Genetic Algorithm-based Framework for Layer-Fused Scheduling of Multiple DNNs on Multi-core Systems

Sebastian Karl*†, Arne Symons*§, Nael Fasfous‡, Marian Verhelst§
*These authors contributed equally, †TU Munich, Germany §KU Leuven, Belgium ‡BMW AG, Germany
sebastian.karl@tum.de, {arne.symons, marian.verhelst}@kuleuven.be, nael.fasfous@bmw.de

***Abstract*—Heterogeneous multi-core architectures are becoming a popular design choice to accelerate the inference of modern deep neural networks (DNNs). This trend allows for more flexible mappings onto the cores, but shifts the challenge to keeping all cores busy due to limited network parallelism. To this extent, layer-fused processing, where several layers are mapped simultaneously to an architecture and executed in a depth-first fashion, has shown promising opportunities to maximize core utilization. However, SotA mapping frameworks fail to efficiently map layer-fused DNNs onto heterogeneous multi-core architectures due to ignoring 1.) on-chip weight traffic and 2.) inter-core communication congestion. This work tackles these shortcomings by introducing a weight memory manager (WMM), which manages the weights present in a core and models the cost of re-fetching weights. Secondly, the inter-core communication (ICC) of feature data is modeled through a limited-bandwidth bus, and optimized through a contention-aware scheduler (CAS). Relying on these models, a genetic algorithm is developed to optimally schedule different DNN layers across the different cores. The impact of our enhanced modeling, core allocation and scheduling capabilities is shown in several experiments and demonstrates a decrease of 52% resp. 38% in latency, resp. energy when mapping a multi-DNN inference, consisting of ResNet-18, MobileNet-V2 and Tiny YOLO V2, on a heterogeneous multi-core platform compared to iso-area homogeneous architectures.**

***Index Terms*—deep learning accelerators, layer fusion, heterogeneous multi-core, genetic algorithm**

I. Introduction

In recent years, deep neural networks (DNNs) have become more complex, with multi-DNN workloads becoming an attractive solution to tackle complex applications [7], [8], [13]. On the other side, heterogeneous multi-core processing platforms are appearing to accommodate these new DNN workloads [8], [12], [13]. Traditional processing approaches execute the workload in a layer-by-layer manner, which has a high memory footprint and can only to limited extent exploit the inherent inter-layer parallelism of the DNN. For applications at the edge, where low latency at high energy efficiency is vital, the multi-core parallelism can be further exploited through a new processing paradigm, called layer fusion [13]. Here, a stack of layers is fused and executed in a depth-first manner, as opposed to a layer-by-layer manner. Finding optimal intra-core mappings has been extensively studied, but the end-to-end hardware overhead of mapping layer-fused processing onto multi-core architectures is lacking.

Most SotA mapping frameworks focus either on layer-fused single-core mappings [2] or they only allow to execute a workload in a layer-by-layer fashion on multi-core systems [4], [7], [8]. As a result, SotA frameworks are not capable of mapping multi-DNN workloads on heterogeneous multi-core platforms in a layer-fused manner. While in layer-by-layer processing the inter-core communication and the weight traffic play a minor role, this additional communication overhead has to be considered for layer-fused processing.

This paper proposes a multi-core mapping framework, which is capable of finding optimal layer-fused mappings for a multi-DNN workload. Moreover, the multi-core architecture presents an additional challenge to allocate the layers to an optimal core. For this, a genetic algorithm (GA) is implemented and deployed. The key contributions of this paper are:

- A multi-core mapping framework with an analytic cost model for layer-fused processing and a genetic algorithm-based core allocation (Sec. III).
- A weight memory manager (WMM), responsible for the weight fetching of the different cores and a contention-aware scheduler (CAS) to handle the inter-core communication (ICC) through a limited bandwidth bus (Sec. III-C).
- The impact of the ICC and the importance of automated core allocation are proved empirically and the framework is exploited to evaluate the best hardware architecture for optimal deployment of a multi-DNN workload (Sec. IV).

The framework is included in Stream [11] and open-sourced on GitHub (https://github.com/ZigZag-Project/stream).

