LAMDA: Learning-Assisted Multi-Stage Autotuning for FPGA Design Closure

Ecenur Ustun*, Shaojie Xiang, Jinny Gui, Cunxi Yu*, and Zhiru Zhang*
School of Electrical and Computer Engineering, Cornell University, Ithaca, NY, USA
{eu49, cunxi.yu, zhiruz}@cornell.edu

***Abstract*—A primary barrier to rapid hardware specialization with FPGAs stems from weak guarantees of existing CAD tools on achieving design closure. Current methodologies require extensive manual efforts to configure a large set of options across multiple stages of the toolflow, intended to achieve high quality-of-results. Due to the size and complexity of the design space spanned by these options, coupled with the time-consuming evaluation of each design point, exploration for reconfigurable computing has become remarkably challenging. To tackle this challenge, we present a learning-assisted autotuning framework called LAMDA, which accelerates FPGA design closure by utilizing design-specific features extracted from early stages of the design flow to guide the tuning process with significant runtime savings. LAMDA automatically configures logic synthesis, technology mapping, placement, and routing to achieve design closure efficiently. Compared with a state-of-the-art FPGA-targeted autotuning system, LAMDA realizes faster timing closure on various realistic benchmarks using Intel Quartus Pro.**

I. Introduction

Limitations in technology scaling have led to a growing interest in non-traditional system architectures incorporating specialized hardware accelerators for improved performance and energy efficiency. Although FPGAs have shown to be a significant potential in hardware specialization [1], weak guarantees of existing CAD tools on achieving design closure out-of-the-box is a main barrier to its adaptation. To achieve high quality-of-results (QoR), CAD tools require huge manual effort to configure a large set of design and tool parameters.

To meet the diverse requirements of a broad range of application domains, FPGA development environments commonly provide users with an extensive set of tool options. For instance, synthesis and place-and-route (PnR) options in Intel Quartus Pro translate to a search space of over 1.8×10^{24} design points. Fig. 1 shows 500 design points randomly sampled from possible combinations of tool options of Intel Quartus Pro, and the resulting critical path delays. Results of default tool options are included for reference. There are two important observations: (1) default timing results are on average 15% higher than the best results of the random samples; (2) more than 30% of the randomly sampled configurations produce better timing than the default configurations. This suggests that there is considerable room for timing improvement by tuning tool options. However, exploring a large number of tool options is extremely inefficient and cannot be effectively carried out by human effort alone. A similar challenge exists also in high-performance computing and software compilation, e.g., autotuners have been developed for automatically optimizing compiler configurations [2].

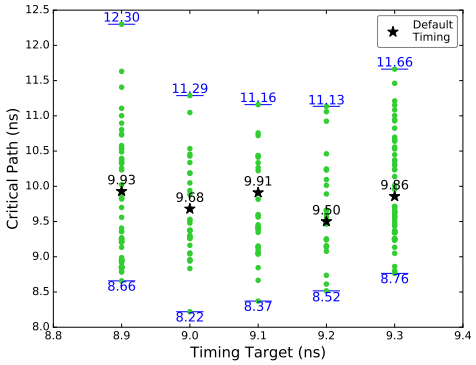


Fig. 1. Timing distribution of `bfly` for various tool settings and timing constraints – x-axis represents target clock period (ns), and y-axis represents critical path delay (ns).

Recent years have seen an increasing employment of machine learning (ML) in EDA to enable rapid design space exploration (DSE) [3]–[7] and automatic configuration of design or tool parameters [2], [8], [9]. However, there are two major limitations with the existing approaches. First, current techniques mainly focus on a single stage of the design flow such as high-level synthesis (HLS) [4] or logic synthesis [5], thereby missing important cross-boundary optimization opportunities. Second, existing methods often use pre-PnR or even pre-synthesis reports for assessing the quality of a design point [10]. While this shortens the execution times, simply relying on crude estimates from an early design stage may prevent DSE from reaching high-quality design points.

To address the aforementioned limitations, we propose LAMDA, a Learning-Assisted Multi-stage Design Autotuning framework that accelerates FPGA design closure. We develop a multi-stage QoR inference model based on online supervised learning, which allows LAMDA to effectively detect and prune unpromising design points over search spaces. LAMDA automatically configures a wide range of CAD tool options through balancing the trade-off between computing effort and estimation accuracy. Our main technical contributions include:

- An ML-based multi-stage autotuner, which leverages features from early stages to estimate post-PnR QoR.
- LAMDA achieves faster design closure using online learning—design points visited during autotuning are used to further increase the ML model accuracy.
- LAMDA achieves 5.43× speedup compared to a state-of-the-art FPGA-targeted autotuning system for multiple realistic designs using Intel Quartus Pro.

