D2.1 - Hardware accelerator architecture report

Efficient implementation of Deep Neural Networks (DNNs) in hardware requires rigorous exploration of the design space on different layers of abstraction, including algorithmic, architectural, and platform layers. At the highest level of the design hierarchy is the algorithm, which is the most abstract description of the data and control flow in the form of a DNN topology. The architecture layer maps the topology to a hardware design, which is implemented on the platform. At the lowest level is the platform, which describes the hardware and its physical properties.

As the DNN topology is on the highest level of the design hierarchy, the changes applied on the topology have potentially the highest impact on the properties of the final implementation. DNNs have to be designed to achieve the required accuracy and fulfill hardware criteria, especially on edge devices because they have small memory, constrained computing capabilities, memory bandwidth, power, and energy budgets. The analysis shows that arithmetic operations are ”cheap” while memory accesses are ”expensive” and have a cost that is a function of the size of the memory being accessed. The relative energy cost should be used as the main guidance for designing DNNs. The hardware-aware DNNs have to be small to fit into on-chip memory ideally, or to mitigate any communication with the external memory, they have to use fewer operations and leverage low-precision data types.

There is a lot of ongoing research on developing networks with lower computation costs and storage consumption without impairing classification accuracy. The efficient models became possible due to multiple macro- and micro-architectural improvements of the models. The types of layers and their arrangement are referred to as macro-architecture. The efficient micro-architectural approaches are: a) very deep models are replaced with fewer layers, but with more channels, b) activation feature maps are kept smaller, c) models are enhanced with skip and residual connections that have been proven to improve accuracy, d) standard convolutions are replaced with depth-wise separable ones. The micro-architecture also defines methods applied to individual layers, like replacing big convolutional kernels with smaller ones and fusing different layers. Further techniques have been proposed to alleviate the computing and storage challenges. Among the most common ones are distillation [6], pruning [7, 8, 9], and quantization or a combination thereof [10]. The resulting model after knowledge distillation does not require special treatment during inference, unlike after pruning and quantization. In summary, pruning and quantization are the primary model compression techniques, but they require special treatment which raises the question of selecting an implementation platform that can fully benefit from them.

Comparing different hardware platforms: Central Processing Unit (CPU), Graphics Processing Unit (GPU), Field-Programmable Gate Array (FPGA) and Application-Specific Integrated

Circuits (ASICs), one can see that these options present a trade-off between flexibility and efficiency. While CPUs, GPUs are highly flexible, they cannot fully benefit from the major optimization techniques, like pruning and quantization. Only ASICs and FPGAs can fully benefit from them. The main reasons why FPGAs and ASICs are highly efficient in comparison to general-purpose computing platforms are: a) optimized memory, b) data specialization, c) massive parallelism, d) reduced overhead, e) algorithm-architecture co-design. Using these advantages, one can use FPGAs to provide ad hoc solutions to facilitate computationally intensive, time-critical tasks at low-power consumption in a reprogrammable manner, unlike ASICs. In summary, FPGA is a computing platform with a unique combination of programmability, development cost, and efficiency that can fully benefit from various compression techniques. How can one use these features to implement optimized DNNs efficiently?

The design of DNN topologies, custom hardware architectures for DNNs, and their implementation on FPGA is a time-consuming process. To efficiently implement DNNs on a specific FPGA platform and to meet certain requirements, e.g., power consumption and latency, we have to consider an enormous amount of design parameters starting from neural topology down to hardware architecture and physical implementation. Importantly, interdependencies between the different design layers have to be considered, making it impossible to find optimal solutions manually. Fast and efficient implementation of DNNs on FPGAs can be achieved by combining recent advances: a) co-design, meaning DNN's topologies and hardware architectures have to be co-designed by joint optimization of performance and efficiency while maintaining accuracy, which can be achieved using Neural Architecture Search (NAS), b) parameterizable hardware templates to build libraries of hardware components that support a wide range of parameters of various layers, c) facilitated hardware design using, e.g. High-level Synthesis (HLS) to accelerate the development process.

There are many techniques for automatic exploration of the vast design space of DNNs. Among the most successful approaches are a) Reinforcement Learning (RL), b) gradient-based, c) and evolutionary algorithms. Evolutionary algorithms do not require training an agent, like RL-based methods, nor a supergraph, like gradient-based methods. It was shown that the evolutionary algorithm can outrun RL-based methods and outperforms state-of-the-art hand-designed models. The proposed methodology is based on the evolutionary algorithm presented in [30]. The main techniques responsible for the high efficiency of the NAS implementation are: a) multi-objective Pareto optimization, which enables multi-criteria optimization, b) Bayesian sampling method to improve the candidate selection process, c) evaluation of the candidates using ”cheap” and ”expensive” objectives to accelerate the evaluation process. Therefore, the NAS finds fully hardware-compatible solutions that are optimal with respect to the hardware optimization objectives and fulfill the application requirements.

To enable cross-layer optimizations, we present a flexible HLS hardware library of highly customizable hardware architectures, which can facilitate various DNN topologies. The hardware library is written as a collection of C/C++ template functions with HLS annotations and modularity in mind to make it easily expandable by new layers. The hardware architecture is

designed to be low power and ultra-low latency. Primarily, this is achieved by a) keeping all weights and intermediate results in on-chip memory since off-chip transfers consume more energy and introduce extra latency, b) external memory is only used to read input data and write results, therefore reducing memory access to the absolute minimum, c) separate hardware modules dedicated to each layer are connected using streaming interfaces to facilitate fast design, debugging, interoperability, and ease of integration, d) the architecture is fully pipelined, allowing all layers to operate concurrently and starting the computation as soon as the inputs are ready to reduce latency and energy consumption.

One of the recent emerging trends to increase energy efficiency is to adapt the low-precision data formats for both inference and training of DNNs. Typically training is done using floating-point formats as they are able to provide wide dynamic range and high precision. The main challenge of using low-bit-width data formats is the limited dynamic range compared to floating-point 32-bit (FP32). One possible solution to compensate for the range limitation of low-bit-width data formats is to shift the data format's representable range to the desired location. This can be done by varying the bias value of the floating-point equation instead of using a common bias value for all values. Our methodology uses statistical analysis to find the optimum bias value for a given DNN model. By this method, we find the bias value, which shifts the low-bit-width data format dynamic range to the location with the highest coverage for a given DNN model. We demonstrate the efficiency of the approach facilitating floating-point 8-bit (FP8) format for several datasets and state-of-the-art DNN topologies. This enables a linear reduction of the total number of Dynamic Random Access Memory (DRAM) accesses, which increases energy efficiency while keeping the accuracy on par with FP32 format.

To facilitate fast implementation of topologies found by the NAS and mapped onto custom hardware architectures, we implemented the Holistic Auto machine Learning for FPGAs (HALF) framework that consists of two main components, which are the hardware-aware NAS and the FPGA implementation framework. The framework automatically produces a hardware implementation for the selected FPGA platform that fulfills the requirements. The HALF framework accelerates the design cycle significantly, it reduces the deployment time from months to days.