Automated Deep Neural Network Inference Partitioning for Distributed Embedded Systems

Fabian Kreß, El Mahdi El Annabi, Tim Hotfilter, Julian Hoefer, Tanja Harbaum, Juergen Becker
Karlsruhe Institute of Technology, Karlsruhe, Germany
{fabian.kress, hotfilter, julian.hoefer, harbaum, becker}@kit.edu

***Abstract*—Distributed systems can be found in various applications, e.g., in robotics or autonomous driving, to achieve higher flexibility and robustness. Thereby, data flow centric applications such as Deep Neural Network (DNN) inference benefit from partitioning the workload over multiple compute nodes in terms of performance and energy-efficiency. However, mapping large models on distributed embedded systems is a complex task, due to low latency and high throughput requirements combined with strict energy and memory constraints.**

In this paper, we present a novel approach for hardware-aware layer scheduling of DNN inference in distributed embedded systems. Therefore, our proposed framework uses a graph-based algorithm to automatically find beneficial partitioning points in a given DNN. Each of these is evaluated based on several essential system metrics such as accuracy and memory utilization, while considering the respective system constraints. We demonstrate our approach in terms of the impact of inference partitioning on various performance metrics of six different DNNs. As an example, we can achieve a 47.5% throughput increase for EfficientNet-B0 inference partitioned onto two platforms while observing high energy-efficiency.

***Index Terms*—Embedded Systems, Deep Neural Networks, Inference Partitioning, Hardware/Software Co-Design**

I. Introduction

The hardware requirements for Deep Neural Network (DNN)-based applications such as Advanced Driver-Assistance Systems (ADAS) or assistance robots are constantly increasing with regard to various performance indicators including latency, throughput and energy efficiency. To address this, several dedicated hardware accelerator architectures have been proposed over the last years. However, these are either tailored to a specific workload offering high performance but less flexibility, or designed to accelerate various applications but suffer from low efficiency. As a result, systems consisting of more than a single hardware accelerator are required to offer well-fitted hardware for different applications executed on embedded platforms. Distributed systems in those use cases consist of several hardware platforms close to the sensors and a central compute unit. Implementing multiple hardware accelerators for DNN inference in a single component thereby leads to a significant increase in design complexity to avoid severe bottlenecks introduced through limited memory bandwidth. As a result, distributing hardware accelerators for DNN inference is advantageous [1].

However, mapping DNNs on such systems in a performant and energy-efficient manner is a complex task, as the architecture of the hardware accelerators must be taken into account in addition to the transmission overhead. In this paper, we address this problem by proposing an automated inference partitioning approach to find an optimal trade-off considering several performance metrics and also hardware constraints. Our proposed framework thereby uses a graph-representation of the DNN to find a Pareto-optimal mapping, which offers low latency, high energy-efficiency and high DNN accuracy. It supports multiple hardware accelerators tailored to different workloads and also considers the overhead introduced by the link between the system platforms. In summary, our contributions are as follows:

- We present our automated framework to evaluate several important metrics of potential DNN partitioning points.
- We introduce our systematic approach to determine an optimal partitioning based on given constraints and the optimization goals.
- We evaluate our methodology for several Convolutional Neural Networks (CNNs) and present our experimental results regarding latency, energy consumption, throughput, top-1 accuracy and memory consumption.

II. Related Work

In recent years, partitioning of DNN inference has been studied in different contexts. Several research has been done on inference partitioning of multi-FPGA environments [2], [3]. They focus on efficient use of shared memory resources between FPGAs to reduce communication overhead and to increase performance. Others [1], [4] perform partitioning in edge clusters and thereby try to distribute the DNN inference in parallel over multiple devices.