- Emulation databases of five realistic designs using Intel Quartus Pro, which enables fast autotuning evaluation. The databases will be open sourced to facilitate further research of autotuning algorithms and tools.

II. Background

Mainstream FPGA compilation flow takes untimed C++/OpenCL or an RTL design as input and generates a device-specific bitstream. This process involves several distinct and modular steps including HLS, logic synthesis, technology mapping, packing, placement, and routing. Each step provides designers a set of configuration switches that select between different heuristics or influence the behavior of a heuristic. These switches need to be calibrated with significant manual effort and expert knowledge to achieve desired QoRs. Due to the lack of predictability and time-consuming FPGA design flow, there is an urgent need to lower the design cost by minimizing human supervision and significantly reducing the time required to obtain accurate QoR estimation.

Autotuning has been used for optimization in FPGA-targeted CAD toolflow by automatically configuring the parameters and tool options to optimize certain objective functions. InTime [8], [11], [12] explores supervised learning techniques to accelerate FPGA timing closure. It automatically selects tools options for a given design by exploring the design space using a timing estimator. DATuner [13] utilizes the multi-armed bandit technique to automatically tune the options for a complete FPGA compilation flow.

Note that these are single-stage autotuning frameworks, meaning that they are based on QoR estimation conducted on features of a single stage. Single-stage autotuning with QoR estimation conducted at a late design stage is time-consuming because each iteration runs through PnR. On the other hand, using early-stage features for assessing the quality of a design point during autotuning shortens runtime. However, simply relying on crude estimates from an early stage may prevent the CAD tool from applying the appropriate set of optimizations, resulting in sub-optimal trade-offs. Lo et al. proposed a multi-fidelity approach for tuning HLS parameters, incorporating features across HLS, synthesis, and implementation stages [14]. This paper demonstrates how multi-stage approach can significantly reduce tuning runtime, and early-stage features (e.g., synthesis features) are particularly informative.

To understand the trade-off between computing effort and estimation accuracy, we show the potential speedups while using features collected from different stages in Fig. 2. Taking the runtime of the single-stage autotuning as the baseline, $5\times$ speedup can be achieved by estimating post-routing results based on the features collected before routing, and $10\times$ speedup can be achieved with features collected before placement. Using features of earlier stages could provide significantly more speedups (e.g., $175\times$ using pre-packing features). However, it is likely to introduce negative effects for design closure because early stage features have limited correlations with post-PnR results. To balance this trade-off, we propose a ML-based multi-stage autotuning framework.

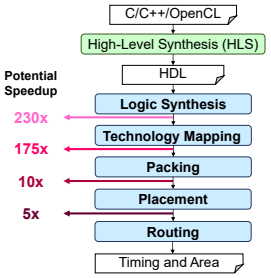


Fig. 2. Potential speedups by leveraging QoR inference at different stages

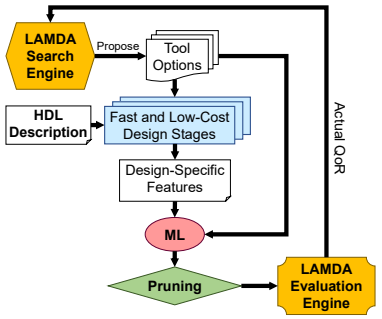


Fig. 3. Overview of LAMDA

III. Approach

The overall autotuning flow of LAMDA is illustrated in Fig. 3. It takes an HDL description as input and automatically configures the tool options across logic synthesis, technology mapping, packing, and PnR stages, where search space is defined by extensive tool options. Table I lists a subset of tunable tool options of Intel Quartus Pro. LAMDA leverages a highly accurate ML model to effectively prune the design space, thus accelerating FPGA design closure. The rest of this section describes key components of LAMDA in more detail.

Multi-stage QoR inference: We develop a multi-stage inference model that estimates post-PnR results based on tool features (configurations of the tool options) and design-specific features. Using multi-stage design-specific features is one of the main contributions compared to InTime and DATuner. As discussed in Section II, collecting early stage features is fast, but could lack QoR estimation accuracy. Collecting features from later stages is more informative, yet time-consuming. Therefore, *fast and low-cost design stages* in Fig. 3 need to be carefully selected to balance accuracy-runtime trade-off. To this end, we analyze the effects of features in Table II, from which one can draw three conclusions. First, design-specific features help estimate QoR more accurately compared to using tool options only (i.e. pre-synthesis). Second, accuracy increases as features from later stages are included in the feature set, bringing about an accuracy-runtime trade-off. Third, although tool estimates are less accurate under tight constraints, design-specific features still help improve estimation accuracy compared to using tool options only. To balance the aforesaid trade-off, LAMDA assigns technology mapping and packing as the *fast and low-cost design stages*. Table III shows design-specific features for these stages.

