Multi-Fidelity Optimization for High-Level Synthesis Directives

Charles Lo and Paul Chow
Department of Electrical and Computer Engineering, University of Toronto,
Toronto, ON, Canada
Email: {locharl1,pc}@eecg.toronto.edu

***Abstract*—High-Level Synthesis (HLS) tools enable rapid hardware development, but design expertise and effort are necessary to tune the high-level descriptions into optimized circuits. To improve designer productivity, automated design-space exploration techniques have been proposed. However, the optimization processes sample expensive CAD flows. In this paper, we adapt multi-fidelity optimization methods to incorporate low-fidelity estimates available in the FPGA CAD flow and speed up tuning of HLS parameters. We find that multi-fidelity optimization techniques can significantly reduce optimization time compared to previous approaches.**

I. Introduction

High-Level Synthesis (HLS) tools allow designs to be described in untimed, compact and easily verified languages such as C and C++. This reduces the design effort needed to create complex hardware systems. Previous studies [1], [2] have shown that, with optimization, the resulting hardware can approach the quality of designs hand-written in traditional hardware description languages (HDLs).

Although HLS systems facilitate rapid development of functional hardware, considerable effort and expertise are still required to obtain optimized designs. The final hardware architecture generated from a C/C++ description can vary dramatically in terms of area and performance depending on the application of *directives* in the HLS flow. For instance, a directive may cause an array construct to be partitioned into multiple RAMs to increase bandwidth at the cost of area. Multiple directives often interact when applied to the same or related sections of code resulting in complex effects on the final area or latency of the generated hardware. In addition, a single setting of directives may not be appropriate for an HLS design. An HLS project can be considered as a re-usable intellectual property (IP) block that may be used in many different contexts and, depending on the goals and constraints of an application, different sets of directives may be applied.

To help designers select directive settings, previous works [3], [4], [5] have explored the use of automated exploration methods. These works used automated techniques to find Pareto-optimal designs or fine-tune implementations to minimize a particular objective function. It was demonstrated that the automated techniques could find good designs by sampling only a small portion of the full design space and without human supervision. Although a small number of samples are required when using these methods, finding the best design still requires evaluating each sample through computationally expensive parts of the computer-aided design (CAD) flow.

To find the best directive setting in terms of clock frequency or area, it is necessary to evaluate a design after all of the optimizations have been applied and after placement and routing. However, at several points in the CAD flow, estimates of varying *fidelity* are provided to designers that help them make decisions earlier and avoid running the full set of processes. For instance, the HLS tool itself may provide low-fidelity area estimates that give a designer intuition about the size of the circuit. If automatic optimization is performed using these estimates alone, the process could choose a non-optimal or infeasible design. However, multiple estimates at different fidelities provide valuable information that can help guide an automated tool toward the final design faster. Previous work [6] using multiple fidelities has considered how to select candidate designs for Logic Synthesis using HLS estimates when constructing Pareto-optimal sets of designs. In this work, we examine how the HLS and Synthesis estimates can help guide an optimizer to efficiently find optimal placed and routed designs when performing direct optimization of HLS directives. We make use of Sequential Model-Based Optimization to explore the design space and augment it with information from the estimates available in an HLS CAD flow targeting Field-Programmable Gate Arrays (FPGAs).

The rest of the paper is organized as follows: Related work on parameter tuning and multi-fidelity optimization is discussed in Section II. The relationship between estimates available at different fidelities is described in Section III. The model used to incorporate estimates from multiple metrics and fidelities is presented in Section IV and multi-fidelity optimization techniques are described in Section V. Results are discussed in Section VI and final conclusions are presented in Section VII.

II. Related Work

Automated parameter tuning of hardware and multi-fidelity optimization have been explored in previous research. In this section, we provide a brief summary of related works in the context of this paper.

A. Parameter Tuning

A number of previous studies have described methods for automatically exploring the design spaces provided by HLS.

Due to the variety of possible directive settings as well as the time required to run the CAD flow and evaluate a single setting, these works made use of *surrogate models* of the design space. For instance, Liu et al. [5] employed Random Forests to model the design spaces and guide iterative refinement of Pareto-optimal sets of designs. Meng et al. [7] also used Random Forests to model performance and logic utilization of systems generated by OpenCL-based design tools, but used a new strategy to speed up construction of the Pareto set. Mahapatra et al. [4] employed decision trees to guide a simulated annealing process to find Pareto-optimal designs. Zuluaga et al. [8] modeled objectives using Gaussian processes and used the predictive uncertainty provided by the model to intelligently search for Pareto-optimal points.

Finding a Pareto set can provide good general design choices to end users. However, a user of an IP generated via HLS may have a number of dimensions of concern when determining optimality such as utilization of specific resources (e.g. Block RAMs and Look-Up Tables (LUTs) when targeting FPGAs). Further, the exact values of metrics such as clock frequency could depend on how the IP will be integrated into a larger system. Lo et al. [3] made use of a Gaussian process model in a Sequential Model-Based Optimization flow to optimize user-defined objective functions. To improve optimization speed, they embedded knowledge of the design space in their formulation. Another technique for embedding domain knowledge into an optimizer was explored in the context of IP parameter tuning by Papamichael et al. [9]. In this work, we also consider direct optimization problems rather than searching for Pareto sets. However, we aim to improve the optimization runtime of previous methods by incorporating information from CAD tool estimates.

B. Multi-Fidelity Optimization

The idea of using faster, lower-fidelity estimates during optimization is a well-developed idea, especially in the computer simulation realm. When simulating physical systems, often the accuracy of a computer simulation model is related to the required runtime. To take advantage of faster running, but lower fidelity simulations, techniques were developed [10] to align fast, low-fidelity models with higher fidelity but slower models. By performing this mapping, optimization could proceed using primarily outputs from the faster, low-fidelity model.