These approaches are not suitable for systems that do not consist of a computing cluster to distribute the workload. Consequently, there has also been research on layer-wise DNN inference partitioning as shown in Table I. Hu et al. [5] proposed an automated dynamic scheduling approach which takes network conditions into account. The graph-based algorithm thereby either tries to optimize the inference towards minimum latency or maximum throughput, depending on the network load. In the evaluation, the authors can show the benefits of partitioning DNN inference, however, their scheduling algorithm does not take hardware performance and limitations into account and is therefore not suited for resource constrained systems. Moreover, the scheduler Neurosurgeon [6] has been proposed to optimize DNN inference partitioning between mobile edge and cloud. It is designed to automatically

 DOI: 10.1109/ISVLSI61997.2024.00019

Framework	DNN Framework	Search Method	Auto. Part.	Multi Part.	HW-aware Part.	(Re-)Training	Target Platform	Optimization Metrics
Hu et al. [5]	Caffe	Graph-based	✔	✘	✘	✘	RPi 3	latency, throughput
Neurosurgeon [6]	Caffe	Model-based	✔	✘	✘	✘	CPU/GPU	latency, energy
Yao et al. [7]	TensorFlow	Model-based	✔	✘	✘	✔	CPU/GPU	latency, bandwidth
Ko et al. [8]	*N/A*	Simulation	✘	✘	✔	✔	ASIC	energy, throughput, bandwidth, accuracy
CNNParted [9]	PyTorch	Simulation	✘	✘	✔	✘	ASIC/GPU	latency, energy, bandwidth
AxoNN [10]	TensorFlow	Graph-based	✔	✔	✔	✘	GPU/DLA	latency, energy
Our Proposal	ONNX	Graph-based	✔	✔	✔	✔	ASIC	latency, bandwidth, energy, memory, accuracy, throughput

TABLE I: State-of-the-art methodologies for evaluation of DNN inference partitioning in the edge

determine a beneficial partitioning point in terms of overall latency and mobile edge energy consumption. Similarly to the previously mentioned approach, their scheduling does not provide a hardware-optimized DNN inference partitioning.

Apart from optimizing latency, energy consumption and throughput by applying inference partitioning, reducing bandwidth is also a major objective in this context. Therefore, Yao et al. [7] proposed an offloading approach which adds an autoencoder structure at the partitioning point to further compress the data transmitted between computing nodes in the system. Since this approach can impact the overall accuracy of the DNN, training of the autoencoder has to be done with regard to all potential partitioning points. Again, the methodology does not take hardware constraints into account and is only evaluated on two mobile devices. In contrast to previous methodologies, Ko et al. [8] proposed a simulation-based DNN inference partitioning approach for more resource-constrained and Application-Specific Integrated Circuit (ASIC) platforms. The presented methodology thereby allows to determine a partitioning point based on energy efficiency and throughput. To further reduce the bandwidth required when distributing inference, the authors propose to encode intermediate feature maps using lossy encoding and fine-tuning of the partitioned DNN. However, the methodology presented only allows to derive design guidelines for other applications and does not include an algorithm for automated partitioning. Similarly, the open-source tool CNNParted [9] offers a hardware-aware design space exploration of beneficial partitioning points but only outputs latency, energy and bandwidth metrics of each potential point to the designer. Nevertheless, since it is based on open-source tools for modeling and evaluating arbitrary DNN hardware accelerators, the exploration also allows to compare different architectures during design time.

However, none of these approaches considers multiple partitioning points during inference. AxoNN [10] aims to find near-optimal schedules for multi-accelerator based System-on-Chips (SoCs) on Commercial Off-The-Shelf (COTS) platforms such as Nvidia Jetson Xavier AGX containing a GPU and a Deep Learning Accelerator (DLA). In particular, the transition costs between the available hardware accelerators in a shared-memory system are taken into account during evaluation by the authors, highlighting that near-optimal schedules mostly include only a single transition between GPU and DLA. The evaluation of partitioning, however, only takes latency and energy consumption into account. As a result, this approach is not suitable for distributed embedded systems.

III. Problem Statement

The scheduling of DNN inference depends primarily on the system architecture, i.e. the number of available hardware accelerators in the system. In the simplest case, we only have two computing platforms available for executing DNN inference and can define a partitioning point as follows:

Definition 1 (Partitioning Point)**.** *In a system consisting of two hardware platforms A and B, a partitioning point is a layer l_p with $p \in \{1, \dots, L\}$ of a given DNN with L layers, such that:*

- *Each layer l_i with $i \in \{1, \dots, L\}$ is executed once per inference on exactly one platform in the system.*
- *Each layer l_i with $i \in \{0, \dots, p\}$ is executed on A.*
- *Each layer l_i with $i \in \{p+1, \dots, L\}$ is executed on B.*
- *The intermediate feature map f_p of l_p is transmitted over a link from A to B.*

