Design Space Exploration For Machine Learning Architectures

Michael Barrow, Olivia Weng, Ryan Kastner
barrow9@llnl.gov, oweng@ucsd.edu, kastner@ucsd.edu

TOPIC: HARDWARE SOFTWARE CO-DESIGN TOOLS FOR MACHINE LEARNING

Hardware designers and software developers operate in fundamentally different ways. Hardware designers deal with highly parallel programming constructs, optimize their designs heavily, and spend a majority of their efforts on verification. Software developers focus on usability, fast incremental development, and integration into complex systems. By and large, hardware and software communities have operated in isolation by using hardware/software interfaces and abstractions (e.g., ISAs).

In recent years, the line between hardware and software has blurred. “Software” companies are seeing the advantages of developing their own custom hardware, e.g., Microsoft’s SQ2 chip. Since the hardware is designed for the application at hand, it provides advantages in performance, power consumption, security, and cost. Furthermore, the emergence of open-source hardware designs, languages, and abstractions have made hardware design easier. Despite all these in-roads, hardware design still remains challenging.

Machine Learning (ML) applications exemplify these challenges. ML is a must-have for software companies. For example, 70% of Google revenue is from ML targeted advertisements. ML algorithms are highly parallel and good targets for specialized parallel hardware. But, creating optimal ML hardware designs is difficult because the hardware design space is not considered during ML algorithm development. The ML algorithm design space is high in dimensions, with many viable network architectures, hyperparameter settings, and optimization approaches (e.g., pruning and quantization). Traditionally, software companies meticulously search this space using high level ML frameworks like PyTorch and Tensorflow to meet their accuracy requirements. These frameworks however largely abstract away and ignore the multi-dimensional space of hardware architectures (e.g., memory bandwidth and hardware precision). ML developers leave it to the hardware specialists to take their finished ML applications as they are and optimize their performance for hardware. This approach is problematic. Because ML frameworks do not take the hardware design space into account, many of the ML models are developed at the expense of hardware. It is only possible to efficiently design hardware for a ML application if the ML algorithm design space is co-explored with possible hardware mappings.

The traditional approach of isolated software and hardware development suffers from this major challenge: *Since software developers have already fixed the dimensions of the ML algorithm design space by the time hardware developers receive them, the hardware design space cannot be effectively explored to achieve hardware-efficient designs.*

CHALLENGE: EFFECTIVE DESIGN SPACE EXPLORATION

State-of-the-art ML applications are complicated, with thousands of dimensions and millions of weight parameters. It is difficult for application developers to understand the impact of their design decisions on hardware performance. Similarly, it is difficult for hardware developers to understand the impact of their decisions on algorithm accuracy. Without feedback between each other, it is not possible to explore the application design space with respect to both the software and hardware intelligently.

Recent developments have sought to effectively and efficiently map neural networks to hardware. Proposals have been made by both hardware and software oriented groups, with mixed results. FINN is a open source framework initially developed by the hardware company Xilinx [5]. FINN is an end-to-end tool that generates hardware accelerated inference networks for Xilinx FPGA platforms, and is focused on quantized neural networks. The tool is implemented using Python and Vivado HLS and has interfaces to high level ML frameworks such as pytorch. The benefit of this is that hardware performance is exposed to the algorithm development environment. Although FINN is capable of generating highly optimal designs, it is not well suited to design space exploration. Modifying a network requires domain specific hardware knowledge. In our experience, any modifications took several months for PhD and Postdoctoral students to make.

hls4ml is another end-to-end Python tool for generating accelerated inference networks, but was developed by a consortia of software-centric academics [1]. hls4ml has a broader support for ML network designs compared to FINN. It is also better documented, leading to shorter development times. However, hls4ml relies on Vitis®to map its generated quantized network IP cores to a Xilinx FPGA and is less optimal. Conversely, FINN expects the user to manually configure a network mapping on the FPGA.

Although these state-of-the-art ML frameworks bridge the gap between application and hardware design, the status quo end-to-end paradigm is limited. Existing tools lack feedback cues between the ML and hardware mapping steps, or lack knobs to tune hardware mappings. Consequently, the end-to-end flow cannot inform iterative exploration of the complete design space, costing many development hours on both the software and hardware side. What is needed is a framework with a more informative feedback loop from the hardware tools to the algorithm developer, with more intuitive (ML-specific) hardware map tuning knobs.