Online learning: Many of the existing autotuning and DSE approaches explore supervised learning and train an ML model using static dataset, also called offline dataset. Simply relying on offline training is not realistic for FPGA design autotuning problems due to the lack of labelled data and the cost of collecting end-to-end data. To balance the data collection efforts and the estimation accuracy, we propose to use online learning to train the multi-stage ML model. The idea is to first perform offline training using a small number of static data points. ML model is periodically updated with newly collected data points throughout the autotuning process.

TABLE I
A SUBSET OF TUNABLE INTEL QUARTUS PRO TOOL OPTIONS

Stage	Options	Values
Logic Synthesis, Technology Mapping, Packing	auto_dsp_recognition, timing_driven_synthesis	{On, Off}
	disable_register_merging, mux_restructure	{Auto, On, Off}
	optimization_technique	{Area, Speed, Balanced}
	synthesis_effort	{Auto, Fast}
Placement & Routing	fitter_effort	{Standard Fit, Auto Fit}
	final_place_optimization, routability_optimization	{Always, Never, Automatically}
	register_packing_effort	{High, Low, Medium}
	allow_register_retiming, auto_delay_chains	{On, Off}
	route_timing_optimization	Normal, Maximum, Minimum

TABLE II
EFFECT OF FEATURES ON TIMING ESTIMATION – RMSE ON BFLY FOR DIFFERENT FEATURE SETS, AND TIGHT (5.0 NS) & MODERATE (6.8 NS) CONSTRAINTS; FOR A TRAINING SET OF SIZE 100.

Features	Target (ns)	
	5.0	6.8
Pre-synthesis	0.72	0.46
Pre-place	0.63	0.35
Pre-route	0.60	0.29

TABLE III
DESIGN-SPECIFIC FEATURES OF INTEL QUARTUS PRO

Design Stage	Type	Features
Technology Mapping and Packing	Resource	#ALM, #LUT, #registers, #DSP, #I/O pins, #fan-out, etc.
	Timing	WS, TNS

Emulation database: We exhaustively collect all design points of a given design space defined by a set of tool options and a specific design. Evaluation process of autotuning then can be emulated by performing a look up on the exhaustive database, which significantly reduces the evaluation runtime for autotuning FPGA design closure. These databases are referred to as *emulation databases*. This enables researchers to quickly evaluate various autotuning approaches and tune the parameters of ML models. The specific emulation databases used in this work are described in Section IV.

Implementation: Let n be the number of tool settings proposed at i^{th} iteration. ML regressor is used to estimate QoR of these n proposals based on tool features and design-specific features from early stages. Top m proposals ($m < n$) with the best estimated QoR are selected. A random subset of these m proposals is generated with a rate R ($0 < R \leq 1$) such that $R \cdot m$ proposals are validated through end-to-end FPGA toolflow. If any of these design points meets objective, autotuning process terminates. Otherwise, they are used for online learning, and autotuner proceeds with $i + 1^{th}$ iteration.

In order to choose a suitable ML routine, we compared XGBoost [15] with InTime's ML routines, which perform binary classification of design points with respect to TNS [11]. For evaluation, we used InTime's publicly available datasets. The results in Table IV show that XGBoost performs better than InTime for six designs. For the remaining three designs, XGBoost and InTime perform similarly. As a consequence, LAMDA employs XGBoost for QoR estimation.

TABLE IV
INTIME AND XGBOOST COMPARISON – ACCURACY AND F1 SCORES FOR INTIME'S ML ROUTINES VS. XGBOOST ON INTIME'S DATASETS.

	Accuracy		F1 Score	
	InTime	XGB	InTime	XGB
aes-128	0.82	0.86	0.82	0.86
dec-viterbi	0.71	0.73	0.74	0.73
mkSwitch	0.78	0.88	0.78	0.88
vga-enh-top	0.65	0.77	0.73	0.77
xge-mac	0.60	0.74	0.62	0.74
eight-bit	0.69	0.78	0.71	0.78
flow	0.56	0.77	0.63	0.77
SOC	0.78	0.79	0.79	0.79
VIP	0.82	0.82	0.82	0.82