Determining an optimal partitioning point under given constraints depends on the main optimization goal which is application specific. As a result, we formulate the objective functions as a minimization problem based on weighted sum of cost functions:

Definition 2 (Minimization Problem)**.** *The minimization problem for a system consisting of two hardware platforms A and B is given as*

$$\underset{l_p=0\dots L}{\text{minimize}} \sum_{i=0}^{N} c_i \cdot \theta_i(l_p)$$

with L being the number of DNN layers, N the number of optimization criteria, and c_i being application dependent coefficients of the cost functions θ_i.

Based on the state of the art, we find that existing frameworks cover overall five different optimization metrics for determining an optimal partitioning point: latency $d(l_p)$, throughput $th(l_p)$, energy consumption $e(l_p)$, link bandwidth $bw(l_p)$, and accuracy $acc(l_p)$. With the emergence of large-scale models and the further increasing complexity of DNNs [11], memory size is becoming a potential bottleneck in embedded

systems as well. Therefore, solving the optimization problem for real-world hardware requires to include the required amount of memory $m_A(l_p)$ and $m_B(l_p)$ for storing DNN parameters and intermediate feature maps on platform A and B, respectively. Each of the different metrics is modeled as a cost function $\theta_0 \dots \theta_N$ depending on the partitioning point and can be constrained as part of the minimization problem.

IV. Our Proposed Framework

Even for a system configuration that features only two hardware accelerators, determining an optimal partitioning point is a non-trivial task. In this section, we therefore present our framework for exploring the design space of potential partitioning points that meet the previously stated objectives. An overview of our approach is given in Figure 1. As input, the framework takes a DNN description as Open Neural Network Exchange (ONNX) file, problem constraints, and the main optimization objective. Using ONNX as input allows to combine our proposed framework with a wide range of DNN frameworks such as TensorFlow or PyTorch.

First, the DNN is converted into a graph representation and then analyzed to find all potential partitioning points without considering given constraints. For reducing the number of feasible schedules of the DNN, these partitioning points are filtered based on memory and link evaluation. This step thereby requires a link model to determine bottlenecks in the link that would otherwise violate system constraints. Furthermore, since hardware accelerators often use integer or fixed-point representation instead of floating-point values for calculation, quantization has to be applied for DNN inference in hardware. However, accuracy of the DNN can be significantly impacted by using such number formats. Consequently, the accuracy drop is evaluated for each remaining partitioning point, optionally including retraining.

The list of all partitioning points not violating accuracy, bandwidth or memory constraints is then forwarded to the Hardware (HW) Evaluation step, where latency, throughput and energy consumption metrics are evaluated for different schedules. Thereby, the performance of hardware accelerators depends on the mapping for each layer and the used process technology. We use Timeloop [12] and Accelergy [13] to find a near-optimal mapping for each layer to the hardware accelerator and to estimate latency and energy consumption.

In the next step, the collected metrics are used to evaluate all potential partitioning points fulfilling the given constraints. Since we perform multi-objective optimization, we use the NSGA-II to determine Pareto-optimal points in the resulting set of potential partitioning points [14]. Thereby, the partitioning point serves as variable of the partitioning problem. Since the complexity of a DNN varies significantly, the population size as well as the number of generations is set depending on the number of layers. Finally, the framework identifies the most favorable DNN partitioning point that fulfills the main optimization objective.