II. Background

DNN HW accelerators: Besides GPUs and FPGAs, custom AI accelerator chips are commonly used to process DNN workloads. Such accelerators are especially popular for applications on the edge due to their higher energy efficiency and/or low latency. Typically, a large MAC array is utilized to exploit the different degrees of parallelism of the DNN layers. The spatial parallelization strategy defines which loop dimensions are spatially unrolled onto the MAC units, while the temporal unrolling defines the temporal execution order and stationary data [9], [13]. This dataflow varies from one implementation to another, and is key in achieving good execution efficiency. Single-core accelerators can either support a fixed spatial dataflow (FDA) or be reconfigurable (RDA). In multi-core accelerators, each core is typically a FDA, while the different cores can provide flexibility through the layer-core allocation [8]. The

978-3-9819263-7-8/DATE23/© 2023 EDAA

challenge is, however, to determine the best hardware mapping of the workloads across the cores, to maximize the hardware utilization and hence system efficiency [13].

DNN mappings: Two different subproblems are distinguished when mapping a (multi-)DNN workload onto a multi-core architecture [8]. Firstly, the different layers of a DNN workload have to be assigned to the available cores. This is referred to as (layer-)core allocation. Secondly, each layer has to be optimally mapped on the core to which it is allocated, referred to as intra-core mapping.

The intra-core mapping is defined through the spatial parallelization strategy and the temporal loop order. Efficiently mapping a DNN layer onto a MAC array has been extensively covered by previous SotA, with multiple frameworks allowing to optimize such intra-core mapping [9], [10].

Layer fusion: Classically, DNN workloads are mapped in a layer-by-layer fashion, which means that the entire layer is executed before moving to a subsequent layer(s). Under modern DNN workloads, this scheme suffers from large memory buffering footprints due to large activation sizes. This degrades hardware performance for edge applications as on-chip memory is typically limited, hence requiring frequent expensive off-chip accesses. Recent research tries to overcome this problem by utilizing fine-grained layer fusion [2], [13]. Under this paradigm, multiple layers are fused together, computing part of a layer's activations and propagating this to the subsequent layer(s) in a depth-first manner. Research has, to a limited extent, shown the performance benefits that this paradigm can bring due to higher hardware parallelism and lower memory buffering footprint [13]. However, an accurate and fast mapping framework for layer-fused mapping onto multi-core architectures is missing.

Mapping frameworks: Table I shows relevant DNN mapping frameworks and their capabilities. All of them include an analytical cost model to rapidly estimate hardware performance of a mapping (in terms of energy, latency and/or memory usage). ZigZag [9] includes an accurate latency model, allowing for multi-port memories with limited bandwidth, to estimate the layer-by-layer performance on a single-core architecture. Planaria [4] extends this exploration to RDAs by adaptively dividing the MAC array into a multi-core system. In order to mitigate reconfiguration overhead, Herald [8] and MAGMA [7] explored heterogeneous dataflow accelerators (HDA) for layer-by-layer processing. Olympus [2] and DNNFuser [6] shifted the focus to layer-fused processing exploration, but are restricted to a single-core architecture. While these frameworks have shown the advantages and disadvantages of multi-core architectures and layer-fused processing separately, an accurate and fast exploration framework for layer-fused processing on multi-core architectures is lacking.

III. Methodology

This section introduces the structure of the proposed multi-core layer-fused DNN mapping framework in III-A. Subsequently, the key innovations of the framework are discussed in detail: 1.) layer fusion through line-based tiling (III-B), 2.) communication overhead modeling (III-C) and 3.) optimal core allocations through a genetic algorithm (GA) (III-D).

TABLE I: Comparison mapping frameworks.

Framework	Analytic Cost Model	Multi-Core Capabilities	Focus on HDAs	Layer Fusion
ZigZag [9]	✓	✗	✗	✗
Olympus [2]	✓	✗	✗	✓
DNNFuser [6]	✓	✗	✗	✓
Planaria [4]	✓	✓	✗	✗
Herald [8]	✓	✓	✓	✗
MAGMA [7]	✓	✓	✓	✗
This work	✓	✓	✓	✓

A. Structure of Framework

The general structure of the implemented mapping and exploration framework can be divided into three steps, as shown in Fig. 1.

1) User inputs: The DNN workload description and hardware architecture are provided to the framework as inputs. Each workload layer contains loop dimension sizes, the stride and computational bit precision. By providing a graph with unconnected subgraphs, multi-DNN workloads are flexibly supported by the framework. The hardware architecture contains the different processing cores, each with their spatial dataflow and memory hierarchy. A communication bus with fixed bandwidth is modeled for the ICC.

