2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM) | 978-1-6654-3555-0/20/$31.00 ©2021 IEEE | DOI: 10.1109/FCCM51124.2021.00062

Using hls4ml to Map Convolutional Neural Networks on Interconnected FPGA Devices

Evangelos Mageiropoulos*, Nikolaos Chrysos†, Nikolaos Dimou‡, Manolis Katevenis§
Foundation for Research and Technology Hellas (FORTH), Crete, Greece
Email:*emageir@ics.forth.gr, †nchrysos@ics.forth.gr, ‡ndimou@ics.forth.gr, §kateveni@ics.forth.gr

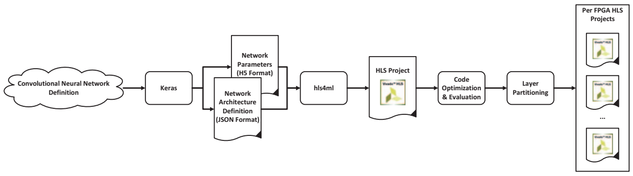
Convolutional Neural Network Definition
Keras
Network Parameters (H5 Format)
Network Architecture Definition (JSON Format)
hls4ml
HLS Project
Code Optimization & Evaluation
Layer Partitioning
Per FPGA HLS Projects
...

Fig. 1. Workflow used in this paper to partition CNNs in multiple FPGAs.

Convolutional Neural Networks (CNNs) have an increasing presence in our everyday life. FPGAs present a competitive platform for implementing CNNs, due to their reconfigurable architecture and high energy efficiency compared to CPUs and GPUs. The dominant implementation of CNN accelerators on FPGAs relies on a generic hardware unit that is time shared among the different layers of the CNN [1]; in this paradigm, each layer typically fetches the parameters (weights and biases) from DRAM, in order to compute its output.

A different approach which we exploit in this paper is to map the layers of a CNN onto a set of discrete FPGAs that work in tandem, and also in pipeline, obtaining a data-flow engine for every CNN, with simple, feed-forward control-flow [2], [3]. In this architecture, we can store all the network parameters in on-chip buffers in order to avoid DRAM-access bottlenecks or the need for DRAM on devices altogether. This architecture is also suitable for distributed, cooperative inference on devices equipped with FPGAs. Furthermore, as each FPGA implements one specific part of the inference, its kernel can utilize multiple, specialized computation engines in the FPGA that work concurrently to process data and increase throughput.

In this paper, we demonstrate this architecture by partitioning the SqueezeNet CNN [4] in six (6) Ultrascale+ FPGAs and use existing tools in order to simplify the path from network definition to HLS. In particular, we use Keras [5] in order to define arbitrary convolutional neural networks and hls4ml [6] to generate HLS kernels that can be used in a Vivado HLS project. An unexpected finding is that original HLS code generated by hls4ml is suboptimal for our purpose. Therefore, in order to achieve satisfactory results, we optimize each of its kernels appropriately by applying optimization directives available by Vivado HLS and changing the original kernel code when needed.

Figure 1 demonstrates the steps that we followed to map SqueezeNet in multiple FPGAs. We first use Keras and hls4ml to create a first HLS project for SqueezeNet. We then individually optimize the functions of hls4ml that implement the convolutional, max pooling, global average pooling and softmax layers. We finally partition the network into sets of building blocks (kernels), with each set assigned to a different FPGA, by taking into consideration the FPGA real estate required for each set, while striving to equally load-balance the work across FPGAs and minimize their communication. We use 8-bit fixed-point data representations for the network parameters. All of the modules in our design run at a clock frequency of 200MHz. We set the Xilinx UltraScale+ MPSoC, part xczu9eg-ffvc900-2-e [7] as the target device for all projects in Vivado HLS.

Our final implementation achieves an inference latency of 24 msecs and a throughput of 251 inference tasks per second, thus satisfying common service-level agreements (SLAs) [8], with an estimated power efficiency reaching 3.33 GOPS/Watt. We use 10 Gbps High-Speed Serial Links for inter-FPGA communication and transfer data between FPGAs with Aurora transceivers and AXI-Stream to Aurora converters. We are currently evaluating the inter-FPGA communication in actual hardware. The size of each inter-FPGA data transfer is at most 392 Kbits. Therefore, the transfer time at 10 Gbps link rate is at most 40 μs, i.e. significantly smaller than the compute time of the layers in an FPGA, which ranges between 0.8 and 3.9 ms. Hence, we expect that inter-FPGA data communication latency will not affect the advertised throughput.

References

[1] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[DL] A Survey of FPGA-Based Neural Network Inference Accelerator," vol. 9, no. 4, p. 26.

[2] C. Zhang, D. Wu, J. Sun, G. Sun, G. Luo, and J. Cong, "Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster:," Aug. 2016, pp. 326–331.

[3] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN Accelerator Efficiency Through Resource Partitioning," in Proceedings of the 44th Annual International Symposium on Computer Architecture, Toronto ON Canada, Jun. 2017, pp. 535–547.

[4] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size.

[5] Chollet, F., & others. (2015). Keras. GitHub.
Retrieved from https://github.com/fchollet/keras

[6] J. Duarte et al., "Fast inference of deep neural networks in FPGAs for particle physics," J. Inst., vol. 13, no. 07, pp. P07027–P07027, Jul. 2018.

[7] Zynq UltraScale+ Device Technical Reference Manual UG1085.

[8] Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. Commun. ACM 56, 2 (February 2013), 74–80.

978-1-6654-3555-0/21/$31.00 ©2021 IEEE
DOI 10.1109/FCCM51124.2021.00062