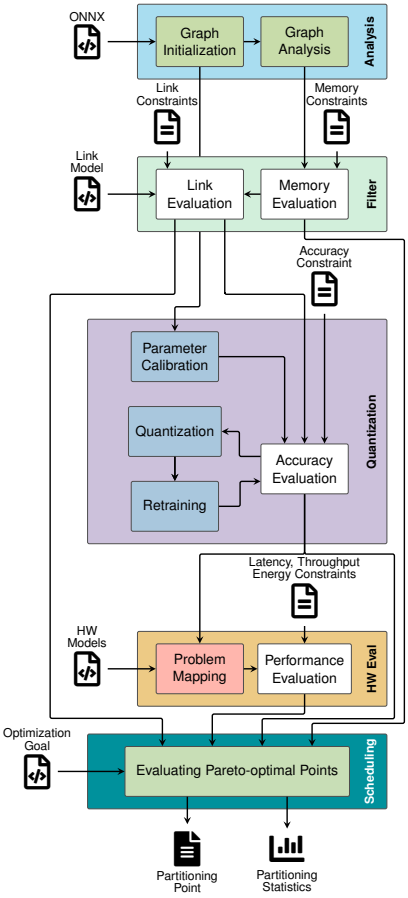

Fig. 1: Overview of our proposed framework. First, a graph is generated based on the ONNX description. After filtering of potential partitioning points considering memory and link constraints, quantization is performed and evaluated. Finally, the framework estimates performance on hardware and selects a Pareto-optimal point.

A. Graph Analysis

Finding potential partitioning points in a given DNN can be done in different ways as shown in Table I. Simulation-based approaches such as CNNParted [9] try to determine branches in the network architecture based on comparing output and input shapes of layers. However, since this approach is only analyzing the DNN structure on basic layer level, simple mathematical operations such as adding, concatenating and flattening tensors have to be identified using complex logic. In contrast, our proposed framework is able to directly extract the graph representation from the ONNX specification.

Furthermore, skip connections are often used to address the vanishing gradient problem. As a result, the Directed Acyclic Graph (DAG) of a given DNN can consist of parallel branches that can be executed independently. Our framework therefore first performs a topological sort of the DAG to find a linear ordering of its vertices. This step is required to determine a schedule for executing the DNN inference sequentially on each of the available hardware accelerators. In case there are

parallel branches, the algorithm randomly selects one of the unscheduled layers as the next node to be added to the linear sequence.

B. Memory Size Estimation

In many recent SoCs, memory allocates a significant part of die area. At the same time, the complexity of DNNs is constantly rising and, thus, the memory demand. As a result, we also have to consider this when looking for a feasible schedule for DNN inference. The required memory size for a sequence of layers without branches is formalized as follows:

Definition 3 (Memory Size)**.** *The memory size required to execute non-pipelined branch-free DNN inference from layer l_n to l_m on a hardware accelerator A is given by*

$$m_A(l_n, l_m) = \left(\sum_{i=n}^{m} s_i + \max(a_n, \ldots, a_m) \right) \cdot b_A, \quad n \leq m$$

$$a_j = f_{j,in} + f_{j,out}$$

with s_i being the number of parameters of layer l_i, a_j the sum of input and output feature map size $f_{j,in}$ and $f_{j,out}$ of layer j, and b_A the quantized bit width of parameters and feature maps.

Based on this, the memory requirements of each individual layer can be calculated. As soon as the resulting demand exceeds the available on-chip memory of a platform, all following potential partitioning points are removed from the list of feasible schedulings. However, in case of branches in the DNN topology, different schedules have to be evaluated since parallel layers can be executed in different orders. Therefore, the framework builds subgraphs for these parallel branches to find the schedule with minimum memory requirements according to Definition 3.

C. Accuracy Exploration

Hardware floating-point operations are not as energy efficient as integer computations. As a result, hardware accelerators in embedded systems typically use integer or fixed-point Multiply-Accumulate (MAC) operations [15], [16]. While DNN models generally tolerate quantization very well, radical quantization schemes worsen the model accuracy. Hence, quantization also has to be considered for finding a beneficial inference schedule in multi-accelerator systems.

Before the actual exploration, our tool has to perform a parameter calibration to determine the ranges of features maps and weights. In the next step, the impact of quantization on the model accuracy for each potential partitioning point has to be evaluated. To determine the resulting accuracy, our proposed framework uses fake quantization, which allows to obtain the results quickly. The degree of quantization thereby depends on the hardware specification of the implemented accelerator in each part of the system. However, DNN accuracy can suffer significantly from quantization, especially in cases where a small bit width is used for inference in the hardware accelerator. As a result, our framework offers the possibility to run Quantization-Aware Training (QAT) based on the trained network parameters to restore the accuracy of the model.

D. Throughput Estimation

Applications such as autonomous driving have certain throughput requirements to enable reliable decision making. Running DNN inference on a single accelerator might provide suitable latency but low throughput. Therefore, it must be considered as well when searching for an optimal partitioning point of a DNN. Since we assume that the hardware platforms can work in parallel as an asynchronous pipeline, the throughput is determined by the platform with the highest latency. As a result, it can be formalized as follows:

Definition 4 (Throughput)**.** *The throughput for partitioned DNN inference on a system consisting of two hardware platforms A and B connected over a $link$ is given by*

$$th(l_p) = min\left(\frac{1}{d_A}, \frac{1}{d_{Link}}, \frac{1}{d_B} \right)$$

with d being the latency of each involved module of the system.

V. Evaluation

Finally, we show experimental results for inference partitioning using our proposed framework. The explorations have been conducted on a system consisting of a 64-core AMD EPYC 9554 and an NVIDIA RTX A6000 running Rocky Linux. To evaluate the latency and energy consumption of each hardware accelerator, Timeloop is configured to use a linear-pruned search algorithm and a victory condition of 100.

A. Workload

First, we performed several explorations for a system model consisting of two hardware platforms A and B according to the problem as described in Definition 2. Platform A is based on a 16-bit Eyeriss-like architecture [16] running at 200 MHz (EYR), and platform B uses a Simba-like accelerator [15] at 200 MHz (SMB). In our simulated system model, the platforms are connected to each other via Gigabit Ethernet. To explore the transition costs of the link, i.e. latency and energy consumption, we used the open-source model provided by CNNParted [9]. We tested our framework for six feed-forward image classification CNNs, i.e. EfficientNet-B0, ResNet-50, RegNetX_400MF, VGG-16, GoogLeNet, and SqueezeNet V1.1 pretrained on ImageNet dataset. In the quantization stage, we performed two epochs of QAT with 1,281,167 images for training and validated the quantized models with 8,192 images from the ImageNet dataset.

B. Experimental Results

The simulation time of our proposed framework is dominated by the optional retraining time as expected. In particular, for models such as EfficientNet-B0, performing retraining for each potential partitioning point takes each about one hour on our system using 32 parallel worker threads for the data loader. In contrast, graph analysis and hardware evaluation together take approx. 40 min for EfficientNet-B0. However,