In the hardware design domain, Todman et al. [11] outlined this approach using a low-fidelity, fast model for exploration with feedback from post-implementation results. Liu et al. [6] considered constructing Pareto curves using HLS estimates and efficiently sampling higher-fidelity, post-Synthesis values when necessary to obtain more accurate estimates. In contrast, we consider direct optimization and use values from HLS and Synthesis processes to estimate placed and routed design metrics.

Kennedy and O'Hagan [12] described a Bayesian approach of modelling arbitrary levels of computer simulation fidelities. Using a probabilistic technique to model the relationship between outputs at different fidelities, their formulation is also able to incorporate uncertainty of outputs. Huang et al. [13] described how to use this model to reduce overall optimization time by dynamically selecting both design parameters and fidelities to explore. Forrester et al. [14] also made use of the model proposed by Kennedy and O'Hagan, for optimization. In contrast to Huang et al., they initially sampled points at low fidelities to guide optimization at the highest fidelity. More recently, Le Gratiet et al. [15] described an equivalent recursive formulation of the model that can be more efficiently computed for large sets of data.

In this work, we make use of the original model by Kennedy and O'Hagan to perform optimization. However rather than model a single computer simulation output, we model combinations of multiple metrics available in an HLS CAD flow such as the area-delay product. We consider adaptations of the optimization methods presented by Huang et al. [13] and Forrester et al. [14] for constrained HLS optimization as well as introduce a new early-stopping heuristic for dynamically selecting fidelities to sample during optimization.

III. Fidelities in FPGA-Based HLS

Several fidelities of estimates are available in the FPGA HLS CAD flow. In this section, we consider the relation of these estimates to their final values.

A. FPGA HLS Design Flow

There are a number of steps required to take a design specified in high-level C/C++ code to a bitstream representing the placed and routed design on a specific FPGA platform. In this paper, we consider the Xilinx Vivado CAD flow, although the techniques are also applicable to other vendors and design methodologies. The major stages in the Vivado design flow are: 1) High-Level Synthesis (HLS), 2) Synthesis and 3) Implementation. At each stage, the design is refined and new estimates of area and delay are available.

The process of *High-Level Synthesis* converts the combination of C/C++ source code and directives to a set of HDL files describing the Register Transfer Level (RTL) hardware. Scheduling and binding of operations is performed in this stage; thus, the number of cycles can be determined via the schedule provided by the tool or through simulation. Preliminary area and delay estimates are also available but can be inaccurate.

During *Synthesis*, the RTL is optimized and mapped into the target FPGA technology primitives. At this stage, area utilization estimates are much more accurate. However, since the physical location of components is not yet determined, delay estimates are still poor. To obtain final values for achievable clock period and area, the steps in *Implementation* must be performed. In this process, final optimization, placement and routing is carried out and a bitstream may be generated. For the purposes of fine-tuning the area or clock frequency of a design, the results at this stage are the final values.

TABLE I: Average execution time of fidelities in seconds

Benchmark	HLS	Synthesis	Implementation
AES	13	159	179
Backpropagation	19	562	625
Radix Sort	9	152	183
ADPCM	37	438	402

B. Comparison of Tool Estimates

To understand the relationship between the estimates available at different parts of the design flow, a large number of designs with different directive settings were implemented using the Vivado 2016.3 tools targeting a Virtex 7 485T-1 device. Three benchmarks, AES, Backpropagation and Radix Sort were chosen from the MachSuite [16] applications as well as ADPCM from the CHStone benchmarks [17]. These benchmarks were chosen since they are amenable to the application of many interacting directives. We applied combinations of the influential directives: loop pipelining, unrolling and memory partitioning. In total, the sizes of the design spaces were 55,566, 23,253, 4,096 and 17,024 for AES, Backpropagation, Radix Sort and ADPCM respectively. The quality of the estimates and final values will depend on a number of factors such as the synthesis optimizations applied, the selected clock frequency and the target FPGA. In this paper, a target clock period of 5ns was chosen with default uncertainty values and no extra implementation settings were applied. These settings are the same as those applied when a designer chooses to run the built-in RTL evaluation available in Vivado HLS.

All of the designs were run through High-Level Synthesis, Synthesis and Implementation on the SciNet [18] General Purpose Cluster nodes. The runtime of each phase of the CAD flow was also collected. In this paper, we use the average runtimes for each fidelity as a constant, independent of the directive setting due to the variability of the tool runtime on our supercomputer cluster. These average runtime values are reported in Table I. As can be seen from the table, the synthesis and implementation stages require about the same amount of time to execute and make up the majority of the tool flow time relative to the HLS process.

Collecting the full design space was important to ensure that true global minimum values could be obtained for each optimization task and thus could be compared to an optimizer's progress. However, as can be seen from even the smallest Radix Sort benchmark, it would take many CPU-days to fully explore directive design spaces in general.

