ScaleHLS: Achieving Scalable High-Level Synthesis through MLIR

Hanchen Ye[1], Cong Hao[2], Hyunmin Jeong[1], Jack Huang[1], Deming Chen[1]
[1]University of Illinois at Urbana-Champaign, [2]Georgia Institute of Technology

ABSTRACT

High-level Synthesis (HLS) has been widely adopted as it significantly improves the hardware design productivity and enables quick design space exploration (DSE). However, existing HLS tools do not scale well to large designs for two main reasons: (1) The intermediate representations (IR) are not initially designed for HLS, thus are not expressive enough for comprehensive HLS design spaces; (2) The traditional HLS algorithms are based on a single-level abstraction, thus cannot easily capture the design hierarchy and are not scalable as the design size grows. To tackle these problems, we present ScaleHLS, a new HLS compilation flow based on a multi-level compiler infrastructure, MLIR. Utilizing MLIR, ScaleHLS introduces a hierarchical representation mechanism for HLS designs, enables scalable optimizations at multi-level abstractions, and directly generates optimized synthesizable HLS code. This approach not only explores the hierarchical design space efficiently but also scales well to large HLS designs. The initial experiments show that comparing to the baseline designs only optimized by the regular LLVM optimizations of Xilinx Vivado HLS, ScaleHLS improves the performance by up to 768.2× on computation kernel level algorithms and 4107.6× on a neural network model MobileNet-v2.

1 INTRODUCTION

High-level synthesis (HLS) technique automatically translates high-level languages to dedicated hardware accelerators, thereby eliminating the cumbersome and error-prone programming of hardware description languages [11]. Recent years, HLS has been widely used in many applications, including neural networks [1], IoT [13], video processing [7], etc. These HLS designs highly rely on user-specified directives and manual code transformation to improve the quality of hardware. However, as HLS tools open up large design spaces, non-ideal design choices may easily lead to sub-optimal solutions and poor overall performance.

Recently, we have witnessed a large number of papers investigating the automatic quality of results (QoR) estimation and design space exploration (DSE) for HLS. Authors of [14–16] extracted necessary design information from static dataflow graphs or dynamic execution traces, then passed such information to predefined analytical models for generating the estimation. Authors of [4, 8, 9] introduced machine learning methods to extract unique features that cannot be easily parameterized by analytical models. On top of QoR estimation, authors of [14, 15] proposed automatic DSE tools based on the guidance of resource and performance estimations. In addition, polyhedral analysis and integer linear programming (ILP)-based algorithms were exploited in [5, 17, 18] for proposing qualified design candidates and searching for the optimal solution under hardware resource constraints.

However, existing DSE methods are difficult to handle large HLS designs containing a large number of sub-modules and sophisticated inter-dependencies. The reason is that existing efforts heavily rely on unified and flattened intermediate representations (IR), such as LLVM, for conducting analysis and optimizations. Such a low-level IR barely supports hierarchical hardware optimization techniques, such as task/module level resource-sharing, scheduling and parallelization, inter-loop analysis and transformation, and hardware IP integration. These design optimizations are located at different abstraction levels and are very difficult to be explored and applied on a flattened low-level IR, thereby obstructing current approaches to comprehensively explore and optimize the large designs through HLS. Furthermore, the flattened IR of a large design will lead to a large non-hierarchical design space, which is hard to be effectively searched through the existing DSE algorithms.

To address the difficulty of handling large HLS designs and make the automatic DSE more scalable and flexible, we introduce *ScaleHLS*, a next-generation HLS tool which can represent and optimize large designs at multiple abstraction levels. The main contributions of this paper are:

- To the best of our knowledge, ScaleHLS is the first MLIR-based end-to-end HLS compilation flow.
- We propose a hierarchical and scalable optimization methodology, which optimizes HLS designs at multiple abstraction levels, including graph, loop, and directive levels, to handle the increasing design space as the HLS design size grows.
- We propose an automated DSE engine to search for the Pareto frontier of the important latency-area trade-off space. A QoR estimator is also developed to evaluate design points discovered by the DSE engine rapidly.
- We design a synthesizable HLS C++ emitter for bridging the gap between the MLIR compilation framework and RTL generation back-ends.

2 SCALEHLS FRAMEWORK

ScaleHLS is built on top of MLIR [2, 6], which is a compilation framework incorporating multiple levels of functional and representational hierarchy. ScaleHLS compiles programs described in high-level programming frameworks (e.g., ONNX [3] and PyTorch [10]) or general-purpose languages (e.g., C/C++) to synthesizable HLS C++ code. Figure 1 shows the architecture of the ScaleHLS framework, where *Dialect* is an MLIR terminology referring to a set of customized operations, types, and attributes. In the ScaleHLS compilation pipeline, the input programs are first parsed into MLIR constructed with tensor-level operations (e.g., ONNX and ATen

LATTE '21, April 15, 2021, Virtual, Earth

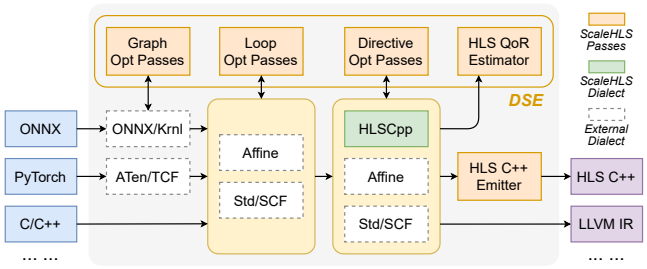

Figure 1: ScaleHLS framework.

Table 1: DSE results of computation kernels.

Kernel	Prob. Size	Speedup	Unroll Factors	Target II
BICG	2048	83.3×	[32,8]	43
GEMM	2048	768.2×	[16,1,16]	6
GESUMMV	2048	199.2×	[4,32]	9
SYR2K	2048	423.8×	[8,8,8]	29
SYRK	2048	542.3×	[32,4,8]	34
TRMM	2048	614.6×	[4,4,64]	25

dialects), which is then lowered to loop-level representations using MLIR built-in Affine, Standard, and SCF dialects. To enable the hierarchical HLS within MLIR, we build an HLSCpp dialect for representing the HLS-specific operations and attributes (e.g., loop pipelining) and enabling the synthesizable C++ code generation.

The hierarchical representation strategy combining external and our customized dialects helps ScaleHLS to leverage the powerful transformation and analysis infrastructures of MLIR while enabling a comprehensive capability of conducting HLS-oriented optimizations. As shown in Figure 1, our ScaleHLS optimization passes are applied at multiple compilation levels, including graph, loop, and directive levels, to improve the design quality. For exploring the large design space brought by the large HLS designs, ScaleHLS provides a fast QoR estimator based on analytical models which can take the structural IR as input and estimate the latency and resource utilization. Combining the QoR estimator and all optimization passes crossing different abstraction levels, we deliver an automated DSE engine to search for the Pareto frontier of the design space, where each design point is corresponding to one combination of optimization passes.

After the completion of all conversions and optimizations, the generated IR is translated into synthesizable C++ code by ScaleHLS C++ emitter and then passed to external HLS tools for generating RTL code. Meanwhile, flattened LLVM IR can also be generated for the purpose of software simulation or passing to other existing LLVM-compatible HLS tools.

3 INITIAL EXPERIMENTAL RESULTS

We evaluate the ScaleHLS DSE engine on 6 computation kernels with a problem size of 2048 and the results are shown in Table 1. Xilinx Vivado HLS 2019.1 is used to generate the RTL code and the target platform is Xilinx XC7Z020 FPGA, which has 220 DSP slices and 4.9 Mb memories on chip. Table 1 lists the optimal unroll factors and pipeline initial intervals (II) discovered by the DSE engine. Loop perfection, loop order permutation, variable loop bound elimination, and array partition are also automatically applied for improving the design quality. Among all 6 benchmarks, a speedup ranging from

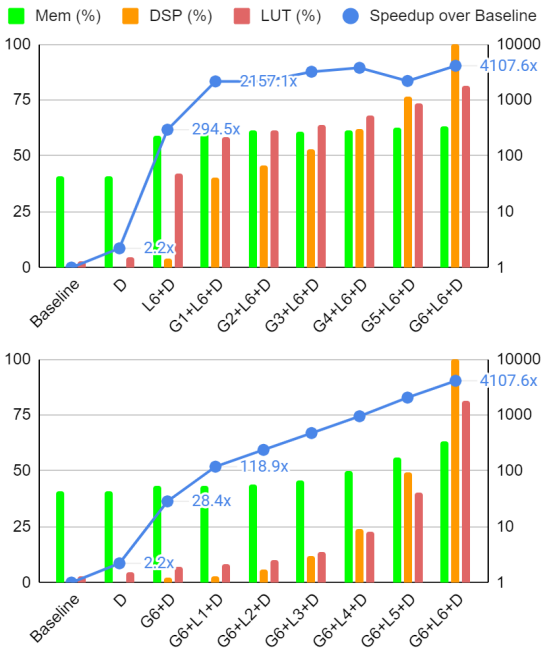

Figure 2: Ablation study results of MobileNet-v2. D, $L\{n\}$, and $G\{n\}$ denote directive, loop, and graph level optimizations, respectively. Larger n indicates stronger optimization.

83.3× to 768.2× is obtained compared to the baseline design which is only processed by the regular LLVM optimizations of Vivado HLS. As previous DSE methods [14, 15] only supports single-level abstraction, they are difficult to find reasonable design points when the problem sizes are large. The proposed multi-level representation enables ScaleHLS to find previously unachievable design points and explore a more comprehensive design space.

To evaluate the ScaleHLS framework when handling large and complicated HLS designs, we take a MobileNet-v2 [12] PyTorch model as the test case and Xilinx VU9P FPGA which has 6840 DSP slices and 345.9 Mb memories as the target platform. To quantify the speedup contributed by each of the three optimizations (directive, loop, and graph) and evaluate the proposed multi-level optimization methodology, we perform ablation studies and the results are shown in Figure 2. From the data of D, $L6+D$, and $G6+D$, we can observe that the directive, loop, and graph optimizations contribute around 2.2x, 133.9x, and 12.9x speedups, respectively. By combining all three optimizations, the generated RTL design achieves a 4107.6× speedup compared to the baseline only optimized by Vivado HLS. Furthermore, ScaleHLS allows to tune the optimization level n from 1 to 6 for loop and graph optimizations, which enables to explore the trade-off space between area and speedup. The experimental results show that through the effective representation and optimization of large HLS designs in MLIR, ScaleHLS can significantly improve the hardware performance, resource efficiency, and the productivity of designing HLS-based hardware accelerators.

ACKNOWLEDGMENTS

We thank Eric Cheng of Laboratory for Physical Sciences (LPS), Stephen Neuendorffer of Xilinx, and Samuel Bayliss of Xilinx for insightful discussions. This work is supported by LPS and Xilinx.

REFERENCES

[1] Yao Chen, Jiong He, Xiaofan Zhang, Cong Hao, and Deming Chen. 2019. CloudDNN: An open framework for mapping DNN models to cloud FPGAs. In *Proceedings of the 2019 ACM/SIGDA international symposium on field-programmable gate arrays*. 73–82.

[2] MLIR contributors. 2021. MLIR: Multi-Level Intermediate Representation. https://github.com/llvm/llvm-project/tree/main/mlir.

[3] ONNX contributors. 2021. ONNX: Open Neural Network Exchange. https://github.com/onnx/onnx.

[4] Steve Dai, Yuan Zhou, Hang Zhang, Ecenur Ustun, Evangeline FY Young, and Zhiru Zhang. 2018. Fast and accurate estimation of quality of results in high-level synthesis with machine learning. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 129–132.

[5] Lorenzo Ferretti, Giovanni Ansaloni, and Laura Pozzi. 2018. Lattice-traversing design space exploration for high level synthesis. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*. IEEE, 210–217.

[6] Chris Lattner, Jacques Pienaar, Mehdi Amini, Uday Bondhugula, River Riddle, Albert Cohen, Tatiana Shpeisman, Andy Davis, Nicolas Vasilache, and Oleksandr Zinenko. 2020. MLIR: A Compiler Infrastructure for the End of Moore's Law. *arXiv preprint arXiv:2002.11054* (2020).

[7] Xinheng Liu, Yao Chen, Tan Nguyen, Swathi Gurumani, Kyle Rupnow, and Deming Chen. 2016. High level synthesis of complex applications: An H. 264 video decoder. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 224–233.

[8] Hosein Mohammadi Makrani, Farnoud Farahmand, Hossein Sayadi, Sara Bondi, Sai Manoj Pudukotai Dinakarrao, Houman Homayoun, and Setareh Rafatirad. 2019. Pyramid: Machine Learning Framework to Estimate the Optimal Timing and Resource Usage of a High-Level Synthesis Design. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 397–403.

[9] Kenneth O'Neal, Mitch Liu, Hans Tang, Amin Kalantar, Kennen DeRenard, and Philip Brisk. 2018. Hlspredict: Cross platform performance prediction for fpga high-level synthesis. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[10] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*. 8026–8037.

[11] Kyle Rupnow, Yun Liang, Yinan Li, and Deming Chen. 2011. A study of high-level synthesis: Promises and challenges. In *2011 9th IEEE International Conference on ASIC*. IEEE, 1102–1105.

[12] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[13] Xiaofan Zhang, Anand Ramachandran, Chuanhao Zhuge, Di He, Wei Zuo, Zuofu Cheng, Kyle Rupnow, and Deming Chen. 2017. Machine learning on FPGAs to face the IoT revolution. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 894–901.

[14] Jieru Zhao, Liang Feng, Sharad Sinha, Wei Zhang, Yun Liang, and Bingsheng He. 2017. COMBA: A comprehensive model-based analysis framework for high level synthesis of real applications. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 430–437.

[15] Guanwen Zhong, Alok Prakash, Yun Liang, Tulika Mitra, and Smail Niar. 2016. Lin-analyzer: a high-level performance analysis tool for FPGA-based accelerators. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[16] Wei Zuo, Warren Kemmerer, Jong Bin Lim, Louis-Noël Pouchet, Andrey Ayupov, Taemin Kim, Kyungtae Han, and Deming Chen. 2015. A polyhedral-based SystemC modeling and generation framework for effective low-power design space exploration. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 357–364.

[17] Wei Zuo, Yun Liang, Peng Li, Kyle Rupnow, Deming Chen, and Jason Cong. 2013. Improving high level synthesis optimization opportunity through polyhedral transformations. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. 9–18.

[18] Wei Zuo, Louis-Noel Pouchet, Andrey Ayupov, Taemin Kim, Chung-Wei Lin, Shinichi Shiraishi, and Deming Chen. 2017. Accurate high-level modeling and automated hardware/software co-design for effective SoC design space exploration. In *Proceedings of the 54th Annual Design Automation Conference 2017*. 1–6.