SustainML

Application Aware, Life-Cycle Oriented Model-Hardware Co-Design Framework for Sustainable, Energy Efficient ML Systems

Open-source hardware exploration toolchain - Initial

Deliverable D2.2

WP2 - Hardware Architectures for Low Power ML

This project has received funding from the European Union's Horizon Europe research and innovation programme (HORIZON-CL4-2021-HUMAN-01) under grant agreement No 101070408

eProsima
The Middleware Experts

DFKI

Inria

upmem

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

IBM

Project

Title:	SustainML: Application Aware, Life-Cycle Oriented Model-Hardware Co-Design Framework for Sustainable, Energy Efficient ML Systems
Acronym:	SustainML
Coordinator:	eProsima
Grant agreement ID:	101070408
Call:	HORIZON-CL4-2021-HUMAN-01
Program:	Horizon Europe
Start:	01 October 2022
Duration:	36 months
Website:	https://sustainml.eu
E-mail:	sustainml@eprosima.com
Consortium:	**eProsima (EPROS)**, Spain **DFKI**, Germany **Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau (RPTU)**, Germany **University of Copenhagen (KU)**, Denmark **National Institute for Research in Digital Science and Technology (INRIA)**, France **IBM Research GmbH**, Switzerland **UPMEM**, France

Deliverable

Number:	**D2.2**
Title:	**Open-source hardware exploration toolchain - Initial**
Month:	24
Work Package:	WP2 - Hardware Architectures for Low Power ML
Work Package leader:	RPTU
Deliverable leader:	RPTU
Deliverable type:	R, DEM
Dissemination level:	Public (PU)
Date of submission:	2024-09-30
Version:	v1.0
Status:	Finished

Version history

Version	Date	Responsible	Author/Reviewer	Comments
v1.0	30-09-2024	RPTU	RPTU	

Executive summary

SustainML project aims to develop a design framework and an associated toolkit, so-called SustainML, that will foster energy efficiency throughout the whole life-cycle of Machine Learning (ML) applications: from the design and exploration phase that includes exploratory iterations of training, testing and optimizing different system versions through the final training of the production systems (which often involves huge amounts of data, computation and epochs) and (where appropriate) continuous online re-training during deployment for the inference process. The framework will optimize the ML solutions based on the application tasks, across levels from hardware to model architecture. It will also collect both previously scattered efficiency-oriented research, as well as novel Green-AI methods. Artificial Intelligence (AI) developers from all experience levels can make use of the framework through its emphasis on human-centric interactive transparent design and functional knowledge cores, instead of the common blackbox and fully automated optimization approaches.

This report corresponds to *Deliverable D2.2 - Open-source hardware exploration toolchain - Initial* of the SustainML project. This deliverable covers *Hardware Exploration Toolchain* designed for generating Field-Programmable Gate Array (FPGA)-based hardware implementations for given Deep Neural Network (DNN) models.

Contents

Acronyms

AI	Artificial Intelligence.
DNN	Deep Neural Network.
FPGA	Field-Programmable Gate Array.
HDL	Hardware Description Language.
HLS	High-level Synthesis.
ML	Machine Learning.
RTL	Register Transfer Level.

1 Introduction

In order to facilitate the estimation of carbon-dioxide production for an inference of a particular DNN model on a given FPGA board, the component developed in the current work package has to be able to estimate the execution latency and the power consumption of the underlying hardware implementation. The approach based on actual execution to measure runtime and power consumption is associated with extreme overhead as configuring FPGA with an actual bitstream requires multiple computationally intensive processes, like logic synthesis, placement, and routing. In this work package, various DNN layers and activation functions are implemented in a hardware library written as a collection of C/C++ templated functions with High-level Synthesis (HLS) annotations. Use of HLS is proved to significantly accelerate the development process and provide results competitive to Register Transfer Level (RTL) code implemented using Hardware Description Languages (HDLs). However, producing RTL code from the high-level code is associated with additional overhead, namely HLS. To avoid any aforementioned overhead, we target developing an agent that is capable of accurately predicting expected power consumption given the DNN model, targeted FPGA board, and the knowledge about the hardware architecture (hardware library) without following all computationally intensive steps. The agent has to be trained on data points providing the hardware resource utilization and power consumption corresponding to the given DNN model and targeted FPGA board. The knowledge about the resource utilization of the given DNN model is crucial to identify if the targeted FPGA has enough resources to accommodate the hardware implementation. The runtime can be computed based on the known frequency of the design and the execution latency per hardware instance of each component of DNN expressed in clock cycles. Knowing the size of the input data, the execution latency can be computed in a straightforward way using corresponding formulas. However, generating many data points, 100s if not 1000s, to train an accurate agent, requires an automatic process of generating hardware implementations of a variety of DNN models for a targeted FPGA. In the following, we are explaining the automatic *Hardware Exploration Toolchain* designed for generating FPGA-based hardware implementations for given DNN models.

2 Hardware Exploration Toolchain

To facilitate automatic implementation of given DNN topologies mapped onto custom hardware architecture, we implemented automatic *Hardware Exploration Toolchain* that is comprised of components for (1) generation of C/C++ HLS code based on a given DNN model, (2) software testing of the HLS code for the equivalence to Python-based DNN model, (3) hardware implementation step to produce a bitstream to configure FPGA, and (4) actual hardware testing on the targeted FPGA, see Fig. 1.

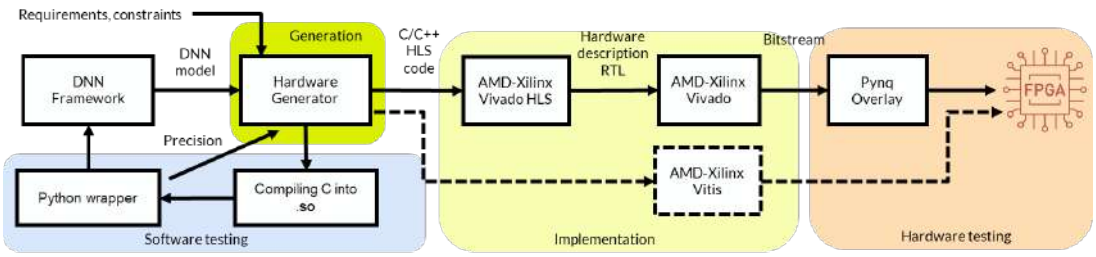

Figure 1: A block diagram of the Hardware Exploration Toolchain.

- The *Hardware Generator* generates the HLS code based on model saved in ONNX format. ONNX is a de facto standard format supported by all DNN frameworks, which enables the toolchain to support models trained in any Python-based DNN framework. Besides the DNN model, the *Hardware Generator* needs application-level requirements to be specified, like throughput, to identify the required parallelism for the hardware.

- The *Software testing* component compiles the HLS code to a library that is wrapped into a Python code that enables invoking the HLS code from any Python-based framework, which allows reusing the original testbench used to validate and test the DNN model. The hardware library supports FP32 format in order to ensure the functional equivalence to the Python-based DNN model. The hardware library also supports custom fixed-point data type for each layer individually for activations, weights, biases, accumulators, etc. Additionally, the hardware library supports integer weights, which is the common representation of quantized weights for CPUs and GPUs. It allows to use post-trained quantization from ONNX, TensorFlow Lite, etc. Using integer weights can be beneficial in comparison to fixed-point format that demands using an iterative procedure to find the optimal fixed-point configuration (position of a decimal point), which can be slow in comparison to post-trained quantization approaches using various divergence metric, like in [1]. Furthermore, post-trained quantization can be performed on GPU benefiting from massive parallelism and GPU-optimized kernels.
- The *Implementation step* is currently based on AMD-Xilinx Vivado HLS for HLS and AMD-Xilinx Vivado for system integration, logic synthesis, placement and routing. The output of this step is a bitstream used to configure an FPGA chip. The alternative approach, which is under investigation, uses a novel implementation flow based on AMD-Xilinx Vitis, which additionally to a hardware configuration generates software to facilitate the communication with memory mapped interfaces.
- The *Hardware testing* component facilitates FPGA configuration and testing the hardware implementation on the board using Pynq Overlay, which provides functions for operating the memory mapped interfaces. The toolchain instantiates the functions depending on the number of interfaces and their configuration, which in the case of using AMD-Xilinx Vitis is done automatically. The software generated by the AMD-Xilinx Vitis is based on OpenCL. The use of OpenCL can significantly ease the implementation of communication between the FPGA and host CPU. However, OpenCL operates kernels which execution is associated with an overheard that is not desirable in low-latency solutions. The comparison of the two implementation flows is still in progress.

The development of the agent to predict the hardware resource utilization and power consumption is conducted in close cooperation with University of Copenhagen. In this project, the targeted applications include surface inspection with a subtask of semantic segmentation. One of the most common DNN models to perform the task is U-Net [2]. The on-going research focuses on automatically generating hardware implementations of various flavors of U-Net model on FPGA, and extracting hardware resource utilization and power consumption to train the agent.

3 Hardware library

The original hardware architecture is designed to be low power and ultra-low latency [3]. Primarily, this is achieved by keeping all weights and intermediate results in on-chip memory since off-chip transfers consume more energy and introduce extra latency. External memory is only used to read input data and write results, therefore reducing memory access to the absolute minimum, see Fig. 2a. However, this approach constraints the size of DNN models to be implemented. To extend the space of supported DNN models, we are exploring hardware architectures proposed in [4, 5] with mixed storage of weights and feature maps, see Fig. 2b. Mapping bottleneck buffers for feature maps or weights to off-chip memory will free scarce on-chip memory resources for deeper topologies. U-Net topology is a perfect candidate for such investigation as DNN model requires very deep FIFO buffers to accommodate the skip connections, see Fig. 3.

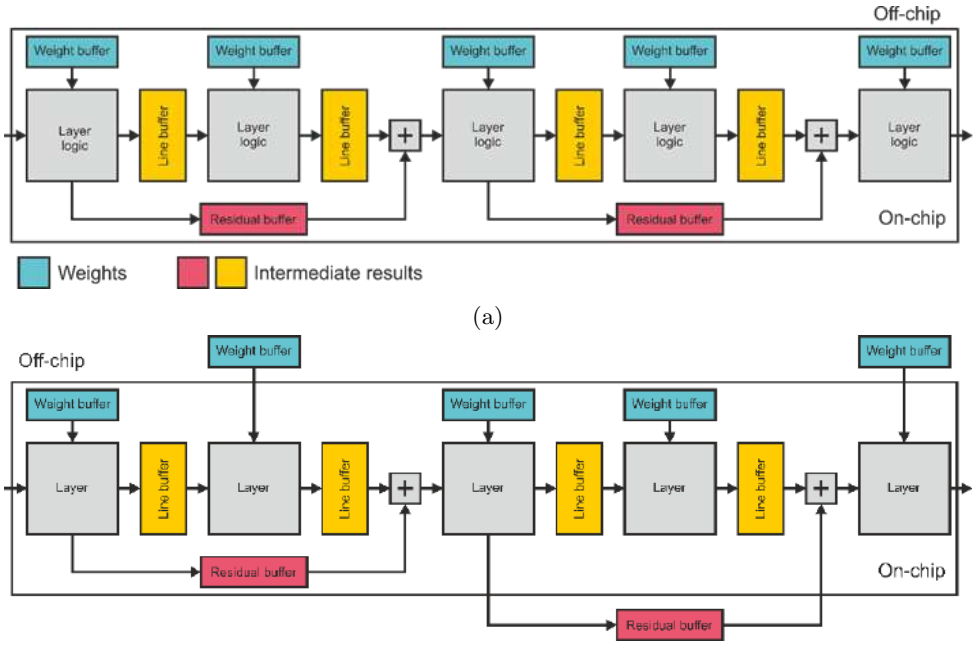

Figure 2: Hardware architecture with (a) all weights and feature maps mapped to on-chip memory resources and (b) with mixed mapping to on- and off-chip memory