For each benchmark, we compared the LUT and Flip Flop (FF) area as well as clock period estimates after HLS and Synthesis to the post-Implementation values. The symmetric mean absolute percent errors as well as the Spearman correlation coefficients between the estimates and the final values are shown in Table II. The HLS tool has varying success in estimating the area metrics depending on the benchmark. For instance, the estimate of flip flop usage is within a few percentage points for AES and Radix Sort, but LUT usage is poorly estimated. Once logic optimization is completed during the Synthesis phase of the CAD flow, the area metrics

TABLE II: Symmetric mean absolute percent errors and correlation coefficients of area and clock estimates relative to post-implementation values

Metric	Post-HLS		Post-Synthesis	
	% Err.	Correl.	% Err.	Correl
AES				
LUT	79.3	0.64	0.4	1.00
FF	4.5	0.99	0.0	1.00
Clock Period	18.4	-0.14	17.7	0.34
Backpropagation				
LUT	47.7	0.82	3.9	1.00
FF	9.8	1.00	1.4	1.00
Clock Period	6.0	-0.11	8.1	0.23
Radix Sort				
LUT	54.8	0.96	14.8	1.00
FF	5.7	1.00	0.0	1.00
Clock Period	51.8	0.06	14.8	0.30
ADPCM				
LUT	43.9	0.89	5.1	0.99
FF	34.8	0.81	0.1	1.00
Clock Period	8.9	0.23	11.1	0.48

are estimated more accurately. Although the percent error of the estimates can be large, these early values provide useful information for an optimization process. This is demonstrated by the generally strong correlation between estimated area values and the final post-Implementation results. Minimizing a function correlated to the desired objective will find designs close to the true minimum values, but a low error is still important when a metric is part of an optimization constraint to ensure points are feasible.

In contrast to the area measures, the clock period is poorly estimated until place and route is completed in the Implementation phase of the CAD flow. Although the target clock frequency can be set in the HLS tool, it is necessary to consider the post-Implementation values for a number of reasons. First, although the HLS tool and Implementation flow attempt to create a design and layout that meets the timing requirements, they are not always successful. This is particularly important when considering integration with other system components such as interconnects. Secondly, the target HLS clock period directs the tool to create a circuit that would likely meet timing, typically at the cost of latency and flip flops. However, we observe that a large amount of slack is available for many points in the design space. Clock period targets can thus be used to parameterize the circuit design, but optimization over the placed and routed clock period estimate is necessary to obtain the best final design.

The design space of the most complex benchmark, AES, is visualized in Fig. 1. Final post-Implementation values are plotted where LUT usage is shown on the y-axis while the product of worst-case latency and clock period is on the x-axis; latency values are collected from the HLS estimates. Highlighted points show the dominating designs predicted using post-Implementation, post-Synthesis and post-HLS estimates. The post-HLS and post-Synthesis values are very informative, predicting designs close to the true Pareto-optimal frontier. However, the post-Implementation values provide the noticeably better designs. In this work, we exploit the fast

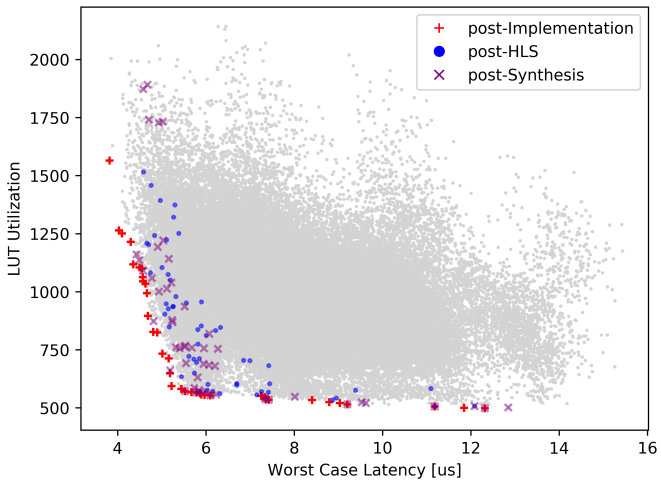

Fig. 1: Design space and estimated Pareto points for the AES benchmark. Each gray point is an implemented design.

estimates from HLS and Synthesis portions of the tool flow to rapidly guide optimizers toward optimal designs.

IV. Multi-Fidelity Modelling

In selecting an optimal combination of HLS directives, we wish to find a setting of directives x in the space of all settings $\mathcal{X}$ that optimizes an objective function $f : \mathcal{X} \to \mathbb{R}$, possibly under a constraint $c : \mathcal{X} \to \mathbb{R}$. For instance, $f(\cdot)$ may be a combination of LUT, BRAM or DSP usage for a particular directive setting. These functions are difficult to model since they involve the interaction of directives and evaluating the functions involve time-consuming executions of the CAD flow. Rather than sample the objective function directly to determine a search direction, a more efficient approach is to construct global models $\hat{f}$ and $\hat{c}$ to guide optimization.

In addition to modelling the final output after the full CAD flow, we desire a model that is able to incorporate estimates obtained early in the CAD flow to direct optimization decisions. To that end, we base our model on one that Kennedy and O'Hagan [12] developed for multi-fidelity computer simulations. In this section, we will describe the multi-fidelity model adapted for the HLS optimization context.

A. Gaussian Process Regression

Gaussian process (GP) regression is a flexible method of modelling many different kinds of functions and underlies the multi-fidelity model. In a GP, points in a function's domain are modelled as variables in a large, possibly infinite, multivariate Gaussian distribution. The predicted value from a GP is the conditional distribution given some sampled points and thus provides a probabilistic estimator. Some features of GPs used in this work will be reviewed in this section; more details can found in work by Rasmussen and Williams [19].

Although a GP is very flexible, it is parameterized by mean function $m(\cdot)$ and covariance function $k(\cdot, \cdot)$. The mean function provides a prior estimate of the value of the target function at different points in the design space; in this work, it is set to be a constant parameter value $m(x) = \beta \ \forall x \in \mathcal{X}$. Meanwhile, the covariance function governs how sampled points influence the prediction of unsampled points. The common and practical covariance function for D-dimensional vectors used in this work is the squared exponential function:

$$k(x, x') = \sigma^2 \exp(-\frac{1}{2}(x - x')^T \Lambda (x - x')), \quad (1)$$

where Λ is the diagonal matrix $\text{diag}(\lambda_1^{-2}, \lambda_2^{-2}, \ldots, \lambda_D^{-2})$. Each entry λ_i is the *length scale* for the associated dimension describing how much distance is required in that dimension before points are uncorrelated. The factor σ is used to scale the overall variance of the model. Given values Y sampled at locations $X \subseteq \mathcal{X}$, the predictive distribution at x is Gaussian and written:

$$\mathcal{N}(\beta + \Sigma_*^T \Sigma^{-1}(Y - \beta), \Sigma_{**} - \Sigma_*^T \Sigma^{-1} \Sigma_*) \quad (2)$$

where Σ_* is the vector of covariances between x and the points in X, Σ_{**} is the variance $k(x, x)$ and Σ is the covariance matrix between the sampled points in X.