OPPORTUNITY: APPLICATION-SPECIFIC FEEDBACK TOOL FOR DESIGN SPACE EXPLORATION

State-of-the-art ML application hardware mapping is done using High Level Synthesis (HLS) compilers. Although HLS has long sought to bridge high level application specifications to hardware mapping, HLS compilers are unwieldy tools. HLS primitives are highly granular so that HLS can be applied to a broad spectrum of application classes. Unfortunately, the consequence is that domain-specific hardware design knowledge is required to describe and schedule ML primitives using HLS tools. *This presents an opportunity to develop ML-focused tools that allow co-exploration of the algorithm and hardware design spaces*.

We want to "lift" the hardware design-space exploration to the software or even the algorithmic level to make hardware software co-design more commonplace. Without exploring the hardware architecture implementation specifics, design choices made using high level ML frameworks to improve model performance can turn out to be unexpectedly costly during hardware deployment. For example, although Resnet-a-like networks have good training speed thanks to gradient highways, they can cause major challenges when trying to realize efficient hardware implementations. Naive skip connections waste resources due to large buffers required to accurately represent the network, but the connections may not even be necessary to achieve acceptable results.

Such pitfalls could be avoided with a tool that gives machine learning developers access to various algorithmic, software, and hardware knobs that they can tune whilst they develop their ML algorithms using an enhanced end-to-end flow. Some example knobs follow in Table 1:

ML Knobs	hyper-parameters	model parameters / topology	time to convergence	model sparsity	quantization
HW Knobs	memory bandwidth	power requirements	resource utilization	throughput	

Table 1: Hardware Software Co-design knobs that could be exposed to ML algorithm developers

MATURITY: REMAINING GAPS IN ML SPECIFIC DESIGN SPACE EXPLORATION APPROACHES

Recent advances in Neural Architecture Search and adjacent research areas offer steps towards such an all-encompassing tuning tool [3, 4, 6]. For example, [4] introduced Adaptive Threshold Non-Pareto Elimination (ATNE), which is a design space exploration framework that uses machine learning to tune various OpenCL-to-FPGA knobs (such as unroll factor and number of SIMD lanes) to automate finding Pareto-optimal designs for a given high level application. This demonstrates how a smart Hardware Abstraction Layer (HAL) can optimize hardware using a set of abstract design knobs and insulate the application developer from needing to understand how to make progress towards an optimal hardware mapping.

Although promising, these works are hampered in ML applications because the HAL is not capable of optimizing the ML network topology. In other words, they do not provide ML network optimization hints to help the application developer explore the hardware design space as well.

Neural Architecture Search has recently been extended to find hardware and resource friendly ML network designs, as seen in [2]'s Lamarckian evolutionary algorithm for multi-objective neural architecture design (LEMONADE) algorithm. In Elsken's work, hardware architecture is assumed fixed, and a subset of ML design knobs from Table 1 are explored by the LEMONADE algorithm. This work represents the "top down" counterpart to "bottom up" approaches such as ATNE that conversely assume the ML design is fixed and explore the HW knobs in Table 1.

Bridging the gap between works such as LEMONADE and ATNE is an exciting prospect that would enable co-exploration of the ML algorithm and hardware design space. Success in this area would streamline the development of some of the most widely used applications today. ML application developers would be provided with the tools they need to develop hardware-friendly networks, and hardware specialists would be able to make informed optimizations to these networks.

References

[1] DUARTE, J., HAN, S., HARRIS, P., JINDARIANI, S., KREINAR, E., KREIS, B., NGADIUBA, J., PIERINI, M., RIVERA, R., TRAN, N., ET AL. Fast inference of deep neural networks in fpgas for particle physics. *Journal of Instrumentation 13*, 07 (2018), P07027.

[2] ELSKEN, T., METZEN, J. H., AND HUTTER, F. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081* (2018).

[3] LIU, H.-Y., AND CARLONI, L. P. On learning-based methods for design-space exploration with high-level synthesis. In *Proceedings of the 50th annual design automation conference* (2013), pp. 1–7.

[4] MENG, P., ALTHOFF, A., GAUTIER, Q., AND KASTNER, R. Adaptive threshold non-pareto elimination: Re-thinking machine learning for system level design space exploration on fpgas. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2016), IEEE, pp. 918–923.

[5] UMUROGLU, Y., FRASER, N. J., GAMBARDELLA, G., BLOTT, M., LEONG, P., JAHRE, M., AND VISSERS, K. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2017), pp. 65–74.

[6] ZULUAGA, M., SERGENT, G., KRAUSE, A., AND PÜSCHEL, M. Active learning for multi-objective optimization. In *International Conference on Machine Learning* (2013), PMLR, pp. 462–470.