2) Cost evaluation of workload graph: In this step, a workload graph is created and the costs for its nodes are evaluated. For this, each layer is divided into smaller tiles by applying a line-based tiling approach (see III-B1). Such a tile is called a computation node (CPN) in the remainder of this paper. The data dependencies of the different CPNs are extracted. For each CPN, the energy and latency cost of executing that node on each compatible core of the system are extracted with the intra-core mapping framework ZigZag [9] (see III-B2). The workload graph includes communication nodes (CMNs) to take into account the data exchange between the processing cores (see III-C1). The WMM keeps track of the energy and latency caused by off-chip memory accesses (see III-C2).

3) Core allocation with genetic algorithm: Next, the CPNs of the different layers have to be assigned to one of the compatible processing cores in the architecture. For this, a GA is utilized in order to find Pareto optimal core allocations of the workload in the vast allocation space. The GA performs a fitness evaluation, which can be based on any combination of latency, energy and on-chip activation buffer requirements. The final output of the GA is a 3D Pareto front of core allocations. The three dimensions of the Pareto front are latency, energy and on-chip activation memory requirements. Besides these metrics, a report about other metrics such as core utilization can be obtained for each design point in the Pareto front. A timeline of

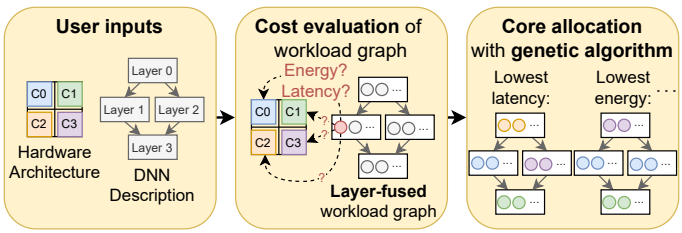


Fig. 1: Overview of implemented framework.

the temporal execution schedule of the nodes in the workload graph is generated, together with the instantaneous memory utilization of the on-chip weight and activation buffers. More details about the GA are provided in III-D.

B. Layer Fusion through Line-based Tiling

1) General Idea: As mentioned earlier, the idea of layer fusion is to split the computation of a DNN's layers into smaller parts. There are many different ways to introduce layer fusion through tiling. One of these tiling strategies is called line-based tiling and it will be used in this work since it has shown promising results in practice [5]. Fig. 2 summarizes which loop dimensions of a layer are captured within a CPN when line-based tiling is applied. Here, each CPN includes one entire line of the output feature map of a layer. This means that all loop dimensions except the activation row dimension (OY) are included in the CPN.

Fig. 3a shows an example for the processing of two subsequent layers with line-based tiling. In the layer-by-layer case (top), the processing of layer 0 can only start when processing of layer 1 is complete. In this scenario, the cores need to store the entire layer's activations and the ICC is infrequent and long. This picture changes when introducing tiling through multiple CPNs per layer. In this example, a one-to-one dependency is assumed between CPNs of layer 0 and layer 1, as is the case in e.g. a point-wise convolution. More processing parallelism between the layers is possible and the amount of data generated, stored and communicated by one CPN is much smaller, leading to frequent and short ICC.

One could be tempted to simply generate many CPNs for each layer in order to maximize the parallelism in the processing and to minimize the output size of each CPN. Yet, this idea leads to a limitation of layer fusion, which is shown in Fig. 3b. The computational overhead increases significantly when splitting the output feature map into tiny parts, due to the reduction in data reuse and parallelization opportunities stemming from the smaller loop dimensions inside a CPN. Another reason for this performance degradation is that the on and offloading of data (input activation and/or weights) becomes more dominant for small CPNs.

2) Cost Evaluation of CPNs: The costs of a CPN are mainly caused by the data movement between the on-chip memories together with the actual computation cost. In order to model the costs, the ZigZag framework is utilized [9]. It uses an analytic cost model to determine the costs of a certain intra-core mapping. Our framework selects the mapping with the lowest energy delay product (EDP). A mapping for each CPN is chosen for each accelerator core in the system. As all the CPNs from the same layer have the same characteristics, their costs are the same. These costs are saved as an attribute for each CPN in the workload graph, to be used for later scheduling.