To make use of a Gaussian process model with a standard squared exponential kernel, the HLS design space is projected onto a vector space. We use the method described by Lo et al. [3] to create a D-dimensional representation of the hierarchical directive settings.

B. Multi-Fidelity Model

Kennedy and O'Hagan [12] demonstrated how a Gaussian process can be extended to model the response of multiple fidelities of estimates. In this section, we will review the version of the model that we adopt for modelling the estimates in the HLS CAD flow.

Let s^m represent the number of fidelities of values available for a certain metric m. Fidelities are ordered from 1 to s^m such that the highest, most accurate, fidelity is s^m. For example, there are three fidelities of estimates available for the LUT usage metric, the highest obtained after Implementation. The value of the metric at fidelity i and directive setting x is written: $z_i^m(x)$. If at fidelity i, a set of directive settings $X_i \subseteq \mathcal{X}$ have been sampled, a corresponding set of output values $Y_i^m = \{z_i^m(x) | x \in X_i\}$ would be available. In the HLS CAD flow $X_i \subseteq X_{i-1}$ since each stage of the flow must occur in order. Thus, if the post-Implementation value is available for some directive setting, the post-Synthesis and post-HLS values will also be available. The predictive model integrates the observations from all fidelities and uses them to estimate the value of the metric at $z_{s^m}^m(x)$.

To integrate multiple fidelities of estimates together, the value of a function at a higher fidelity is modelled as a scaled version of a lower-fidelity estimate with an additive correction term. More formally, the model is written:

$$\hat{z}_1^m(x) = \delta_1(x | \beta_1, k_1(\cdot, \cdot)) \quad (3)$$

$$\hat{z}_i^m(x) = \underbrace{\rho_{i-1} \hat{z}_{i-1}^m(x)}_{\text{Lower Fidelity}} + \underbrace{\delta_i(x | \beta_i, k_i(\cdot, \cdot))}_{\text{Correction}}, i \in [2, s^m] \quad (4)$$

where $\hat{z}_i^m(x)$ is the model estimate at fidelity i. The lowest fidelity $\hat{z}_1^m(x)$ is modelled using a single Gaussian process δ_1 parameterized by its mean β_1 and covariance function $k_1(\cdot, \cdot)$. Higher fidelity estimates scale the lower fidelity output by a

factor ρ_{i-1} and add an independent Gaussian process δ_i to model the remaining differences.

With this structure, each $\hat{z}_i^m(x)$ is a set of $|\mathcal{X}|$ random variables, where $|\mathcal{X}|$ is the size of the design space. Together, the s^m fidelities form a joint distribution of $s^m|\mathcal{X}|$ variables representing the outputs at all of the fidelities. Predictions are made very similarly to a normal Gaussian process, where the conditional distribution is computed given the sampled points $(X_i, Y_i^m)\forall i \in [1, s^m]$. However, the covariance must be computed properly to relate outputs obtained at different fidelities. To simplify notation, let $\prod_a^b(\cdot) = 1$ if $a > b$. If x_1 and x_2 are sampled at fidelities h_1 and h_2 respectively then the following covariance function is used to compute elements in Σ_*, Σ_{**} and Σ in Eqn. 2.

$$\text{cov}(x_1, x_2) = \sum_{i=1}^{\min(h_1,h_2)} \Big((\prod_{n=i}^{h_1-1} \rho_n)(\prod_{n=i}^{h_2-1} \rho_n) k_i(x_1, x_2) \Big). \tag{5}$$

The mean value at fidelity h is written:

$$\mu_h = \sum_{i=1}^{h} \Big((\prod_{n=i}^{h-1} \rho_n)\beta_i \Big) \tag{6}$$

such that the predicted mean for x at fidelity s^m becomes:

$$\mu_{s^m} + \Sigma_*^T \Sigma^{-1} [Y_1^m - \mu_1, Y_2^m - \mu_2 \ldots Y_{s^m}^m - \mu_{s^m}]^T \tag{7}$$

assuming that the sampled values are ordered from lowest fidelity to highest.

Parameter Estimation: The parameters for $k_i(\cdot,\cdot)$ as well as β_i and ρ_i must be estimated. We follow the technique proposed by Kennedy and O'Hagan where the parameters of each fidelity are optimized in multiple stages. Specifically, parameters for $k_1(\cdot,\cdot)$ and β_1 are selected to maximize the log marginal likelihood of the data sampled in X_1. For the remaining fidelities, ρ_{i-1} and the parameters of δ_i are jointly optimized to model $z_i^m(x) - \rho_{i-1} z_{i-1}^m(x)$ for $x \in X_i$.

C. Modelling Metrics in HLS

When creating HLS designs, there may be several metrics that a designer would want to include in their objective or constraint functions such as utilization, clock period and latency metrics. In this paper, we consider LUT usage, worst-case estimated cycle count and clock period as relevant design metrics. Specifically, for unconstrained optimization, we consider the product of all three measures as an area-delay product and in constrained optimization we consider LUTs as the objective function constrained by the product of cycle count and clock period as a measure of wall clock runtime. Estimated values of these metrics are provided individually at multiple fidelities in the CAD flow. However, latency estimates are provided only from the HLS tool.

