase the hardware utilization, we multiply each input to the weight of 1 and apply the *argmax* function to find the maximal value. Thereby, a partial pooling output (O_{p1}) in the sub-sampling operation can be calculated by:

$$\begin{aligned} O_{p1} &= Max(O_{c1}, O_{c2}, O_{c5}, O_{c6}) \\ &= P_{i=1,2,5,6}(W \times I_i), \end{aligned} \tag{2}$$

where W is 1 and the pooling operator ($P_{i=1}^{N} I_j$) is equal to [39]:

$$P_{i=1}^{N} I_i = argmax(I_i). \tag{3}$$

By repeating this operation, all outputs of the pooling layer can be computed, as shown in Fig. 3(b).

In the ***fully-connected layer***, each neuron in one layer is fully connected to all neurons in the next layer. As illustrated in Fig. 3(c), a partial output in the fully-connected layer O_{f1} can be expressed by:

$$\begin{aligned} O_{f1} &= W_1 \times O_{p1} + W_2 \times O_{p2} + W_3 \times O_{p3} + W_4 \times O_{p4} \\ &= \sum_{k=1}^{4} (W_k \times O_{pk}). \end{aligned} \tag{4}$$

Using a similar equation, the final outputs (*i.e.*, O_1 and O_2) can be obtained.

Based on the analysis, we can conclude that most operations in CNN (such as convolution and pooling) can be executed using MAC operations. As shown in Fig. 3, by this procedure, a flattened CNN model is obtained, that is called ANN-like CNN.

After obtaining the ANN-like CNN (Fig. 4(a)), mapping algorithms [38] should be applied to map the flattened model to the NoC platform. As shown in Fig. 4(b), the first task is to divide neurons into different execution groups. We cluster neurons layer by layer, and the number of neurons per group depends on the computational power of each PE. As shown in this figure, neurons in Layer 1 are divided into Group 0 and Group 1 and their computations are assigned to PE0 and PE1, respectively (Fig. 4(c)). Similarly, neurons in Layer 2 are divided into Group 2, Group 3, and Group 4 and their corresponding PEs are PE2, PE3, and PE4, respectively. When the computations in a PE is completed, the partial results are packaged and sent to all PEs in the next layer. For example, the obtained results from PE0 and PE1 are transmitted to all PEs in Layer 2 (*i.e.,* PE2, PE3, and PE4).

5 EVALUATION RESULTS AND ANALYSIS

In this section, we compare the conventional DNN design with that of the NoC-based design. The comparison is made in terms of performance and the number of memory accesses. UNPU [47] is selected as the representative of the conventional DNN design. The reason for this selection is that UNPU provides a look-up table-based PE (LBPE) to support matrix multiplication and MAC operation. Thereby, UNPU offers better computational flexibility in calculating various DNN operations, in comparison with other DNN designs such as Eyeriss [7]. To evaluate the UNPU design, we have described its computing behavior in SystemC while for the NoC-based DNN design, we utilized the CNN-Noxim simulator [39]. To have a fair comparison, the latency model of UNPU is used for the analysis of both conventional and NoC-based designs. In addition, for the sake of simplicity, we employed mesh topology, random mapping, and XY routing in the simulation process. However, the performance can be improved by employing other advanced topologies [42][45], mapping [40], and routing techniques [48]. For the memory bandwidth, we assume that the bandwidth is enough to access 16 bits of data within one cycle (*i.e.*, similar to the assumption in UNPU [47]). With regards to the target DNN networks, we evaluated LeNet [49], MobileNet [50], and VGG-16 [4] as the representative of a small, medium, and large-scale DNN model, respectively. As shown in Fig. 5, the ANN-like CNN model of LeNet, MobileNet, and VGG-16 involves 286K, 269M, and 17B MAC operations, respectively.