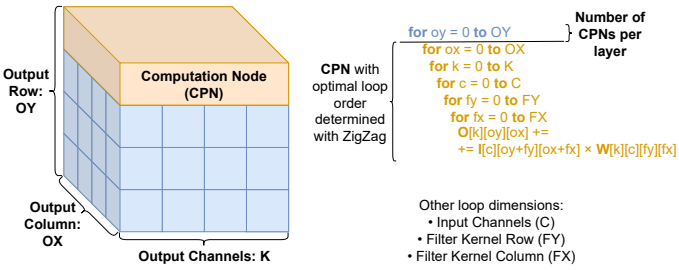
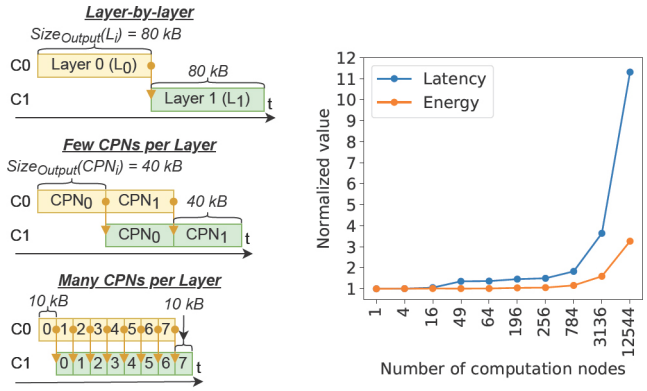


Fig. 2: Output feature map and loop dimensions of a computation node in the context of a DNN layer when line-based tiling is applied.



(a) Computation nodes in layers. (b) Computational overhead.

Fig. 3: (a) Parallel processing capabilities when introducing layer fusion. (b) Resulting computational overhead.

When two or more CPNs of the same layer are executed subsequently, the layer's weights and part of the activations are already present within the core. This leads to more efficient processing, effectively creating a bigger tile with more reuse opportunities. The tool takes these effects into account and determines the correct energy and latency when scheduling multiple CPNs of the same layer.

C. Off-Chip Data Movement

Fig. 4 shows a simplified workload graph utilized in the third step of the framework. Its main components are CPNs and communication nodes (CMNs) which are connected with edges in order to represent data dependencies. III-B2 explained how ZigZag is used to estimate the energy and latency of the CPNs in the workload graph. The CMNs are required in order to take care of the ICC, which occurs when data moves between cores. III-C1 discusses how the energy and latency of the CMNs are determined and how they are scheduled with a contention-aware scheduler (CAS). III-C2 elaborates how accesses to the off-chip DRAM are handled through the WWM.

1) Inter-core Communication (ICC): The output activation of a CPN has to be moved to another core in the system in case two dependent CPNs are not executed on the same core. In this case additional energy and latency are caused

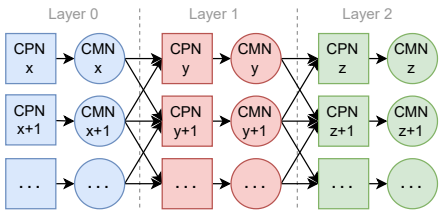


Fig. 4: Simplified workload graph with computation nodes and communication nodes of different layers.

by the inter-core data movement. In order to model the data exchange between the cores, a communication bus is modeled in the hardware architecture. While the processing cores take care of the execution of the CPNs in the workload graph, the communication bus is responsible for executing the CMNs which are inserted between the CPNs. The latency of each CMN is determined by

$$Runtime_{cc} = \frac{Size_{output}(CPN_i)}{BW_{Bus}} \quad (1)$$

where $Size_{output}(CPN_i)$ is the size of the output activation of the producer CPN in $bits$. BW_{Bus} is the bandwidth of the communication bus in $\frac{bits}{cc}$. The communication energy is determined through:

$$Energy = Size_{output}(CPN_i) \times E_{Bus} \quad (2)$$

where E_{Bus} is the energy of a bus transfer in $\frac{J}{bit}$.

