Hardware Acceleration of Complex Machine Learning Models through Modern High-Level Synthesis

Serena Curzel*
serena.curzel@polimi.it
Politecnico di Milano
Milano, Italy

Antonino Tumeo
antonino.tumeo@pnnl.gov
Pacific Northwest National Laboratory
Richland, Wa, USA

Fabrizio Ferrandi
fabrizio.ferrandi@polimi.it
Politecnico di Milano
Milano, Italy

INTRODUCTION

Modern scientific experiments employ instruments (e.g., electron microscopes, detectors in particle accelerators, environmental sensors) that generate exponentially growing volumes of raw data. Machine learning (ML) and deep learning algorithms are well suited to process and analyze large amounts of data, as it has been repeatedly proven in applications such as image classification, natural language processing, or recommendation systems. Both ML training and inference are compute- and memory-intensive, leading to widespread adoption of heterogeneous systems containing specialized accelerators. While graphic processing units (GPUs) are the established platform of choice to accelerate training, they are often too power-hungry to run inference tasks, or cannot meet the strict latency requirements of scientific experiments. A variety of custom solutions implemented as field programmable gate arrays (FPGAs) or application-specific circuit (ASICs) have been proposed in their place, ranging from generic "neural processors" to accelerators that focus on a narrow set of models with great efficiency.

While FPGAs and ASICs promise to provide the necessary hardware specialization, experimental workflows and the related machine learning methods evolve quickly. Even if FPGAs can be configured after deployment, allowing to update the accelerators and support new models [4], they still require significant expertise in hardware design and hardware description languages (HDLs) to be effectively used.

On the other hand, domain scientists are used to leverage python-based high-level frameworks (e.g., TensorFlow, Pytorch) to quickly develop and explore ML algorithms, and typically do not have any hardware design expertise. To bridge the two worlds, some works propose to exploit High-Level Synthesis [1] [3], a well established technique that reduces the effort required to design hardware accelerators by automatically translating a behavioral description (traditionally C or C++) into Verilog/VHDL.

Most of the HLS-based frameworks for Machine Learning use C/C++ as an intermediate representation of the input model, augmenting it with tool-specific directives that drive the synthesis to obtain an efficient design. Our proposal, instead, is a compiler-based, open source framework that exploits Multi-Level Intermediate Representation (MLIR [5]) to lower the input model, apply optimization passes at the correct level of abstraction, and exploit knowledge about the HLS process to guide them.

A limitation of the existing frameworks is that they usually focus on a narrow set of models, specifically Multi-Layer Perceptrons and Convolutional Neural Networks. While it is true that these cover a significant part of ML applications (especially in the computer vision field), there is room for exploring other types of models, for example to accelerate scientific applications that work on graphs or sparse data structures. A narrow focus limits the possibility of quickly adapting to newer algorithmic approaches, a factor which needs to be taken into higher consideration.

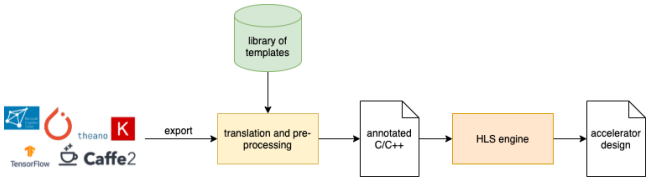
Finally, as the MLIR project continues to grow, our tools are going to be available to other classes of algorithms, not exclusively Machine Learning models: in fact, any domain-specific framework with a lowering to basic MLIR dialects will be able to exploit our optimizations and translate algorithms into an efficient hardware accelerator. In the context of a scientific experiment, this will allow to accelerate also other stages in the data acquisition pipeline, such as pre-processing, analysis, and simulation.

"CLASSIC" AND "MODERN" HLS FLOWS

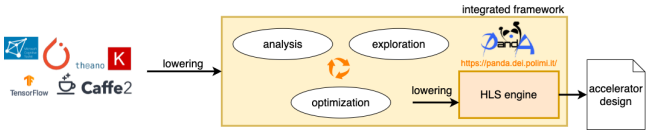
Current frameworks that help automating the design of Machine Learning accelerators (e.g., hls4ml [3], FINN [1]) use existing High-Level Synthesis tools (e.g. Vitis/Vivado HLS) as back-end: they parse a model exported from popular ML frameworks and replace operators with C/C++ functions taken from a library of templates, adding tool-specific directives to guide optimizations (Figure 2a). The HLS tool processes this intermediate C/C++ representation and produces a corresponding accelerator design without further manual intervention.

Our compiler-based flow SODA [6] has two fundamental differences with respect to such a "classic" approach (Figure 2b). First,

Figure 1: Two different ways of generating hardware accelerators through HLS.



(a) "Classic" HLS flow with intermediate C/C++.



(b) "Modern" compiler-based HLS flow.

*Also with Pacific Northwest National Laboratory.

instead of translating one high-level description into another high-level language such as C/C++, we embrace the MLIR multi-level approach to work with progressive lowerings. Increased modularity makes it is possible to apply and explore both high-level algorithmic optimizations and low-level hardware-oriented ones. Second, we integrate the open-source HLS tool Bambu [2] in the loop, to have more control on the underlying High-Level Synthesis process.