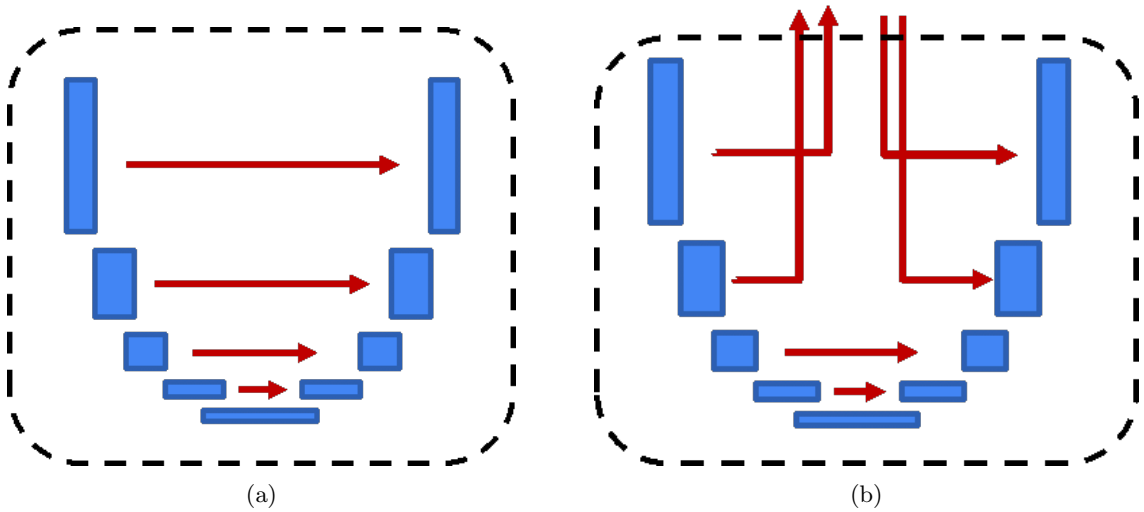

Figure 3: The U-Net topology with skip connections mapped to (a) on-chip vs. (b) off-chip memory resources

4 Summary

The on-going research is focused on the following key aspects:

- To enable the *Hardware Exploration Toolchain* to generate hardware implementations for a wider range of various DNN layers and topologies, to facilitate the generation of topologies with mixed mapping of weights and feature maps to on-chip and off-chip memory, to investigate the AMD-Xilinx Vitis implementation flow.
- Training an agent to accurately predict the hardware resource utilization and power consumption.
- To enable the hardware architectures to facilitate implementation of larger DNN models on FPGA.

References

[1] *N2D2 (for Neural Network Design Deployment) is CAD framework for designing and simulating Deep Neural Network (DNN), and building full DNN-based applications on embedded platforms.* `https://github.com/CEA-LIST/N2D2`. Accessed: 2024-10-11.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer. 2015, pp. 234–241.

[3] Jonas Ney et al. "HALF: Holistic Auto Machine Learning for FPGAs". In: *Accepted for publication, the 31st International Conference on Field Programmable Logic and Applications (FPL)*. IEEE. 2021.

[4] Xiaofan Zhang et al. "DNNExplorer: a framework for modeling and exploring a novel paradigm of FPGA-based DNN accelerator". In: *Proceedings of the 39th International Conference on Computer-Aided Design*. 2020, pp. 1–9.

[5] Xiaofan Zhang et al. "DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs". In: *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2018, pp. 1–8.