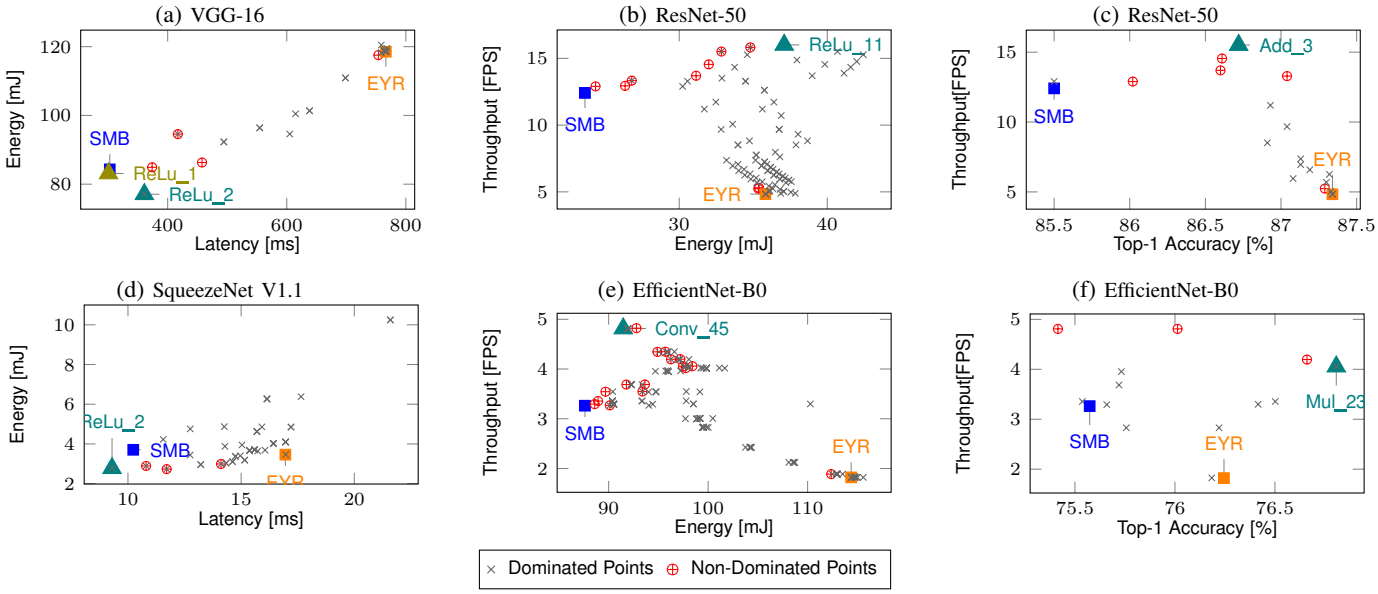

Fig. 2: Selected exploration results for a system consisting of an Eyeriss-like (EYR) accelerator in platform A and a Simba-like (SMB) accelerator in platform B linked via Gigabit Ethernet. The orange and blue squares mark the cases in which the inference is performed either completely on platform A or B, triangles highlight beneficial solutions.

quantization to 16 or 8-bit usually only leads to a slight loss of accuracy, which is why retraining several epochs is often not necessary. Nevertheless, we will show below how retraining can affect the partitioning decision for ResNet-50 and EfficientNet-B0. As shown in Figure 2(a) for VGG-16, partitioning the inference can lead to lower overall energy consumption of the system if the second ReLu is chosen as the partitioning point. Furthermore, it is even possible to achieve lower latency and lower energy consumption when partitioning the inference at *ReLu_1* compared to running the entire CNN on SMB. Similar results can be obtained for SqueezeNet V1.1, as shown in Figure 2(d). In this case, energy consumption and latency are significantly reduced by choosing *ReLu_2* as the partitioning point. In terms of throughput, as expected, the use of two accelerators allows higher values to be achieved when pipelining is used. For ResNet-50, as can be seen in Figure 2(b), the highest possible throughput is achieved with *ReLu_11*, which requires only slightly more energy than if the entire inference were performed on EYR. An increase in throughput of 29% is achieved. A significantly larger increase of 47.5% can be observed for EfficientNet-B0 at the partitioning point *Conv_45* (see Figure 2(e)). This point also offers only marginally higher energy consumption compared to the pure execution of the inference on SMB. However, the results of the evaluation also show that the throughput can drop significantly if the partitioning point is not chosen carefully. This shows the advantages of our approach over AxoNN [10] and CNNParted [9], which do not explicitly include throughput in their search. The two accelerators EYR and SMB differ in particular in the bit width, which can have an influence on the accuracy of DNNs. For this reason, this

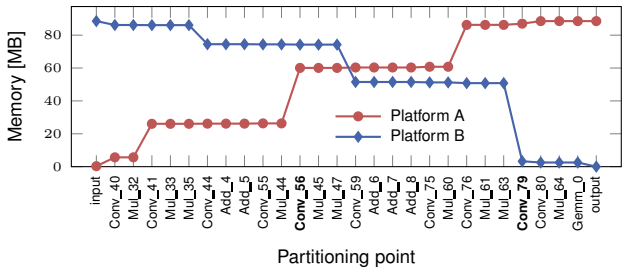

Fig. 3: EfficientNet-B0 results of the analysis of memory resources for a system consisting of two 16-bit platform architectures A and B.

metric must also be taken into account during evaluation of inference partitioning. In this regard, the results for ResNet-50 (Figure 2(c)) and EfficientNet-B0 (Figure 2(f)) show that, as expected, partitioning leads to an improvement in top-1 accuracy compared to running the network entirely on SMB. As a guideline, the later the partitioning of the network is performed, i.e. the more layers are executed on EYR, the higher the top-1 accuracy. However, especially in the case of ResNet-50, the throughput suffers from a later partitioning point as shown in Figure 2(b). For this reason, it is necessary at this point to carefully balance the two metrics depending on the use case.