5.1 The Number of Memory Accesses

In conventional DNN accelerators, the overall system performance is highly affected by the off-chip memory accesses [45]. These memory accesses are necessary as the partial results are stored in the off-chip memory for further processing by other PEs. In other words, PEs communicate with each other, mainly through read/write operations from/to the off-chip memory. In the NoC-based design, however, PEs can communicate with each other through the network, and thus the off-chip memory accesses can be significantly reduced. Moreover, the cache-coherence technique is an important aspect in NoC to reduce memory accesses. The data synchronization can be achieved with mature memory-coherent techniques in NoC [51]. Using the synchronization techniques, for example, a weight can be loaded from the off-chip memory into one PE and then synchronized with other PEs that need this weight. Hence, the DNN-based NoC design not only reduces the number of off-chip memory accesses but performs the neuron computing distributively.

Table 1: Comparison between conventional and NoC-based designs regarding the number of off-chip memory accesses.

	Conventional [47]	NoC-based Design
LeNet [49]	884,736	**47,938 (-94.6%)**
MobileNet [50]	1,061,047,296	**4,360,616 (-99.6%)**
VGG-16 [4]	1,165,128,192	**138,508,072 (-88.1%)**

In this section, we compare the number of memory accesses in the NoC-based versus the conventional design. We assume that the resources on the platform are enough to transfer all parameters of the given model from the off-chip memory to the NoC platform at once. This assumption is aligned with recent NoC-based architectures (such as Intel's Skylake [52]) that reduce the global memory size and increase those of the local ones. TABLE 1 shows the benefit of the NoC-based DNN design, assuming that sufficient distributed on-chip local memory is available. For example, in the LeNet network, the NoC-based design first read all network parameters from the off-chip memory through $47,938$ read accesses. It should be noted that the local memory accesses have not been counted in TABLE 1 as their imposed latency and power consumption are much lower than off-chip memory accesses. The conventional design, on the other hand, requires $884,736$ memory accesses due to frequent off-chip read and write operations. Similarly, the number of off-chip memory accesses can be reduced from approximately 106M to 4M in MobileNet and from 10B to 138M in VGG-16 .

In sum, the results show that the distributive neuron computing in the NoC-based design, on average, can reduce the memory accesses by 94%, 99%, and 88% in the LeNet, MobileNet, and VGG-16 network, respectively. This memory access reduction can reduce the power consumption and increase the system performance significantly.

5.2 Performance under Different NoC Sizes

In this section, we study the performance of the selected DNN models under different NoC sizes as well as different group sizes. For the performance analysis, we report the PE computational latency, the NoC data delivery latency, and the total latency. The PE computational latency is the total execution time to compute neuron operations in PEs, taking into account the parallel computing. The data delivery latency is the total time that packets spend in NoC.

For the evaluation, we consider three DNN models as LeNet, MobileNet, and VGGnet-16; and three different NoC sizes as 6×6, 8×8, and 12×12. The involved packet injection rate (PIR)depends on the execution time of a single PE, which further relies on the group size (*i.e.*, the PIR is equal to the reciprocal of the group size in this work). To show the impact of different group sizes, we also investigate the performance under three different group settings (*i.e.*, the number of neuron computations per PE).