To estimate a combination of metrics with different available fidelities, we first create separate models for each individual metric before integrating them together. We use the log-transformed value of each metric rather than its direct value to provide numerical stability and facilitate this integration. The estimator for each metric at a directive setting x is then considered to be an independent log-normal distribution at the highest available fidelity. The estimator for a product of independent log-normal metrics is also log-normal and can be computed easily. For instance the log area-delay product would have the following mean and variance:

$$\mu_{\text{Log AD}}(x) = \mu_{\text{Log Latency}}(x) + \mu_{\text{Log Period}}(x) + \mu_{\text{Log LUT}}(x) \tag{8}$$

$$\sigma^2_{\text{Log AD}}(x) = \sigma^2_{\text{Log Latency}}(x) + \sigma^2_{\text{Log Period}}(x) + \sigma^2_{\text{Log LUT}}(x). \tag{9}$$

By modelling each metric separately the individual models are simpler and we are able to easily integrate metrics with different fidelities of estimates together.

V. Model-Based Optimization

The multi-fidelity models provide predictive distributions of values at different points in the design space. The optimization task is to use these models to efficiently explore the design space to minimize the objective function $f(\cdot)$ under a constraint $c(\cdot) \leq \gamma$. In the following, $f(\cdot)$, $c(\cdot)$ and γ are log-transformed values.

To perform optimization, a sequential model-based method is used. First, an initial random sample from the design space is chosen to seed the models. During each iterative optimization step, the current sampled points are used to predict values of the metrics across the design space. Next, the *incumbent* or best sampled design point is estimated. An *expected improvement* measure over the incumbent is computed for each point in the design space that is, in turn, used to either terminate the optimization or select the next directive setting and fidelity to sample. Finally, the chosen setting is evaluated up to the target fidelity, new values are added to the sampled set and the GP model parameters are updated if necessary[1]. In this work, we assume that the CAD flow is only evaluated when results are not available. That is, if HLS was already performed for some design point, it would not be executed again to obtain the post-Synthesis value. An outline for this algorithm is shown in Alg. 1. The following sections will describe incumbent estimation, directive and fidelity selection as well as termination in more detail.

A. Incumbent Estimation

In single-fidelity optimization with noiseless outputs, the incumbent value f^* is generally taken to be the minimum $\exp(f(x))$ that has been sampled so far during optimization. However, in the multi-fidelity case, we may wish to use a point that has been sampled at a lower fidelity but not yet at the final fidelity if we are confident in its value. Thus, following Huang et al. [13], we choose a design point:

$$x^* = \arg\min_{x \in X_1} E[\exp(\hat{f}(x))] + SD[\exp(\hat{f}(x))] \tag{10}$$

where $E[\cdot]$ and $SD[\cdot]$ are the expected value and standard deviation of the log-normal estimate. For constrained problems, the incumbent must also satisfy:

$$E[\exp(\hat{c}(x^*))] + SD[\exp(\hat{c}(x^*))] \leq \exp(\gamma) \tag{11}$$

The incumbent value f^* is then taken to be $E[\exp(\hat{f}(x^*))]$.

[1] Parameters for a metric were updated if the error between the prediction and sampled value was over 10% or after 9 iterations without an update.

Let s^m be max. fidelity for metric m and s be $\max(s^m)\forall m$
Let $z_i^m(\cdot)$ be the log values of metric m at fidelity i
Select initial points $X_s \subseteq X_{s-1} \cdots \subseteq X_1 \subset \mathcal{X}$
For each m, $Y_i^m \leftarrow \{z_i^m(x)|x \in X_i\}\forall i \in [1, s^m]$
while *not done* **do**
  foreach *metric* m **do**
    Compute $\hat{z}_{s^m}^m(x)\ \forall x \in \mathcal{X}$ using $X_i, Y_i^m\ i \in [1, s^m]$
  end
  Estimate objective and constraint: $\hat{f}(x)$, $\hat{c}(x)$
  $x^* \leftarrow$ INCUMBENT($\hat{f}(x), \hat{c}(x)$)
  Compute Expected Improvement: EI
  if *TERMINATE(EI)* **then**
    done $\leftarrow$ True
  else
    $\bar{x}, h \leftarrow$ SELECT(EI)
    $X_i \leftarrow X_i \cup \{\bar{x}\}$ for $i \in [1, h]$
    For each metric m, evaluate unsampled values:
      $Y_i^m \leftarrow Y_i^m \cup \{z_i^m(\bar{x})\}, i \in [1, \min(s^m, h)]$
    Update GP model parameters
  end
end

Algorithm 1: Sequential Model-Based Minimization

B. Directive Selection

The method of selecting a new point to sample is a critical part of model-based optimization. Given a model of the design space, it is important to not only exploit where the objective function is predicted to have a low value but also explore areas where the model is uncertain. One powerful method is to select the next point that maximizes the improvement of the objective function compared to the current best point. Since the models provide a distribution of values at a particular directive setting x, we can compute the *expected improvement* [20] of sampling that point by taking the expected value: $E[max(f^* - \hat{f}(x), 0)]$ where f^* is the incumbent value. In this work, since log-normal values are estimated, we compute $E[max(f^* - e^{\hat{f}(x)}, 0)]$. Let μ_f, σ_f be the mean and standard deviation of $\hat{f}(x)$ respectively, the log-transformed expected improvement was shown [21] to have the form:

$$LEI(x) = f^*\Phi(\frac{\ln(f^*) - \mu_f}{\sigma_f}) + e^{\frac{\sigma_f^2}{2}+\mu_f}\Phi(\frac{\ln(f^*) - \mu_f}{\sigma_f} - \sigma_f). \quad (12)$$

where Φ and ϕ are the cumulative distribution function and probability density function of the standard normal respectively.