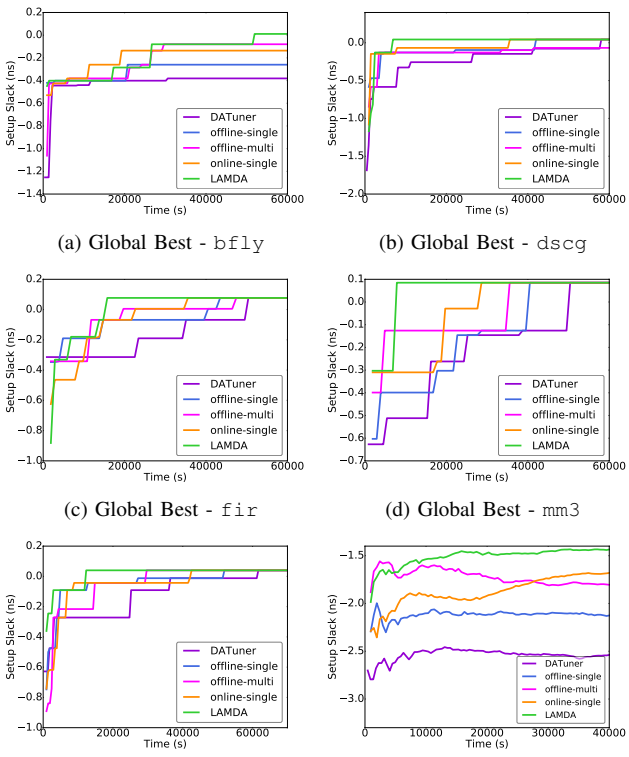
IV. EVALUATION

We evaluate LAMDA by targeting timing closure of five realistic benchmarks listed in Table V, using Intel Quartus 17.1.0 Pro Edition targeting Arria 10. To demonstrate the effectiveness of multi-stage and online learning separately, we compare four ML-based autotuning modes, namely *online-multi* (i.e. LAMDA), *online-single*, *offline-multi*, and *offline-single*. Here, *single* represents estimating timing based on only tool options; *multi* represents leveraging design-specific features from early stages; *online* represents online learning; and *offline* represents offline learning. We implemented *online-single* to quantify one of the main differences between InTime and LAMDA, which is leveraging early design stages. We also compare with DATuner [13], a state-of-the-art open-source FPGA autotuner. Experimental results are obtained using a machine with a 10-core Intel Xeon operating at 2.5 GHz.

LAMDA is built based on OpenTuner [2], while we apply multi-stage learning to guide the autotuning process. Specifically, XGBoost regressor is used to estimate timing from tool and design-specific features. At the first iteration of autotuning, XGBoost regressor is trained on the offline dataset. Given an optimization objective, LAMDA picks a set of unseen configurations proposed by OpenTuner. Design-specific features are collected from technology mapping and packing stages after running the toolflow up to packing. Proposed configurations are ranked with respect to ML inference results. Top-ranking configurations are evaluated through the complete FPGA toolflow to obtain the actual QoRs, whereas the remaining configurations are pruned away. Data obtained from evaluation are used to update OpenTuner's search engine and the ML model iteratively. Autotuning process terminates when one of the newly proposed configurations meets timing, or timeout is reached. Collecting design-specific features and the evaluation process are parallelized to speed up the autotuning process.

TABLE V
BENCHMARKS TARGETING ARRIA 10 – POST-PNR UTILIZATION WITH DEFAULT TOOL OPTIONS; TARGET INDICATES TIMING CONSTRAINT.

	#ALUT	#FF	#DSP	Target (ns)
bfly	6348	1868	4	6.8
dscg	6246	1679	4	7.0
fir	5753	1648	4	8.1
mm3	4001	1023	3	8.0
ode	4118	1104	2	8.1



(a) Global Best - `bfly` (b) Global Best - `dscg`

(c) Global Best - `fir` (d) Global Best - `mm3`

(e) Global Best - `ode` (f) Global Mean - `dscg`

Fig. 4. Timing closure of `bfly`, `dscg`, `fir`, `mm3`, and `ode` using DATuner, *offline-single*, *offline-multi*, *online-single*, and LAMDA – (a)-(e) Best timing results achieved over time. (f) Mean of all timing results visited over time.

Data collection: We construct emulation databases of five designs listed in Table V by choosing ten different tool options that cover logic synthesis, technology mapping, packing, and PnR. Each database includes 5184 data points. Experimental results in the next section are obtained by emulating the autotuning process through looking up these databases.

Timing closure: Fig. 4 demonstrates that LAMDA outperforms DATuner with a factor of 5.43× and *offline-single* with a factor of 4.38×, averaged over four benchmarks; while for the fifth benchmark (`bfly`), other autotuners cannot achieve LAMDA's QoR. LAMDA and *online-single* perform better than DATuner due to learning-assisted pruning. Since offline training set size is not sufficient to accurately estimate timing, *offline-multi* and *offline-single* prune the design space based on inaccurate QoR estimations, resulting in worse performance compared to LAMDA and *online-single*, respectively. This suggests that online learning increases autotuning efficiency. LAMDA performs better than *online-single*, indicating that design-specific features help guide the autotuning process effectively. Fig. 4f plots the average QoR visited by each autotuner over time. There are two observations echoing aforementioned conclusions: ***Effectiveness of multi-stage learning***—average QoR of LAMDA is always higher than *online-single*; ***Effectiveness of online learning***—average QoR of LAMDA and *online-single* increase over time, and converge to a higher value than *offline-multi* and *offline-single*, respectively.

V. Conclusions

This paper presents LAMDA, a novel learning-assisted FPGA autotuning framework, that leverages online learning from a multi-stage perspective. LAMDA learns the timing behavior of a design from tool options (applied across all design stages) as well as the design-specific features (collected from fast and low-cost design stages). By means of its accurate timing estimator, LAMDA successfully handles cross-stage optimizations, leading to a high efficiency. The paper introduces emulation databases of 5 designs to help researchers quickly evaluate autotuning approaches. The emulation databases will be released as open source contribution.

Acknowledgment

We would like to thank Dr. Aravind Dasu, Bain Syrowik, and Kevin Stevens from Intel Corp.; and the anonymous reviewers for their invaluable feedback. This work was supported in part by Intel Strategic Research Alliance (ISRA) Program.

References

[1] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu *et al.*, "Serving DNNs in Real Time at Datacenter Scale with Project Brainwave," *IEEE Micro*, 2018.

[2] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe, "OpenTuner: An Extensible Framework for Program Autotuning," *Int'l Conf. on Parallel Architectures and Compilation (PACT)*, 2014.

[3] D. Li, S. Yao, Y. Liu, S. Wang, and X. Sun, "Efficient Design Space Exploration via Statistical Sampling and AdaBoost Learning," *Design Automation Conference (DAC)*, 2016.

[4] H. Liu and L. P. Carloni, "On Learning-Based Methods for Design-Space Exploration with High-Level Synthesis," *Design Automation Conference (DAC)*, 2013.

[5] C. Yu, H. Xiao, and G. D. Micheli, "Developing Synthesis Flows Without Human Knowledge," *Design Automation Conference (DAC)*, 2018.

[6] S. Kang and R. Kumar, "Magellan: A Search and Machine Learning-based Framework for Fast Multi-core Design Space Exploration and Optimization," *Design, Automation and Test in Europe (DATE)*, 2008.

[7] W. Jia, K. A. Shaw, and M. Martonosi, "Stargazer: Automated Regression-Based GPU Design Space Exploration," *Int'l Symp. on Performance Analysis of Systems & Software (ISPASS)*, 2012.

[8] N. Kapre, H. Ng, K. Teo, and J. Naude, "InTime: A Machine Learning Approach for Efficient Selection of FPGA CAD Tool Parameters," *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2015.

[9] M. M. Ziegler, R. B. Monfort, A. Buyuktosunoglu, and P. Bose, "Machine Learning Techniques for Taming the Complexity of Modern Hardware Design," *IBM Journal of Research and Development*, 2017.

[10] S. Dai, Y. Zhou, H. Zhang, E. Ustun, E. F. Young, and Z. Zhang, "Fast and Accurate Estimation of Quality of Results in High-Level Synthesis with Machine Learning," *IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2018.

[11] Q. Yanghua, H. Ng, and N. Kapre, "Boosting Convergence of Timing Closure using Feature Selection in a Learning-driven Approach," *Int'l Conf. on Field Programmable Logic and Applications (FPL)*, 2016.

[12] Q. Yanghua, C. A. Raj, H. Ng, K. Teo, and N. Kapre, "Case for Design-Specific Machine Learning in Timing Closure of FPGA Designs," *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2016.

[13] C. Xu, G. Liu, R. Zhao, S. Yang, G. Luo, and Z. Zhang, "A Parallel Bandit-Based Approach for Autotuning FPGA Compilation," *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2017.

[14] C. Lo and P. Chow, "Multi-Fidelity Optimization for High-Level Synthesis Directives," *Int'l Conf. on Field Programmable Logic and Applications (FPL)*, 2018.

[15] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Int'l Conf. on Knowledge Discovery and Data Mining (KDD)*, 2016.