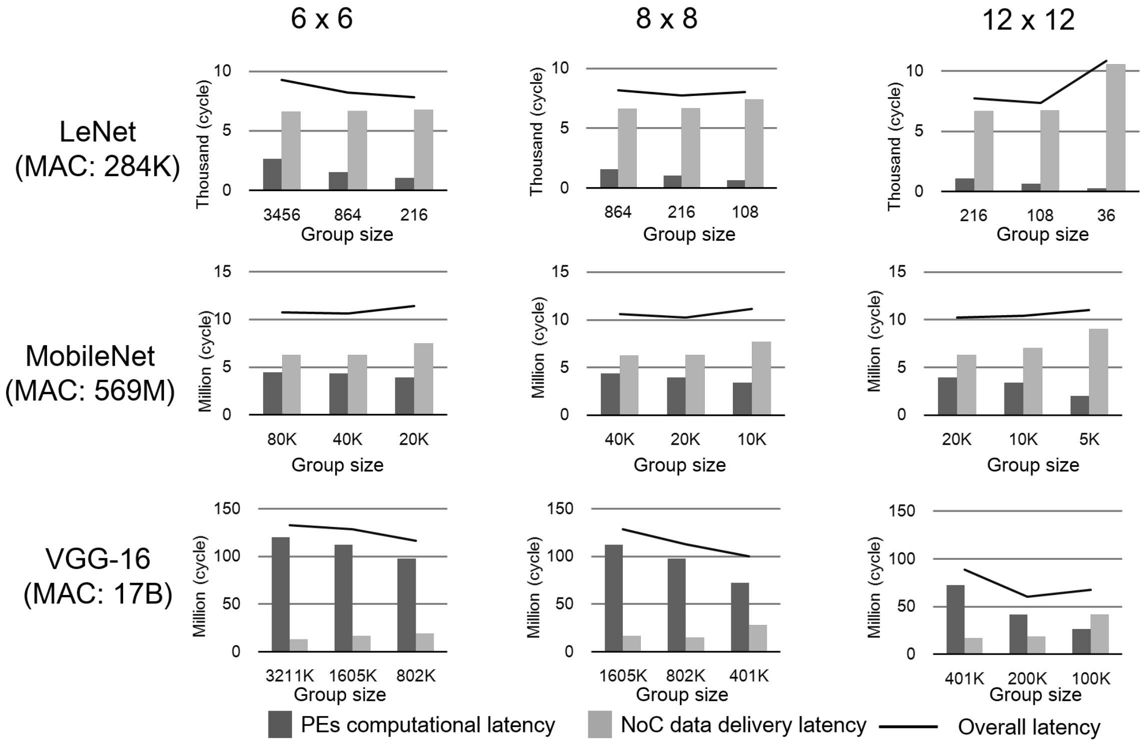

Figure 5: The performance evaluation of three DNN models under different NoC and neuron group sizes.

Let us first assume the LeNet network with 284K operations under the 6 × 6 NoC size. We consider a configuration where the maximum computational capacity of a PE is 3456 neuron computations. To assign neurons to PEs, we start with the first PE in the platform and allocate the computation of as many neurons as to utilize the maximum PE computational capacity (*e.g.*, 3,456 neurons or 10,368 MAC operations by assuming that each neuron involves 3 MAC operations). Then we continue with the next PE in the platform and so on. If all PE assignments for one layer is completed, the assignment in the next layer will be started using a new PE.

As can be seen in all sub-figures, by reducing the computational capacity of a PE, the PE latency is reduced. On the other hand, by a smaller group size setting, more transaction data is generated and delivered through the NoC platform. Hence, the NoC data delivery latency will be increased. The growth of the NoC latency is smaller in small network sizes (*e.g.*, 6 × 6 and 8 × 8) while the difference becomes more significant in larger network sizes (*e.g.*, 12 × 12). The main reason is the longer paths in larger networks.

It can also be observed that in smaller networks (*e.g.*, LeNet and MobileNet), the total latency is dominated by the NoC communication latency while in large-scale networks (*e.g.*, VGG-16), the total latency is dominated by the PE computational latency. Therefore, the neuron clustering should be done in such a way that neither the NoC communication latency nor the PE computation latency becomes the system performance bottleneck. Therefore, besides improving the PE design, efficient topology arrangement, routing, and mapping algorithms are also essential to improve the performance of the NoC-based DNN accelerators.

6 CONCLUSION

In this paper, we first investigated conventional platforms for DNN computing (*i.e.*, CPU, GPU, ASIC, and FPGA) and discussed the pros and cons of each platform. Then, we introduced the NoC-based DNN accelerator which brings the main benefits of reducing off-chip memory accesses and enhancing runtime computational flexibility. For the execution of a DNN model in a NoC-based platform, the model was first flattened into an ANN-like network. Then it was clustered, and each group was mapped to a PE by using random mapping. The XY-routing was used to route packets in the network. Compared with the conventional DNN design, the number of memory accesses in NoC-based design was reduced by 88% up to 99% under three different DNN models. The performance analysis showed a direct relationship between the neuron clustering and its effect on the PE and NoC latency. Finally, we came to the conclusion that efficient PE and NoC designs are both critically important to keep the system performance high in the NoC-based DNN accelerators.