Fig. 5 shows a simplified example to illustrate how a subset of the CPNs and CMNs from Fig. 4 are scheduled on a two core system with a communication bus. The activation buffer requirements can be calculated based on the schedule of the CPNs and CMNs. While the data is moved into the memory of a specific core (e.g. time step a to b in Fig. 5), the corresponding space in core 0's activation memory is already allocated. The actual processing of CPN y can only start as soon as all relevant data is loaded (time step b). During the processing of CPN y (b to c) the CPN's input activation as well as the output activation are kept in core 0's memory. Input activations can be discarded as soon as they're no longer required for future computations (time step c). Output activations are kept in memory until they have been transferred to the next core (time step d). The scheduling as described here is managed by the CAS.

2) Off-chip DRAM Accesses: Before a CPN can be processed, its required weights have to be present in the memory of its processing core. The weight memory manager (WMM) keeps track of which weights are present in each core and which weights are to be loaded and discarded. The weights typically have to be fetched from off-chip DRAM if they are not present in the on-chip memory. In such a case, additional costs (latency and energy) are caused by the off-chip memory accesses through the DRAM port resource.

Fig. 6 shows how the weights for CPN x are fetched from the off-chip memory through the DRAM port (time step a to b). The CPNs in this example are from different layers (see Fig. 4) and therefore, they do not share any weights. The actual

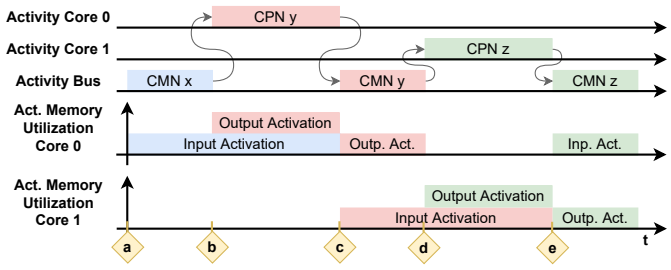


Fig. 5: Sequence chart for data exchange between two accelerator cores through a communication bus.

processing of the CPN x happens from b to c while the weights of the next CPN y are loaded. If the weight fetching can be covered completely with the computation of the CPN x, then processing of the next CPN y can start immediately (c). If the weight fetching takes longer than the processing of the previous CPN, a gap in the core activity occurs (d to e). If the weight memory would overflow when loading new weights, the WMM removes the oldest layer's weights first in a FIFO manner.

The latency of the weight loading operation can be calculated with

$$Runtime_{cc} = \frac{Size_{weight}(CPN_i)}{BW_{DRAM}} \quad (3)$$

where CPN_i is the CPN associated with the weight loading. The $Size_{weight}$ is the size of the loaded weights in $bits$ and the BW_{DRAM} is off-chip memory bandwidth of the DRAM port in $\frac{bits}{cc}$. The required energy can be determined with

$$Energy = Size_{weight}(CPN_i) \times E_{DRAM} \quad (4)$$

where E_{DRAM} is the off-chip access energy in $\frac{J}{bit}$.

D. Genetic Algorithm for Core Allocation

The GA holds a population of individuals and every individual of a generation holds a genome with several genes. The best individuals of a population are selected before crossover and mutation are performed. An iteration of these steps is called a generation of a GA. The GA finds gradually better solutions by evolving its population over several generations and escaping local minima [3]. The overall objective of the GA is to find Pareto optimal core allocations based on different optimization criteria [3]. These optimization criteria can be any combination of the energy, latency and on-chip activation buffer size. Fig. 7 gives an overview about the general structure of the utilized GA. Its implementation details are explained next.

Encoding: Each gene in the genome of an individual represents one core-layer allocation. Meaning that the integer value of the i^{th} gene describes to which core the CPNs of the i^{th}-layer are mapped to. The cardinality of the solution space for the genetic algorithm is equal to m^n as a result of the chosen encoding with m cores in the system and n layers in the workload.

Fitness Evaluation: The fitness evaluation of an individual is based on the best schedule compatible with the individual's core allocation of the workload graph. The schedule's energy can be determined by summing up all the energy values of the CPNs, CMNs and accesses to the DRAM. In order to get the latency and the on-chip activation buffer requirements, the workload graph is temporally mapped to determine the start and end time of all the nodes in the graph.

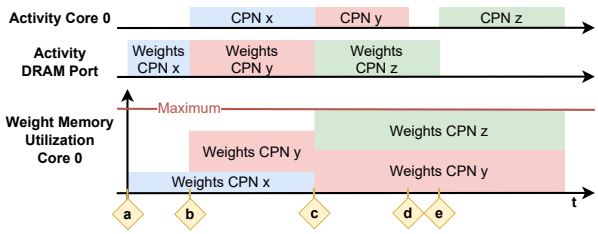


Fig. 6: Sequence chart for weight fetching through DRAM port.

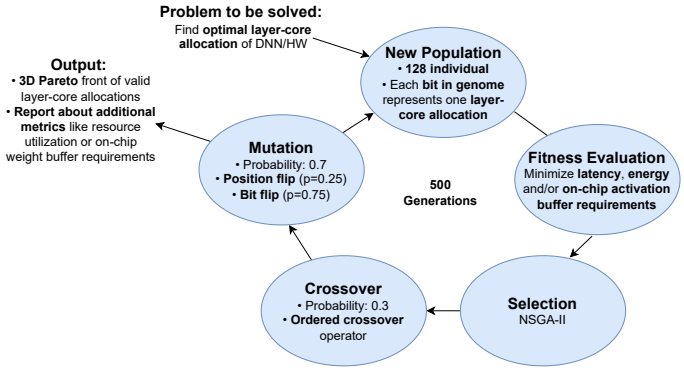


Fig. 7: Overview of the used genetic algorithm.

Selection, Crossover and Mutation: The surviving individuals of a generation are chosen through a NSGA-II selection procedure [3]. The probabilities for crossover and mutation were determined through a grid search in order to ensure a fast convergence. Crossover is performed through an Ordered Crossover operation. If the genome of an individual should be mutated, then either a bit flip mutation or a position flip mutation is done.

IV. Experiments

In this section several experiments are performed to demonstrate the opportunities the proposed framework brings. At first the importance of core-to-core communication modeling is shown by varying the bandwidth of the ICC bus (see IV-A). Next, the impact of the GA-based core allocation is outlined by comparing its results to a naive greedy allocation (see IV-B). Our last experiment highlights the flexibility of the framework to a broad variety of single core and multi-core hardware architectures, and different DNN workloads (see IV-C).

All experiments include all convolutional and pooling layers of the targeted DNNs. Since the paper mainly aims for edge applications, the batch size is set to 1. The reported scatter plots are 2D projections of the 3D Pareto front obtained from the GA, where each design point represents one specific layer-core allocation.

A. Variation of Communication Bus Width

In this first experiment a heterogeneous quad-core architecture is used (see Fig. 8). Each core has a predefined spatial dataflow, depicted inside the cores. An additional pooling core handles the pooling layers. Each core has an on-chip memory hierarchy connected to the off-chip DRAM memory through

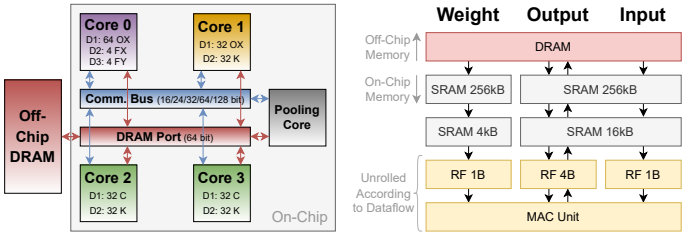


Fig. 8: Heterogeneous quad-core architecture (left) and memory hierarchy in each core (right) for communication bus bandwidth variation experiment.

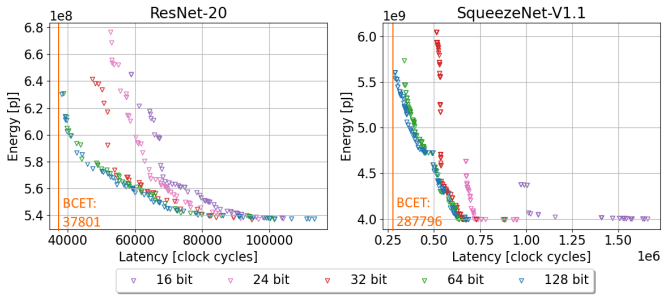


Fig. 9: Influence of the communication bus width on the energy and latency.

a DRAM port. The required access energies for the different memories are obtained through CACTI 7 [1]. The different cores are connected through a communication bus, which has a certain width. The experiment sweeps the bus' bandwidth from 16 bits up to 128 bits/cc, to show the impact of correctly modeling bus congestion on the system's energy and latency.

Fig. 9 shows the results of this experiment for ResNet-20 and SqueezeNet-V1.1. Each design point in the plot is equal to one layer-core allocation and every marker style corresponds to a certain bus width. For both networks, the achievable latency and energy differs significantly when considering a limited communication bus bandwidth. At infinite bandwidth, the best-case execution time (BCET), i.e. the minimal achievable latency, is shown. A latency degradation of 20%, resp. 44% can be observed for ResNet-20, resp. SqueezeNet-V1.1 for the **best** latency point of the 32 bit architecture. This shows the importance of taking the bus traffic into into account since it can cause significant performance degradation.

B. Greedy versus Genetic Algorithm based Core Allocation

This experiment demonstrates the impact of the framework's capability to autonomously optimize the layer-core allocation. As a baseline, SqueezeNet-V1.1 is mapped onto the architecture of Fig. 8 in a naive greedy allocation scheme, and compared with our automatic GA-based scheme. The greedy allocation assigns CPNs of a layer to the core with the best EDP performance. This allocation is compared to three different outcomes of the GA-based allocation: the energy, latency and activation memory leaders. Table II summarizes the relative performance of these three GA outcomes relative to the baseline (greedy allocation). It is clear that the GA outperforms the greedy allocation as it is able dynamically overcome local minima in the optimization space. A decrease of 62% resp. 54% can be seen for the latency resp. energy when comparing the corresponding metric leader to the base line. It is also possible to pick another design point of the Pareto front somewhere between the metric leaders which still ensures that they offer a unique trade-off due to the usage of NSGA-II selection [3].

TABLE II: Relative performance of genetic algorithm-based core allocation compared to greedy allocation.

Metric	Latency Leader	Memory Leader	Energy Leader
Latency	**- 62%**	- 26%	- 12%
Activation Memory Requirements	+ 42%	**- 48%**	- 19%
Energy	- 36%	- 49%	**- 54%**

C. Hardware Architecture Exploration

In this last and extensive experiment, three different homogeneous quad-core architectures as well as three different single-core architectures are compared against the previously mentioned heterogeneous multi-core system, to demonstrate the flexibility of the framework. All seven architectures have the same area footprint as they contain the same amount of MAC units and on-chip memory. The memory hierarchies are specialized to support the spatial unrolling of the different cores and the characteristics of the communication bus, the DRAM and the pooling core are the same in all systems. Each of the homogeneous multi-core systems holds all four cores with the same spatial unrolling (i.e. $C|K$, $OX|K$ or $OX|FX|FY$). As shown in Fig. 8, each core of the heterogenous quad-core system holds different spatial unrollings. The single-core architectures have one big core with one of these three unrollings. For this experiment, the HW architectures are slightly adapted. Now each core has its own DRAM port. If the weights of a layer do not fit into the on-chip memory, then the weight fetching is included in the cost estimation with ZigZag.

In order to see the influence of the architectural design decision of the different systems, the mappings of SqueezeNet-V1.1 as well as a multi-DNN workload are evaluated. ResNet-18, MobileNet-V2 and Tiny YOLO V2 are jointly executed in the multi-DNN workload. Especially the multi-DNN workload has a huge solution space for the quad-core architectures since 80 layers have to be freely mapped to the four cores of the systems. As a result, there are $4^{80} = 1.46 \times 10^{48}$ possible layer-core allocations. Sec. IV-B already outlined how important it is to have an automated core allocation algorithm to find optimal solutions. This experiment also draws a comparison to a layer-by-layer processing of a workload, which is commonly done in other SotA multi-core mapping frameworks [7], [8]. For the layer-by-layer mappings, the data is always moved back to the off-chip DRAM because in most cases the activation of a layer cannot stay on chip since the on-chip memories are too small.