Finally, Figure 3 shows the required memory size for executing partitioned DNN inference on two 16-bit hardware accelerators A and B. When compared to other DNNs where the overall memory utilization is dominated by the first layers, the memory size required for EfficientNet-B0 increases the later the partitioning in the DNN is performed. In this case,

Model	1 Partition	2 Partitions	3 Partitions	4 Partitions
SqueezeNet V1.1	1	5	7	1
VGG-16	2	8	8	2
GoogLeNet	2	14	8	2
ResNet-50	2	10	10	5
RegNetX-400MF	2	6	12	13
EfficientNet-B0	2	11	18	19

TABLE II: Number of Partitions for inference that near-optimal schedules include for a system consisting of four accelerators connected via Gigabit Ethernet

it is preferable to select a layer before *Conv_56* or after *Conv_79* in order to reduce the required system memory and thus the required die area. Consequently, the results show the importance of evaluating memory consumption for DNN inference partitioning in embedded systems.

C. Increasing Number of Partitioning Points

In certain areas, such as the automotive sector, the embedded system may have additional computing platforms between the sensor nodes and the central unit, such as the zonal gateway. These can also be equipped with a DNN accelerator. As a result, we also evaluate the impact on using more than a single partitioning point. For our evaluation, we assume a system consisting of two EYR-based platforms in the beginning and two SMB-based platforms at the end of the chain of platforms, connected each via Gigabit Ethernet. We configure our framework to find pareto-optimal points regarding overall latency, energy consumption and link bandwidth. The results of our exploration are shown in Table II.

Especially in small DNNs, the use of all available accelerators for inference does not seem to have any practical benefit. The reason for this is the high transmission costs, which lead to poorer latency and lower energy consumption. Larger DNNs such as RegNetX-400MF and EfficientNet-B0, on the other hand, benefit from a system architecture consisting of more platforms, as a significantly higher throughput can be achieved. These findings demonstrate again that, in contrast to the approach and the experimental results presented by the authors of AxoNN [10], the consideration of other metrics in addition to latency and energy consumption is essential to find a performant and efficient inference partitioning of DNNs in distributed embedded systems.

VI. Conclusion

Efficiently mapping DNN-based applications onto distributed embedded systems is a complex task that requires the evaluation of several important performance metrics. To address this problem, in this work we have presented an automated design space exploration framework for hardware-aware inference partitioning of DNNs in distributed embedded systems. The framework takes into account several system constraints to automatically determine a near-optimal schedule for a given DNN. Our experimental results prove the effectiveness of the proposed framework by revealing multiple beneficial partitioning points besides running inference on a single accelerator for different models. Furthermore, we were able to show that partitioning the DNN inference over more than two accelerators can be useful, especially for large DNN architectures. As a result, our results emphasize the importance of holistic hardware/software code design to enable efficient inference in systems with multiple accelerators.

Acknowledgment

This work was funded by the German Federal Ministry of Education and Research (BMBF) under grant number 16ME0817 (CeCaS). The responsibility for the content of this publication lies with the authors.

References

[1] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, 2018.

[2] T. Alonso *et al.*, "Elastic-DF: Scaling performance of dnn inference in fpga clouds through automatic partitioning," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 2, dec 2021.

[3] D. R. Agut, R. Tornero, and J. Flich, "Towards efficient neural network model parallelism on multi-fpga platforms," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.

[4] A. K. Kakolyris, M. Katsaragakis, D. Masouros, and D. Soudris, "RoaD-RuNNer: Collaborative dnn partitioning and offloading on heterogeneous edge systems," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.

[5] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019.

[6] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," ser. ASPLOS '17, 2017.

[7] S. Yao *et al.*, "Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, ser. SenSys '20, 2020.

[8] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2018.

[9] F. Kreß *et al.*, "CNNParted: An open source framework for efficient convolutional neural network inference partitioning in embedded systems," *Computer Networks*, vol. 229, 2023.

[10] I. Dagli, A. Cieslewicz, J. McClurg, and M. E. Belviranli, "AxoNN: Energy-aware execution of neural network inference on multi-accelerator heterogeneous socs," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22, 2022.

[11] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning," *Sustainable Computing: Informatics and Systems*, vol. 38, apr 2023.

[12] A. Parashar *et al.*, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019.

[13] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

[14] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, 2020.

[15] Y. S. Shao *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52, 2019.

[16] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, 2019.