ACKNOWLEDGMENTS

This work was supported by the Ministry of Science and Technology under the grant MOST 108-2218-E-110-010 and MOST 108-2218-E-110-006, TAIWAN; the STINT and VR projects, SWEDEN.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[2] L. Xu, J. S. Ren, C. Liu, and J. Jia, "Deep convolutional neural network for image deconvolution," in *Advances in neural information processing systems*, 2014, pp. 1790–1798.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, 2012, pp. 1097–1105.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[5] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2015.

[6] V. Vanhoucke and M. Z. Mao, "Improving the speed of neural networks on cpus," 2011.

[7] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan 2017.

[8] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, "High-performance fpga-based cnn accelerator with block-floating-point arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–12, 2019.

[9] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, "Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, Aug 2013.

[10] 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.

[11] A. Yasoubi, R. Hojabr, H. Takshi, M. Modarressi, and M. Daneshtalab, "Cupan – high throughput on-chip interconnection for neural networks," in *Neural Information Processing*, S. Arik, T. Huang, W. K. Lai, and Q. Liu, Eds. Cham: Springer International Publishing, 2015, pp. 559–566.

[12] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss v2: A Flexible and High-Performance Accelerator for Emerging Deep Neural Networks," *arXiv e-prints*, p. arXiv:1807.07928, Jul 2018.

[13] X. Liu, W. Wen, X. Qian, H. Li, and Y. Chen, "Neu-noc: A high-efficient interconnection network for accelerated neuromorphic systems," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2018, pp. 141–146.

[14] B. Wolford, T. Speier, and D. Bhandarkar, "Qualcomm centriq 2400 processor," in *Hot Chips: A Symposium on High Performance Chips*, ser. HC29, 2017.

[15] J. Jeffers, J. Reinders, and A. Sodani, *Intel Xeon Phi Processor High Performance Programming*, 1st ed. Cambridge, MA, USA: Morgan Kaufmann, 2016.

[16] J. Dai, Y. Wang, X. Qiu, and D. D. et al. (2017) Bigdl: A distributed deep learning framework for big data.

[17] F. Abuzaid, S. Hadjis, C. Zhang, and C. Ré, "Caffe con troll: Shallow ideas to speed up deep learning," *CoRR*, vol. abs/1504.04343, 2015.

[18] T. Baji, "Evolution of the gpu device widely used in ai and massive parallel processing," in *IEEE 2nd Electron Devices Technology and Manufacturing Conference*, ser. EDTM'18, 2018, pp. 7–9.

[19] Q. Wang, N. Li, L. Shen, and Z. Wang, "A statistic approach for power analysis of integrated gpu," *Soft Computing*, vol. 23, no. 3, pp. 827–836, 2019.

[20] C. Farabet, C. Poulet, and Y. LeCun, "An fpga-based stream processor for embedded real-time vision with convolutional networks," in *IEEE 12th International Conference on Computer Vision Workshops*, ser. ICCV'09, 2009.

[21] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," *CoRR*, vol. abs/1502.02551, 2015.

[22] B. Moons and M. Verhelst, "A 0.3âĂŞ2.6 tops/w precision-scalable processor for real-time large-scale convnets," in *IEEE Symposium on VLSI Circuits*, ser. VLSI-Circuits'16, 2016, p. 2.

[23] S. Yin, P. Ouyang, and S. Tang, "A 1.06-to-5.09 tops/w reconfigurable hybrid-neural-network processor for deep learning applications," in *IEEE Symposium on VLSI Circuits*, ser. VLSI-Circuits'17, 2017, p. 2.

[24] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," *IEEE Micro*, vol. 33, no. 3, pp. 16–27, May 2013.

[25] 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 *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14, 2014, pp. 269–284.

[26] T. Luo, S. Liu, L. Li, Y. Wang, S. Zhang, T. Chen, Z. Xu, O. Temam, and Y. Chen, "Dadiannao: A neural network supercomputer," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 73–88, Jan 2017.

[27] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, June 2015, pp. 92–104.

[28] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15, 2015, pp. 369–381.

[29] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, S. Zheng, T. Lu, J. Gu, L. Liu, and S. Wei, "A high energy efficient reconfigurable hybrid neural network processor for deep learning applications," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 968–982, April 2018.

[30] N. P. Jouppi and et. el., "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17, 2017, pp. 1–12.

[31] J. Liu, J. Harkin, L. P. Maguire, L. J. McDaid, J. J. Wade, and G. Martin, "Scalable networks-on-chip interconnected architecture for astrocyte-neuron networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 12, pp. 2290–2303, Dec 2016.

[32] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks," *CoRR*, vol. abs/1712.01507, 2017.

[33] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Cycle-accurate network on chip simulation with noxim," *ACM Trans. Model. Comput. Simul.*, vol. 27, no. 1, pp. 4:1–4:25, Aug. 2016.

[34] D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2013, pp. 86–96.

[35] A. V. de Mello, "Atlas-an environment for noc generation and evaluation," 2011.

[36] R. N. et al., "Gan playground - experiment with gan in your browser," 2017.

[37] D. Smilkov and S. Carter, "A neural network playground - tensorflow," 2017.

[38] K. J. Chen and T. Wang, "Nn-noxim: High-level cycle-accurate noc-based neural networks simulator," in *2018 11th International Workshop on Network on Chip Architectures (NoCArc)*, Oct 2018, pp. 1–5.

[39] K.-C. J. Chen, T.-Y. G. Wang, and Y.-C. A. Yang, "Cycle-accurate noc-based convolutional neural network simulator," in *Proceedings of the International Conference on Omni-Layer Intelligent Systems*, ser. COINS '19, 2019, pp. 199–204.

[40] "Testing aware dynamic mapping for path-centric network-on-chip test," *Integration*, vol. 67, pp. 134 – 143, 2019.

[41] L. Huang, S. Chen, Q. Wu, M. Ebrahimi, J. Wang, S. Jiang, and Q. Li, "A lifetime-aware mapping algorithm to extend mttf of networks-on-chip," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2018, pp. 147–152.

[42] A. Firuzan, M. Modarressi, and M. Daneshtalab, "Reconfigurable communication fabric for efficient implementation of neural networks," in *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, June 2015, pp. 1–8.

[43] M. Palesi and M. Daneshtalab, *Routing Algorithms in Networks-on-Chip*. Springer Publishing Company, Incorporated, 2013.

[44] M. Ebrahimi, M. Daneshtalab, F. Farahnakian, J. Plosila, P. Liljeberg, M. Palesi, and H. Tenhunen, "Haraq: Congestion-aware learning model for highly adaptive routing algorithm in on-chip networks," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, May 2012, pp. 19–26.

[45] H. Kwon, A. Samajdar, and T. Krishna, "Rethinking nocs for spatial neural network accelerators," in *Proceedings of the Eleventh IEEE/ACM International Symposium on Networks-on-Chip*, ser. NOCS '17, 2017, pp. 19:1–19:8.

[46] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*. Springer, 2010, pp. 92–101.

[47] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. 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, Jan 2019.

[48] M. Ebrahimi and M. Daneshtalab, "Ebda: A new theory on design and verification of deadlock-free interconnection networks," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, June 2017, pp. 703–715.

[49] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in neural information processing systems*, 1990, pp. 396–404.

[50] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[51] D. Giri, P. Mantovani, and L. P. Carloni, "Noc-based support of heterogeneous cache-coherence models for accelerators," in *Proceedings of the Twelfth IEEE/ACM International Symposium on Networks-on-Chip*, ser. NOCS '18, 2018, pp. 1:1–1:8.

[52] J. Doweck, W. Kao, A. K. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz, "Inside 6th-generation intel core: New microarchitecture code-named skylake," *IEEE Micro*, vol. 37, no. 2, pp. 52–62, Mar 2017.