Expected Improvement is used for global optimization where we wish to find the minimum function value in the entire domain $\mathcal{X}$. However, an important use case for HLS designs is constrained optimization where we might wish to minimize a design metric given some fixed requirements; for instance, minimizing LUT utilization given a latency or clock period constraint. Previous work [22] has proposed an extension to the expected improvement metric to incorporate inequality constraints. To perform the optimization: $\min_{c(x)\leq\gamma} f(x)$, we maximize $PF(x) \times LEI(x)$ instead of $LEI(x)$, where $PF(x) = \Phi(\frac{\gamma - \mu_c(x)}{\sigma_c(x)})$ and $\mu_c(x), \sigma_c(x)$ are the mean and standard deviation of $\hat{c}(x)$. Here, $PF(x)$ is the probability that the constraint will be satisfied at x.

C. Fidelity Selection

The heuristic of expected improvement is effective for selecting which point in the design space to select. However, it does not define directly how different fidelities can be used. We consider three methods that make use of the multi-fidelity model to select the next point and fidelity to explore: Dynamic Fidelity Selection, Low-Fidelity Initialization and a variation of the latter with early stopping.

1) Dynamic Selection: Huang et al. [13] proposed a method of performing optimization by dynamically selecting not only the next design point to sample but also the fidelity at which to sample. With this method, the fidelity becomes an extra dimension in the search space and the expected improvement metric is modified to consider the benefits and costs of sampling at different fidelities. In this paper, we consider noiseless estimates from each fidelity resulting in the following augmented expected improvement:

$$AEI(x, i) = LEI(x) \cdot \text{corr}[\hat{f}_i(x), \hat{f}(x)] \cdot C_s/C_i \quad (13)$$

where s is the highest fidelity. For this method, $\hat{f}_i(x)$, the predictive distribution at fidelity i is required in addition to the estimate at the highest fidelity $\hat{f}(x)$. In this equation, corr is the correlation between the estimated values, C_i is the log runtime cost of executing the CAD flow for fidelity i and thus C_s is the log runtime cost of obtaining the final values. Essentially the expected improvement for a fidelity i is penalized by its lack of correlation with the final result but increased by how much faster it is relative to the highest fidelity. Since the time to compute the output at a certain fidelity is cumulative, the cost considered is as well. For instance, the cost of obtaining an estimate after Implementation is computed as the total runtime through HLS, Synthesis and Implementation.

To adapt the method for constrained search, we apply the correlation factor to the probability of meeting the constraint to obtain an augmented probability function $APF(x, i) = PF(x) \cdot \text{corr}[\hat{c}_i(x), \hat{c}(x)]$, and maximize $AEI(x, i) \times APF(x, i)$.

2) Low-Fidelity Initialization: One way to make use of the lower fidelity estimates is to quickly establish the shape of the design space with cheaper estimates before homing in on the optimal value using the highest fidelity. To accomplish this, points from the lower fidelities are randomly sampled to create a model before optimization is performed, sampling at the highest fidelity.

Forrester et al. [14] used this method for optimization with a space-filling technique for selecting the points at which to sample. In this paper, the design space is not a hypercube. We consider the use of uniform random sampling from the design space rather than directed space-filling, although other methods of initialization such as Transductive Experimental Design [23] may be explored in future work. In addition, since HLS estimates are informative and require much less runtime to obtain, we only evaluate the initial points at the HLS fidelity.

3) Early Stopping: The final technique we consider is a modification of the initialization method. As we noted earlier, the CAD flow requires outputs of each stage to proceed. Thus, when sampling post-Implementation values, the post-HLS and post-Synthesis values are also generated. We can take advantage of these values to stop sampling a design point early if the estimates reveal that the final output will

not improve the current minimum. This would save overall runtime since the lengthy Implementation or Synthesis steps could be avoided. In applying this technique, a balance must be made between aggressively stopping search and improving the predictive model since high-fidelity, post-Implementation values can improve the model and search direction even if they do not improve the current minimum. We stop evaluation of a design point $\bar{x}$ if at any point in the design flow:

$$\frac{PF(\bar{x}) \times LEI(\bar{x})}{\max_{x \in \mathcal{X}}(PF(x) \times LEI(x))} \leq \theta_e, \tag{14}$$

where $PF(\cdot) = 1$ for unconstrained problems. After each stage of the CAD flow, the expected improvement is re-evaluated for all of the design points. If the expected improvement of the current directive setting is θ_e times worse than the new maximum expected improvement, then evaluation is stopped and the new design that maximizes expected improvement is evaluated. For the experiments in this paper, we chose θ_e to be 10^{-8} empirically; in addition, we prevent early stopping if no feasible point has been found.

4) Termination: In this paper, optimizers were terminated if no new minimum was found and $\max(EI(x))/f^*$ was less than 0.001 for D consecutive iterations, where D is the dimensionality of the design space. Early stopping methods were terminated if the termination criteria were met for $3 \times D$ iterations since they are checked more often. Optimizers were also terminated if no feasible points were found for 300 iterations. These criteria allowed most optimization experiments to reach the global minimum within reasonable runtime.

VI. Results

The optimization methods were evaluated using the datasets described in Section III for unconstrained LUT $\times$ delay minimization as well as LUT minimization constrained by delay. The delay consists of the product: Latency $\times$ Clock Period and we chose constraints that would make only 30% of the design space feasible for each benchmark. The global minimum for each optimization problem was found using the post-Implementation values from each benchmark's dataset. We record the percent difference between the optimizer's chosen minimum point evaluated through Implementation and the global minimum at each point in time. Each experiment was repeated 50 times and we consider the median best value found by the optimizers at each point in time.

A. Single-Fidelity Optimizers

Optimization using only post-Implementation values was performed as a *single-fidelity* baseline method. However, each metric was still modelled by independent GPs as described in Section IV-C. In addition, the intra-program *transfer* method described by Lo et al. [3] was implemented. This technique modifies the GP model to more closely relate multiple uses of the same HLS directive. The relation provides an improved estimate of how a directive affects the design metrics when only a few settings have been sampled. We used a modified version of transfer where dimension-specific scaling parameters are included since we found it improved performance.

B. Initialization

Single-fidelity optimizers were initialized with one random post-Implementation design sample. The multi-fidelity methods were initialized with 25 randomly selected design points, which worked well across the benchmarks. For low-fidelity initialization and the early stopping variant, 25 post-HLS samples were used. To establish the relation between fidelities for the dynamic selection method, one of the 25 points was evaluated through the Implementation stage. [2]

C. Discussion

Runtime results are tabulated in Table III where the values indicate the time in seconds required for a method to reach a specified distance from the global minimum; "inf" entries indicate that the distance was not reached before termination.

The first trend visible in these results is that the multi-fidelity methods provide significant reductions in optimization time relative to the single-fidelity methods on most tasks. A notable exception is unconstrained Radix Sort optimization, where transfer is able to direct the optimizer to a good value, close to the minimum, early in the search. Although in this case the multi-fidelity methods still perform competitively and reach the global minimum faster. Similar to providing design-space information using intra-program transfer, the lower fidelity estimates give general guidance to the surrogate model as to the shape of the objective and constraint functions. However, since the low-fidelity estimates are directly sampled from the design space, they characterize the specific objective and constraint functions being explored rather than the general designer knowledge applied in the transfer method.

The results also demonstrate that early stopping improves the runtime of the random initialization method. Taking advantage of the incremental stages of the HLS CAD flow enables the optimizer to avoid expensive Synthesis and Implementation tool executions. In the majority of tasks, the simple early stopping heuristic is able to determine when it is beneficial to continue or stop evaluation of a directive setting leading to significant reductions in runtime. In the case of unconstrained optimization on the Backpropagation benchmark, the HLS and Synthesis estimates become uninformative near the optimal value. Thus, exploiting the low-fidelity values through early stopping is not as beneficial in this problem and sampling post-Implementation values alone is able to reach the global minimum faster.

Finally, the dynamic selection method performs competitively with the low-fidelity initialization methods. As with early stopping, a heuristic is used to determine the fidelity to sample. However, the dynamic selection tends to more aggressively use lower fidelities at the beginning of search. This approach works well in some cases but can be detrimental when lower fidelities are not as informative for minimizing the objective function.

[2] We also experimented with initializing all methods identically with an extra post-Implementation sample for the low-fidelity initialization methods. This did not significantly affect the relative performance of the methods.

TABLE III: Time in seconds required for median minimum value to reach 10, 5, 1 and 0 percent error with the evaluated optimization methods. Minimum times are in bold.

Unconstrained Optimization					
	Method	10%	5%	1%	0%
AES	Single-Fidelity	16569	21182	39488	39814
	Single-Fidelity with Transfer	6331	13370	30297	31352
	Dynamic Selection	4733	10876	25988	26880
	Low-Fidelity Init.	3492	8096	18826	19868
	Earlystopping	**2108**	**5339**	**11463**	**11463**
Backpropagation	Single-Fidelity	4826	4826	60336	68788
	Single-Fidelity with Transfer	6032	6032	54302	57917
	Dynamic Selection	2883	6559	29291	76556
	Low-Fidelity Init.	2881	4088	**25828**	**45149**
	Earlystopping	**1675**	**2902**	30721	60209
Radix Sort	Single-Fidelity	1725	1725	7933	8279
	Single-Fidelity with Transfer	1380	**1380**	**1380**	6211
	Dynamic Selection	2947	3036	3185	**3245**
	Low-Fidelity Init.	1614	1959	3338	5066
	Earlystopping	**1278**	1617	2608	3542
ADPCM	Single-Fidelity	33356	46545	46548	48297
	Single-Fidelity with Transfer	5266	21943	22820	24575
	Dynamic Selection	**4746**	17960	18842	20228
	Low-Fidelity Init.	9719	14998	16763	21149
	Earlystopping	6216	**9652**	**10294**	**13605**

Constrained 30% Optimization					
	Method	10%	5%	1%	0%
AES	Single-Fidelity	11621	12334	20101	inf
	Single-Fidelity with Transfer	3869	4925	9849	62613
	Dynamic Selection	2366	3548	8823	45283
	Low-Fidelity Init.	1738	2788	7030	33329
	Earlystopping	**1397**	**2098**	**4923**	**24619**
Backpropagation	Single-Fidelity	4826	4826	26546	67591
	Single-Fidelity with Transfer	6033	6033	15692	39821
	Dynamic Selection	2864	2864	2883	**20486**
	Low-Fidelity Init.	2862	2862	2881	40318
	Earlystopping	**1675**	**1675**	**1675**	20991
Radix Sort	Single-Fidelity	9664	10007	11038	14145
	Single-Fidelity with Transfer	5520	5864	7935	15523
	Dynamic Selection	**2629**	**2688**	**2784**	**4162**
	Low-Fidelity Init.	4369	4377	5405	11656
	Earlystopping	2685	2813	3010	6644
ADPCM	Single-Fidelity	24576	27208	37746	49185
	Single-Fidelity with Transfer	9653	11409	18429	21940
	Dynamic Selection	**2817**	**3050**	**5856**	**9830**
	Low-Fidelity Init.	8839	11484	14117	15875
	Earlystopping	4506	5890	8021	10044

VII. Conclusions and Future Work

In this paper, we have considered the use of estimates available in the FPGA HLS CAD flow for automated optimization. We found varying accuracy of estimates, but good correlations of area values across fidelities. When performing optimization, it is clear that the fast estimates provided by the HLS and Synthesis steps are informative and multi-fidelity methods that adaptively sample the different fidelities are able to dramatically speed up optimization time over previously examined approaches. However, additional studies are necessary to determine the most effective methods for sampling the different estimates available in the CAD flow. In future work, we will develop and evaluate more advanced heuristics for fidelity selection.

Acknowledgements

The authors would like to thank NSERC and Xilinx for the funding and tools used in this project as well as the anonymous reviewers for their helpful comments.

References

[1] F. Winterstein *et al.*, "High-Level Synthesis of Dynamic Data Structures: A Case Study Using Vivado HLS," in *Proc. Int. Conf. on Field-Programmable Technology (FPT)*, Dec. 2013, pp. 362–365.

[2] J. Monson *et al.*, "Optimization Techniques for a High Level Synthesis Implementation of the Sobel Filter," in *Proc. Int. Conf. on Reconfigurable Computing and FPGAs (ReConFig)*, Dec. 2013, pp. 350–355.

[3] C. Lo *et al.*, "Model-Based Optimization of High Level Synthesis directives," in *Proc. Int. Conf. on Field-Programmable Logic and Applications (FPL)*, Aug. 2016, pp. 389–398.

[4] A. Mahapatra *et al.*, "Machine-Learning based Simulated Annealer method for High Level Synthesis Design Space Exploration," in *Proc. Electronic System Level Synthesis Conference (ESLsyn)*, May 2014, pp. 25–30.

[5] H.-Y. Liu *et al.*, "On Learning-Based Methods for Design-Space Exploration with High-Level Synthesis," in *Proc. Design Automation Conference (DAC)*, May 2013, pp. 348–354.

[6] D. Liu *et al.*, "Efficient and Reliable High-Level Synthesis Design Space Explorer for FPGAs," in *Proc. Int. Conf. on Field-Programmable Logic and Applications (FPL)*, Aug. 2016, pp. 460–467.

[7] P. Meng *et al.*, "Adaptive Threshold Non-Pareto Elimination: Re-thinking Machine Learning for System Level Design Space Exploration on FPGAs," in *Proc. Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2016, pp. 918–923.

[8] M. Zuluaga *et al.*, "e-PAL: An Active Learning Approach to the Multi-Objective Optimization Problem," *Journal of Machine Learning Research*, vol. 17, no. 104, pp. 1–32, 2016.

[9] M. K. Papamichael *et al.*, "Nautilus: Fast Automated IP Design Space Search Using Guided Genetic Algorithms," in *Proc. Design Automation Conference (DAC)*, Jun. 2015, pp. 255–260.

[10] J. W. Bandler *et al.*, "Space Mapping Technique for Electromagnetic Optimization," *IEEE Transactions on Microwave Theory and Techniques*, vol. 42, no. 12, pp. 2536–2544, 1994.

[11] T. Todman *et al.*, "Reconfigurable Design Automation by High-Level Exploration," in *Proc. Int. Conf. on Application-Specific Systems, Architectures and Processors (ASAP)*, Jul. 2012, pp. 185–188.

[12] M. C. Kennedy *et al.*, "Predicting the Output from a Complex Computer Code When Fast Approximations Are Available," *Biometrika*, vol. 87, no. 1, pp. 1–13, 2000.

[13] D. Huang *et al.*, "Sequential kriging optimization using multiple-fidelity evaluations," *Structural and Multidisciplinary Optimization*, vol. 32, no. 5, pp. 369–382, 2006.

[14] A. I. J. Forrester *et al.*, "Multi-Fidelity Optimization via Surrogate Modelling," *Proceedings: Mathematical, Physical and Engineering Sciences*, vol. 463, no. 2088, pp. 3251–3269, 2007.

[15] L. Le Gratiet *et al.*, "Recursive Co-Kriging Model for Design of Computer Experiments With Multiple Levels of Fidelity," *International Journal for Uncertainty Quantification*, vol. 4, no. 5, pp. 365–386, 2014.

[16] B. Reagen *et al.*, "MachSuite: Benchmarks for Accelerator Design and Customized Architectures," in *Int. Symp. on Workload Characterization (IISWC)*, Oct. 2014, pp. 110–119.

[17] Y. Hara *et al.*, "Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis," *Journal of Information Processing*, vol. 17, pp. 242–254, 2009.

[18] C. Loken *et al.*, "SciNet: Lessons Learned from Building a Power-efficient Top-20 System and Data Centre," *Journal of Physics: Conference Series*, vol. 256, p. 012026, 2010.

[19] C. E. Rasmussen *et al.*, *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

[20] D. R. Jones *et al.*, "Efficient Global Optimization of Expensive Black-Box Functions," *Journal of Global Optimization*, vol. 13, no. 4, pp. 455–492, Dec. 1998.

[21] F. Hutter *et al.*, "An Experimental Investigation of Model-based Parameter Optimisation: SPO and Beyond," in *Proc. Conf. on Genetic and Evolutionary Computation (GECCO)*, Jul. 2009, pp. 271–278.

[22] J. Gardner *et al.*, "Bayesian Optimization with Inequality Constraints," in *Proc. Int. Conf. on Machine Learning (ICML)*, Jun. 2014, pp. 937–945.

[23] K. Yu *et al.*, "Active Learning via Transductive Experimental Design," in *Proc. Int. Conf. on Machine Learning (ICML)*, Jun. 2006, pp. 1081–1088.