Optimizing Hardware for Neural Network Inference using Virtual Prototypes

Jan Zielasko[1,2] and Rolf Drechsler[1,2] *

[1]Institute of Computer Science, University of Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, Germany

Abstract

Identifying the optimal hardware configuration for running neural network inference on ultra-low-power edge devices is critical for reducing cost and maximizing performance of smart applications. Tailoring hardware designs to specific applications significantly increases resource utilization, which is essential to meet the strict performance and energy constraints. Virtual Prototypes enable early design-space exploration before any actual hardware is built, thus shortening time-to-market. They are essentially an executable model of the entire hardware platform and can model intricate hardware/software interactions including accelerators, while still being fast and easy to use. Building on prior work, we demonstrate how Virtual Prototypes can uncover previously unknown hardware optimizations using representative edge AI workloads from the MLPerf Tiny benchmark suite.

Introduction

Recent advances in AI have increased the demand for running complex applications such as *Neural Networks* (NNs) on resource-constrained devices, e.g., for sensor data processing on embedded systems or *Internet of Things* (IoT) nodes [1, 2, 3]. While *General-Purpose Processors* (GPPs) are commonly used as the computational core for *System-on-Chip* (SoC) designs, they are inefficient for NN inference. To meet the strict requirements for energy consumption and performance, GPPs can be tailored to applications to increase their efficiency. Previous approaches address this challenge at various abstraction levels ranging from the high-level down to the gate-level (e.g. [4, 5]). *Virtual Prototypes* (VPs) such as the RISC-V VP [6] can model the entire *Hardware* (HW) platform including subsystems such as peripherals, accelerators and other subsystems of the SoC. As such, VPs enable early *Software* (SW) development and analysis before any real HW is built. VPs achieve simulation speeds orders of magnitude faster than RTL simulation by modeling the SoC with *Transaction Level Modeling* (TLM), commonly implemented with SystemC [7]. Previous work introduces the *RISC-V Opt VP* [8] as an analysis platform for HW optimization. The tool traces the execution of RISC-V applications on the instruction level and analyzes every encountered instruction sequence up to a set depth. [8] demonstrates, that the tool can be used to identify a wide range of different HW optimizations for typical edge applications found in the Embench™ suite [1]. Compared to previously analyzed applications like *sha256* or *RIOT OS*, inference of NNs poses a much greater challenge, as the critical section is almost always part of the activation function or matrix multiplication. In this work, we analyze a set of typical edge AI applications to investigate, how effectively this approach identifies optimizations in more difficult inference scenarios. We chose a selection of applications using the *TensorFlow Lite for Microcontrollers* [2] framework. This set includes the *MLPerf Tiny* [3] benchmark set, which consists of tasks like anomaly detection, keyword spotting and image classification. Our results show that we are able to identify the expected types of optimization candidates (e.g. MAC), but also a set of instruction sequences outside the matrix multiplication kernel, which are non-trivial to find from either the source or gate-level. The tool[3] and all ML benchmarks [3] are available as open-source.

Methodology

As a first step to identifying HW optimizations, we configure the VP to represent a minimal configuration that can run the target application, i.e. configuring the supported extensions and setting memory to encompass the tensor arena size required to run the NN. When executing an application, the VP traces the execution of every instruction including all required metadata about the system and any inter-instruction dependencies. It stores the information in k-trees with a predetermined maximum depth. A separate tree is created for each instruction, containing all sequences starting with that instruction. The bound limits the length of instruction sequences considered for optimization. As determined in previous work, a bound

*Corresponding author: Jan.Zielasko@DFKI.de .
This work was supported by the German Federal Ministry of Education and Research (BMBF) within projects ECXL under grant no. 01IW22002 and FAIRe (grant no. 01IS23074).
[1] https://www.embench.org/
[2] https://github.com/tensorflow/tflite-micro
[3] https://github.com/agra-uni-bremen/opt-vp

Figure 1: *Instruction distribution for image classification*

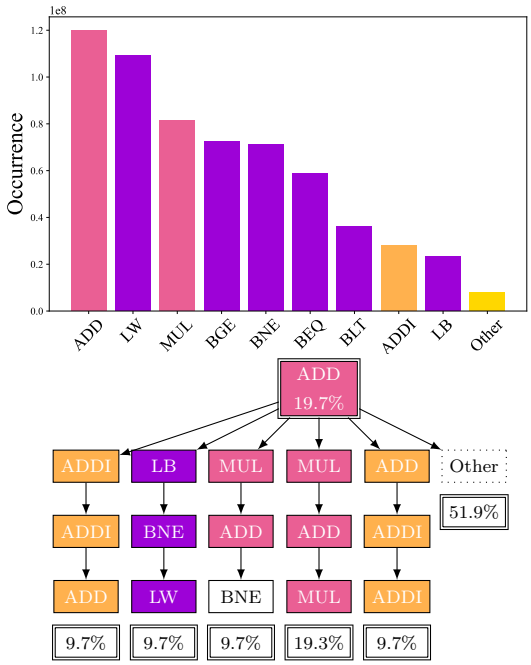

Figure 2: □ *Other,* ■ *MAC,* ■ *Addition,* ■ *Branching*

of 50 is sufficient to discover all relevant sequences for almost any application. Further analysis shows, that in practice, a bound of 10 is enough to discover all relevant sequences for smaller applications. If the bound is set too low, the tool suggests increasing it to uncover additional sequences.

Case Study

We demonstrate the tool in a case study using the image classification reference model from *MLPerf Tiny*. It is a ResNet8 model trained on the CIFAR10 dataset able to classify low resolution images of 10 different categories. Running a single inference cycle executes approximately 608 million instructions, simulating 23.2 seconds. The execution time of the simulation and tracing largely depends on the tree bound and takes around 1h with depth 7. Analysis of the trees, i.e. exhaustively exploring all available sequences takes less than 1 second. Figure 1 shows the distribution of executed instructions among all sequences. ADD (19.7%) and LW (17.9%) were most frequently executed with only 9 instructions amounting to (98.7%) of total execution. Figure 2 represents an excerpt from the tree containing all sequences starting with an *ADD*. As expected, by analyzing the trees we are able to identify multiple different variations of MAC operations. Two of which are contained in the *ADD* tree highlighted in ■. ■ highlights a chain of *ADD* operations related to the matrix multiplication. The instructions highlighted in ■ contain part of the most interesting sequence discovered by our analysis. They are part of a set of longer sequences containing a number of branch and load instructions depending on a single value from memory. This makes the sequence a good candidate for parallel execution as a new custom instruction.

Discussion and Future Work

In this extended abstract we presented an extension to the VP-driven approach, and analyzed a range of ML applications. We found that we are able to identify typical ML optimizations like MAC operations. Additionally, we discovered that the tool can be used to identify additional HW optimizations that are non-trivial even for other analysis approaches. Comparing the results from classifying different images or running multiple inference cycles shows that the sequences remain almost identical with differences of fewer than 2000 instructions across runs (compared to a total of $6 * 10^8$) between different images in the case study. This implies that a HW accelerator designed for a representative case should generalize well across typical inputs. Common patterns across different ML applications suggest that designs could even be reused for new applications. Analyzing the similarities can then identify the accelerator with the highest reusability. For future work we plan to automatically generate custom instructions or HW accelerators using SpinalHDL [9] to assess the HW performance improvement beyond our program-level analysis.

References

[1] John L. Hennessy and David A. Patterson. "A new golden age for computer architecture". In: *Commun. ACM* 62.2 (Jan. 2019), pp. 48–60. ISSN: 0001-0782. DOI: 10.1145/3282307. URL: https://doi.org/10.1145/3282307.

[2] Carsten Bormann et al. *Terminology for Constrained-Node Networks*. RFC 7228. May 2014. DOI: 10.17487/RFC7228. URL: https://www.rfc-editor.org/info/rfc7228.

[3] Colby Banbury et al. "MLPerf Tiny Benchmark". In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks* (2021).

[4] Weier Wan et al. "A compute-in-memory chip based on resistive random-access memory". In: *Nature* 608 (). DOI: 10.1038/s41586-022-04992-8.

[5] Yijin Guan et al. "FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates". In: *FCCM*. 2017, pp. 152–159. DOI: 10.1109/FCCM.2017.25.

[6] V. Herdt et al. "RISC-V based virtual prototype". In: *Journal of Systems Architecture* 109 (2020).

[7] F. Ghenassia. *Transaction-Level Modeling with SystemC*. New York: Springer, 2010.

[8] J. Zielasko and R. Drechsler. "Virtual Prototype Driven Application Specific Hardware Optimization". In: *FDL*. IEEE, 2023, pp. 1–8.

[9] Charles Papon and Yindong Xiao. *SpinalHDL*. URL: https://github.com/SpinalHDL/SpinalHDL.