Fig. 10 shows a plot of the design points (core-allocations) of the different architectures. The latency leader of the heterogneous system has an 8% lower latency and a 38% lower energy for the multi-DNN workload when comparing to the latency leader of the homogeneous quadcore with $OX|K$ spatial unrolling. The energy leader of the heterogeneous system has a 52% lower latency and a 14% lower energy when comparing to the energy leader of homogeneous quadcore ($C|K$). The lower performance of the homogenoeus systems is a result of only employing one type of spatial unrolling. This experiment clearly outlines the higher performance of the layer-fused mappings compared to the layer-by-layer mappings. The layer-by-layer mappings suffer from limited processing parallelism and a significantly higher energy and latency due to more off-chip memory accesses. As a result, the best latency for the heterogeneous system is 61% lower and the best energy is 83% lower when processing the multi-DNN workload in a layer-fused way instead of layer-by-layer. This experiments clearly shows the benefits of heterogeneous multi-core architectures and the opportunities provided by layer fusion.

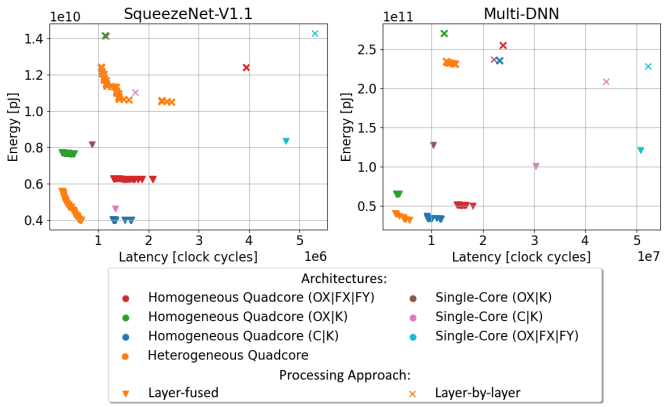


Fig. 10: Pareto plot of latency/energy costs for different architectures and processing approaches.

V. Conclusion

This work introduced a framework for accurate hardware cost estimation of multi-DNN layer-fused mappings on homogeneous and heterogeneous multi-core architectures. A limited bandwidth communication bus is used to accurately assess the inter-core communication impact and the off-chip accesses caused through the layer-fused paradigm are managed through a memory manager. Moreover, a genetic algorithm is implemented and deployed to find the optimal layer-core allocations. Several experiments have shown the importance of the communication bus and off-chip access modeling, as well as the importance of the automatic genetic algorithm-based layer-core allocation. Finally, the framework was exploited to conduct a hardware architecture exploration, showing that a heterogeneous system can achieve a 38% lower energy and a 52% lower latency for a multi-DNN workload compared to iso-area homogeneous architectures.

References

[1] R. Balasubramonian *et al.*, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM TACO*, vol. 14, no. 2, 2017.

[2] X. Cai *et al.*, "Olympus: Reaching memory-optimality on dnn processors," *IEEE TC*, 2021.

[3] N. Fasfous *et al.*, "Anaconga: Analytical hw-cnn co-design using nested genetic algorithms," in *DATE*, 2022.

[4] S. Ghodrati *et al.*, "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *MICRO*, 2020.

[5] K. Goetschalckx and M. Verhelst, "Depfin: A 12nm, 3.8tops depth-first cnn processor for high res. image processing," in *VLSI*, 2021.

[6] S.-C. Kao *et al.*, "Dnnfuser: Generative pre-trained transformer as a generalized mapper for layer fusion in DNN accelerators," *CoRR*, vol. abs/2201.11218, 2022.

[7] S.-C. Kao and T. Krishna, "Magma: An optimization framework for mapping multiple dnns on multiple accelerator cores," in *IEEE HPCA*, 2022.

[8] H. Kwon *et al.*, "Heterogeneous dataflow accelerators for multi-dnn workloads," in *IEEE HPCA*, 2021.

[9] L. Mei *et al.*, "Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators," *IEEE TC*, 2021.

[10] A. Parashar *et al.*, "Timeloop: A systematic approach to dnn accelerator evaluation," in *IEEE ISPASS*, 2019.

[11] A. Symons *et al.*, "Towards heterogeneous multi-core accelerators exploiting fine-grained scheduling of layer-fused deep neural networks," 2022. [Online]. Available: https://arxiv.org/abs/2212.10612

[12] K. Ueyoshi *et al.*, "Diana: An end-to-end energy-efficient digital and analog hybrid neural network soc," in *IEEE ISSCC*, vol. 65, 2022.

[13] M. Verhelst *et al.*, "Ml processors are going multi-core: A performance dream or a scheduling nightmare?" *IEEE SSC-M*, vol. 14, no. 4, 2022.