A significant drawback of the "classic" flow is that the library of templates is necessarily tied to a single, specific HLS tool: each tool expects coding patterns, annotations, and optimization directives that are not portable to other tools; incompatible directives would be ignored, resulting in a very inefficient design. As typically there is one commercial HLS tool for each FPGA vendor, the choice of target boards is limited to the ones supported by the chosen HLS tool. This is not the case for our design flow, since Bambu is a multi-platform tool supporting multiple FPGA (Xilinx, Intel, Lattice, NanoXplore) and ASIC targets without any modification in the input code: such flexibility enables a smooth transition from high-level design, to FPGA prototyping, to ASIC manifacturing and deployment.

MODEL DIVERSITY CHALLENGES

Machine Learning is an umbrella term that covers a broad spectrum of algorithms; research works about FPGA acceleration and HLS-based design flows are mostly focused on a subset of ML models, i.e., Multi-Layer Perceptrons and Convolutional Neural Networks. Sometimes their field of action can be even smaller: for example, the original implementation of hls4ml was optimized for small, fully-connected models under tight latency constraints, reflecting the needs of a high-energy physics experiment at CERN. Appropriate implementation choices for such a specific case may not be beneficial to a generic model: hls4ml proposed to store network weights inside on-chip logic and unroll all loops to increase parallelism, which quickly depletes FPGA resources when considering a neural network with more layers and weights.

A second challenge after supporting models with different sizes is their degree of sparsity: large models can be compressed to reduce their computation and memory requirements, for example by employing low precision data types (quantization) or by removing operations with zero values. Quantization is well suited to hardware acceleration, since custom precision operators can be implemented quickly and efficiently (also through dedicated HLS libraries). Sparse tensors, on the other hand, imply irregular computation, communication, and memory access patterns, which result in poor efficiency when mapped on accelerators designed for dense models.

Another class of models that could benefit from hardware acceleration and requires unique design choices is Graph Neural Networks. Graph structures provide great expressive power to represent and analyze data in a variety of applications, from chemistry to language, social networks, recommendation systems etc. Machine Learning models that work on graphs include both sparse (aggregation) and dense (feature extraction) computation patterns, which are also affected by the input graph size; such characteristics could benefit from a task-based parallelism paradigm. Existing HLS-based design flows are good at extracting data- and instruction-level parallelism (e.g. by unrolling loops), but they are not equipped to deal with the irregular task-based patterns required by graph processing.

Table 1: Preliminary results on CNN and GCN models.

	FPGA target			
	Clock (MHz)	**Registers**	**LUTs**	**Latency (s)**
LeNet	146.75	44171	44 325	0.689
ResNet-50	217.82	20861	20 522	284.640
GCN 1 (dense input)	204.80	14812	8965	1302.990
GCN 2 (sparse input)	210.00	14639	8685	6.414

	ASIC target			
	Clock (MHz)	**Size (mm^2)**	**Cells**	**Latency (s)**
LeNet	200	0.262	187 415	0.362
ResNet-50	200	0.305	220 156	212.190
GCN 1 (dense input)	200	0.062	44 864	572.738
GCN 2 (sparse input)	200	0.062	45 028	4.138

Table 1 presents some results obtained synthesizing two CNNs and a simple Graph Convolutional Network with SODA. LeNet is a relatively small network, which was synthsized as a single Verilog module; for ResNet-50 instead each single layer was outlined and implemented as a single accelerator. The GCN was implemented in two slightly different versions, one for dense inputs and one for sparse inputs. We also experimented with two different targets among those offered by Bambu, a Xilinx Zynq-7000 FPGA and a 45nm ASIC. These preliminary results do not include any optimization, but they show that the SODA design flow has the potential to tackle a diverse set of models; more research will allow to tune the compilation passes according to the specific needs of different ML algorithms.

CONCLUSION

The SODA framework provides a novel approach to the design of accelerators for Machine Learning, bridging the gap between algorithmic frameworks and hardware design through MLIR and High-Level Synthesis. Its multi-level strategy allows to exploit knowledge about the internal mechanism of the selected HLS tool, Bambu, and to operate analysis and transformations as compiler passes on different intermediate representations, retrieving relevant information that might otherwise be lost in the translation to C/C++. Future developments will focus on extending the framework to support different classes of ML models, for example tailoring the compilation to the needs of sparse and graph-based applications (benefiting also non-ML algorithms).

REFERENCES

[1] Michaela Blott, Thomas B Preußer, Nicholas J Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, et al. 2018. FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11, 3 (2018), 1–23.

[2] Politecnico di Milano. [n.d.]. *Bambu: A Free Framework for the High-Level Synthesis of Complex Applications.* Retrieved Aug. 11, 2021 from https://panda.dei.polimi.it/?page_id=31

[3] Javier Duarte, Song Han, Philip Harris, Sergo Jindariani, Edward Kreinar, Benjamin Kreis, et al. 2018. Fast inference of deep neural networks in FPGAs for particle

physics. *Journal of Instrumentation* 13, 07 (2018), P07027.

[4] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, et al. 2018. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 1–14.

[5] MLIR project. [n.d.]. *Multi-Level IR Compiler Framework*. Retrieved Aug. 11, 2021 from https://mlir.llvm.org/

[6] Jeff Zhang, Nicholas Bohm Agostini, Shihao Song, Cheng Tan, Ankur Limaye, Vinay Amatya, et al. 2021. Towards Automatic and Agile AI/ML Accelerator Design with End-to-End Synthesis. In *IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE.