Correlated Multi-objective Multi-fidelity Optimization for HLS Directives Design

Qi Sun[1], Tinghuan Chen[1], Siting Liu[1], Jin Miao[2], Jianli Chen[3], Hao Yu[4], Bei Yu[1]
[1]The Chinese University of Hong Kong [2]Synopsys [3]Fudan University [4]SusTech
{qsun,byu}@cse.cuhk.edu.hk

***Abstract*—High-level synthesis (HLS) tools have gained great attention in recent years because it emancipates engineers from the complicated and heavy hardware description language writing, by using high-level languages and HLS directives. However, previous works seem powerless, due to the time-consuming design processes, the contradictions among design objectives, and the accuracy difference between the three stages (fidelities).**

To find good HLS directives, in this paper, a novel correlated multi-objective non-linear optimization algorithm is proposed to explore the Pareto solutions while making full use of data from different fidelities. A non-linear Gaussian process is proposed to model relationships among the analysis reports from different fidelities for the same objective. For the first time, correlated multivariate Gaussian process models are introduced into this domain to characterize the complex relationships of multiple objectives in each design fidelity. A tree-based method is proposed to erase invalid solutions and obviously non-optimal solutions. Experimental results show that our non-linear and pioneering correlated models can approximate the Pareto-frontier of the directive design space in a shorter time with much better performance and good stability, compared with the state-of-the-art.

I. Introduction

High-level synthesis (HLS) tools have made it possible for users who are not experts in writing hardware description languages (HDLs) to implement their FPGA designs, by translating high-level programming languages (*e.g.*, C/C++) to low-level HDLs (*e.g.*, Verilog HDL and VHDL). Given different HLS directive configurations, the final hardware architectures generated from the same high-level language description may vary a lot from each other in terms of power, delay, and resource consumptions. Therefore, with the help of HLS directives, we can optimize the designs without changing the high-level program source codes, which greatly reduces the time and labor costs of rapid prototyping implementations. Fig. 1 shows an example of HLS directives. With these advantages, HLS tools have been widely used in many applications [1].

However, the complexities of HLS directives still hinder its further spreads. Composed of three stages, *i.e.*, high-level synthesis (HLS), logic synthesis (Synth), and physical implementation (Impl), the whole design flow is time-consuming. Later stages can report more accurate results, at the cost of longer running times. The reported results of these three stages are usually in non-linear relationships which make it difficult to map between them. That is why we cannot guarantee whether a design is valid in the Impl stage, even though its HLS performance is good. It is also difficult to find a good balance among various design objectives (Pareto configurations), which are mutually contradicted. The whole design flow is shown in Fig. 2. This kind of multi-stage design is also called multi-fidelity design. Users may prefer only conducting the first stage to promote fast developments, while sacrificing accuracy and bringing high risks of losing optimal solutions.

To tackle these challenges, great efforts have been made. Some analytical methods were proposed to estimate the performance with no need of running the real design flow. All possible directive configurations are analyzed to find good directives for general applications by using an analytic model or a simulator [2], [3]. Some formulations were proposed to prune some directive configurations for deep neural network applications [4]. Some proposed well-designed heuristics methods, *e.g.*, [5].

```
comp(int in[10], int out[10]):
  #PRAGMA HLS INLINE={ON, OFF}
  for(i = 0; i < 10; i ++) {
    #PRAGMA HLS UNROLL factor={2,5,10}
    in[i] = out[i];
  }
```

Fig. 1 HLS pseudo-codes and directives. The directives are in red. Each directive has some factors, *e.g.*, 2, 5, and 10.

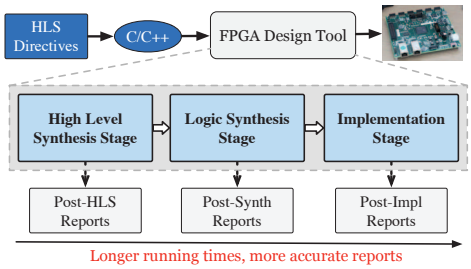


Fig. 2 The FPGA HLS design flow. C/C++ source code and HLS directives are fed into the design tool. There are three analysis stages and the later stages obtain more accurate reports but consume longer running times.

However, these works highly depend on the accuracy of the analytical models and lack generality. The highly non-linear and complicated mechanisms in real applications make these methods unable to handle complicated or new directives.

Some model-based works use machine learning algorithms to fit the system performance models. Compared to the analytical methods, model-based methods are more flexible and general. In these works, the authors collect lots of data to train machine learning models. [6]–[10] proposed several linear and non-linear regression models. Some typical models include Linear Regression, Artificial Neural Network, and Gradient Tree Boosting. However, in these algorithms, huge amounts of real design reports are necessary to guarantee the model accuracy. The characteristics of multi-fidelity are not utilized. To reduce the simulation costs and make full use of the existing reports, recently, Bayesian optimization (BO) approaches based on Gaussian process (GP) have been proposed. The multiple objectives are modeled as independent GP models [11] and then solved with Bayesian inference. This work is further extended by considering linear multi-fidelity designs [12].

However, it is regrettable that some important characteristics of the directive design are ignored. Firstly, the multiple design objectives are in complex correlated relationships. This brings great challenges to the previous methods, especially the heuristics methods. The correlation has been proven to be an important factor in practical scenarios [13] and it is indispensable to take it into consideration. Secondly, the performance values of the three design stages and the directives are in highly non-linear relationships. No directives can control the performance

978-3-9819263-5-4/DATE21/©2021 EDAA

explicitly, *e.g.*, the clock period or power. Some evaded these by only considering the lowest fidelity, though they missed some data, *e.g.*, the clock period after Synth and Impl. Unfortunately, to the best of our knowledge, most of the previous methods did not focus on these two characteristics, no matter the analytical methods, or the model-based methods.

In this paper, to help solve these problems, we proposed a novel correlated multi-objective multi-fidelity optimization method. Our contributions are summarized as follow:

- Non-linear multi-fidelity models are built to measure the non-linear relationship between the three stages, and balance data utilization, model accuracy, and model training costs.
- Correlated multi-objective GP models are proposed to tackle with both of the correlated relationships among various design objectives and the implicit and complicated mapping relationship between directives and objective values, to find Pareto configurations accurately.
- An effective tree-based method is proposed to prune the configuration design space, by removing incompatible and invalid designs.
- Three design objectives, power, performance, and area (PPA) are considered in this paper, thus making the task more practical and very challenging. The proposed techniques are integrated by constructing a powerful GP-based Bayesian optimization algorithm. The experimental results show the outstanding performance of our algorithm.

The rest of our paper is organized as follows. Section II provides preliminaries including models and definitions. Section III presents the tree-based pruning method and the whole optimization flow. Section IV introduces our non-linear multi-fidelity model and Pareto-driven correlated multi-objective model. Section V conducts several experiments to validate our methods, followed by conclusions in Section VI.

II. Preliminaries

An HLS directive configuration can be represented as a vector $\boldsymbol{x} \in \mathbb{R}^D$, in design space $\mathcal{X}$. The goal is to find one configuration from $\mathcal{X}$ that optimizes objective function f which can be deterministic or noisy, depending on the design specifications. In multi-objective optimization problem, for example, there are M objectives, $f^m : \mathcal{X} \to \mathbb{R}$, for $m \in \{1, \ldots, M\}$. Denote the objective value space as $\mathcal{Y}$, and each value point in it as $\boldsymbol{y} = [f^1(\boldsymbol{x}), f^2(\boldsymbol{x}), \ldots, f^M(\boldsymbol{x})]^\top$.

A. Gaussian Process Regression

Gaussian process (GP) regression is a flexible method to model the objective function, which is specified by the mean function $m(\boldsymbol{x})$ and covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$. $m(\boldsymbol{x})$ provides the prior estimations of the objective values for input $\boldsymbol{x}$, and typically a constant mean function $m(\boldsymbol{x}) = \mu_0$ is widely used. A common form of $k(\boldsymbol{x}, \boldsymbol{x}')$ is a squared exponential function of $\boldsymbol{x}$ [14].

Define a single-objective training set $\{\boldsymbol{X}, \boldsymbol{Y}\}$, where $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ is a set of directive configurations and $\boldsymbol{Y} = \{y_1, \ldots, y_n\}$ is the corresponding single-objective value set. Define the objective function as $f(\boldsymbol{x}) \sim N(\mu(\boldsymbol{x}), \Sigma(\boldsymbol{x}))$, which is assumed to be influenced by the independent and identical zero-mean Gaussian noise $\epsilon_e \sim \mathcal{N}(0, \sigma_e^2)$. Therefore, we can derive $y_i = f(\boldsymbol{x}_i) + \epsilon_e$, with $i = 1, \ldots, n$. For a newly sampled configuration $\boldsymbol{x}_*$ and its corresponding objective function f_*, the joint distribution between f_* and the data set $\boldsymbol{Y}$ sampled in previous steps is defined as follows:

$$p(\boldsymbol{Y}, f_*) = \mathcal{N}(\begin{bmatrix}\boldsymbol{\mu}_0 \\ \mu_0\end{bmatrix}, \begin{bmatrix}\boldsymbol{K}(\boldsymbol{X}) + \sigma_e^2 \boldsymbol{I} & \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_*) \\ \boldsymbol{k}^\top(\boldsymbol{X}, \boldsymbol{x}_*) & k(\boldsymbol{x}_*, \boldsymbol{x}_*)\end{bmatrix}), \tag{1}$$

where $\boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_*)$ is a vector of covariance between $\boldsymbol{x}_*$ and all of the configurations in $\boldsymbol{X}$, and $\boldsymbol{K}(\boldsymbol{X})$ is the intra-covariance matrix among configurations in $\boldsymbol{X}$. If a new point $\boldsymbol{x}_*$ is sampled from the design space, we will update $f(\boldsymbol{x})$ accordingly. Before the posterior is calculated, the hyper-parameter σ_e need to be determined by maximum likelihood estimation.

B. Bayesian Optimization

Bayesian optimization is a sequential strategy for optimization of black-box function which does not assume any explicit forms, with the help of GP models [14]. The whole design space is already defined, *i.e.*, all of the points are already known except their objective values. An initial set of points is sampled from the design space to initialize the GP model. Then it iteratively samples a new point from the design space and updates function f in each optimization step. An acquisition function, which balances the exploration and exploitation of data, is defined to determine which point will be sampled in the next optimization step. Expected improvement (EI) is a widely used acquisition function, which can be defined as Equation (2).

$$\begin{aligned} \mathrm{EI}(\boldsymbol{x}) &= \sigma(\boldsymbol{x})(\lambda\Phi(\lambda) + \phi(\lambda)), \\ \lambda &= \frac{\tau - \xi - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}, \end{aligned} \tag{2}$$

where τ is the current best objective value, and ξ is a jitter to improve the ability of exploration, $\sigma(\boldsymbol{x})$ is the uncertainty of GP model, $\mu(\boldsymbol{x})$ is the mean function, $\Phi(\cdot)$ and $\phi(\cdot)$ are the Cumulative Distribution Function and Probability Density Function, respectively.

C. Multi-objective Optimization and Pareto Optimality

The multiple objectives to be optimized would be conflicting with each other. One straightforward strategy is to define the objective value as a summation of all objectives with weights. More practically, to find one solution that strikes a balance among the multiple objectives, we want to identify the Pareto-optimal set.

Definition 1 (Pareto Optimality): In an M-dimension minimization problem, an objective vector $f(\boldsymbol{x})$ is said to dominate $f(\boldsymbol{x'})$ if

$$\begin{aligned} &\forall i \in [1, M], f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{x'}) \text{ and} \\ &\exists j \in [1, M], f_j(\boldsymbol{x}) < f_j(\boldsymbol{x'}). \end{aligned} \tag{3}$$

A point $\boldsymbol{x}$ is Pareto-optimal if there is no other $\boldsymbol{x'}$ in design space satisfying that $f(\boldsymbol{x'})$ dominates $f(\boldsymbol{x})$. In the whole design space, the set of points that are not dominated by others is called the Pareto-optimal set, denoted as $\mathcal{P}(\mathcal{Y}) \in \mathcal{Y}$. For Pareto-optimal points, there does not exist an alternative choice that can improve every objective without sacrificing others [15].

D. Multi-fidelity Optimization

Fidelity and Multi-fidelity Model are defined as follows:

Definition 2 (Fidelity): Fidelity refers to the degree to which a model reproduces the state of a real-world project or application. It is therefore a measure of the realism of the model. Low fidelity is a low-precision model and high fidelity has higher precision.

Definition 3 (Multi-fidelity Model): For a multi-fidelity problem, each fidelity i has a regression model $f_i(\boldsymbol{x})$. The multi-fidelity model can be defined as:

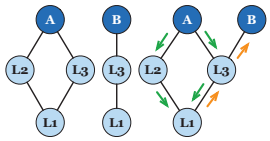
$$f_{i+1}(\boldsymbol{x}) = z(f_i(\boldsymbol{x}), \boldsymbol{x}), \tag{4}$$

where $z(\cdot)$ is a machine learning model or function, f_i is the model of the lower fidelity and f_{i+1} is the higher one.

The higher fidelities have higher accuracies at the cost of longer running times, compared to the lower fidelities. If the results at low fidelities are good enough, we will not run that test up to the high fidelities, to save time. Therefore, we need to measure the quality of the reports at each fidelity to determine whether we need to run the later design stages. In the rest of this paper, the three fidelities in FPGA design shown in Fig. 2 are shorted as hls, syn, and $impl$.

```
for L1 in range(0,N1):
  for L2 in range(0,N2):
    op(A[ L1 * 10 + L2 ])
  for L3 in range(0,N3):
    op(A[ L1 * 10 + L3 ])
    op(B[ L1 * 10 + L3 ])
```

(a)



(b)

Fig. 3 An example of tree-based pruning method. (a) Code with three loops and two arrays. (b) Trees of array A and B, and the merged tree.

III. HLS Directive Design Optimization

In this section, the tree-based pruning method is described in detail. We propose a whole design optimization flow, together with the directive encoding method and objective selection.

A. Tree-based Design Space Pruning

Some HLS directives are widely used, including pipelining (with or without initialization interval specification), loop unrolling, and array partitioning, *etc*. In general, the C/C++ source codes are composed of several for-loops and some arrays. Traditionally, the design space is simply generated by permutations and combinations of directives. However, some directives are conflicting and some are obviously non-optimal, especially for loop unrolling and array partitioning. Infeasible configurations may mislead our model and increase running times. For example, for an array used in a for-loop, if the array partitioning factor is less than the loop unrolling factor, this loop may not be unrolled successfully because the visits to the array are limited by the partitioning factor. If the array partitioning factor is greater, more memory resources are consumed without increasing the system parallelism. Under this circumstance, the compatible unrolling and partitioning are the best. A tree-based design space pruning method is proposed to help solve this problem, as shown in Algorithm 1.

Algorithm 1 The Pseudo-code of Tree-based Pruning Method

1: **Inputs**: High-level programming language source code, directive description files;
2: **Outputs**: Pruned design space $\mathcal{X}$, initially $\mathcal{X} \leftarrow \emptyset$;
3: Construct a tree for each array, with itself as the root node and related loops as children nodes;
4: Merge trees with common nodes, denote the set of trees as $\mathcal{T}$;
5: **for all** tree $t_i \in \mathcal{T}$ **do**
6: **for** root (array) node a_j in t_i **do**
7: **for** partitioning factor f_k of a_j **do**
8: Assign f_k to a_j;
9: Assign a unrolling factor to each loop node in t_i;
10: Backtrack from leaf nodes, assign partitioning factors to array nodes in t_i, except a_j;
11: **end for**
12: Record feasible configurations of a_j as set C_j; $\mathcal{X} \leftarrow \mathcal{X} \cup C_j$;
13: **end for**
14: **end for**
15: Traverse $\mathcal{X}$ and remove repeated configurations;
16: **return** Pruned design space $\mathcal{X}$.

Fig. 3 shows an example. If we partition A with type `CYCLIC`, then we will assign unrolling factors for $L2$ and $L3$. But we will not unroll $L1$, because $L1$ is incompatible with `CYCLIC` partitioning of A and unrolling of $L2$ and $L3$. After that we will backtrack from $L1$ to assign `CYCLIC` partitioning factors to B because A and B are in the same loop $L3$ and their partitioning types should be the same. In the same manner, with the above tree-based method, a large amount of invalid and incompatible directive configurations can be pruned.

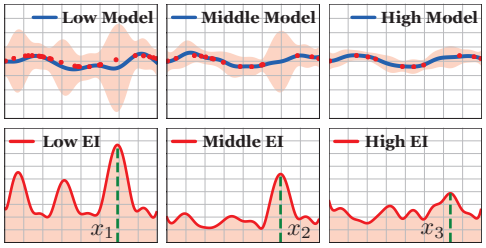


Fig. 4 A toy example to explain the 3 fidelities and EI functions. Lower fidelities have wider error ranges (light red fillers). The lowest fidelity model obtains the highest EI value at x_1. Therefore, in this optimization step, the lowest fidelity is selected and x_1 is added into CS.

B. Directive Encoding

Each directive configuration should be encoded to be an input feature vector of the mathematical models. The TRUE/FALSE factors are represented as 0 or 1 directly. The directives which have several factors are represented as normalized features, *e.g.*, three factors $\{2, 5, 10\}$ are encoded as $\{0, 0.375, 1\}$. This normalization method highlights the differences between these two factors while computing the distance between these two feature vectors and therefore it is better than the one-hot encoding. The final feature vector for a code segment is the concatenations and combinations of features of all the directives in it.

C. Objective Selection

Power, performance, area (PPA) are three popular metrics. To measure the system performance, we choose to use task time length (Delay), *i.e.*, the product of latency and clock period. Latency reflects how many clock cycles are needed to finish one task. The clock period is the time length of each cycle, which reflects the congestion information of the designs and is a key design indicator for some applications. We use the utilization of look-up table (LUT) as the area metric. LUT can be used to implement the control logic and simple computations. For tasks requiring high parallelism designs, the LUT utilization is usually the key metric. Power as a metric is directly used in this paper. Compared to works that consider only one or two metrics or linear combinations of these metrics, our work is more practical and challenging.

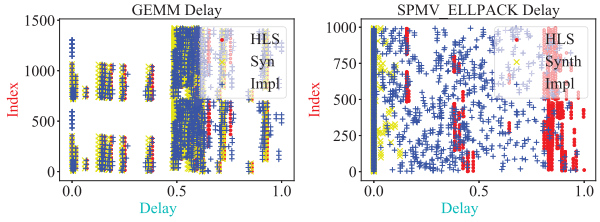
D. Overall Optimization Flow

Bayesian optimization method is adopted as the algorithm skeleton to sample the Pareto-optimal directive configurations. Sometimes we do not want to run the whole design flow to get reports from all fidelities because it is time-consuming. Multi-fidelity models are defined to make full use of the data from different fidelities. The low-fidelity reports can be systematically combined with the input features to predict the high-fidelity outputs more efficiently. For each fidelity, to find the Pareto point, an expected improvement (EI) function of the correlated multi-objective model is defined. Concrete forms of the GP models and EI functions are defined in Section IV.

The overall optimization flow is shown in Algorithm 2. Firstly, we define and prune the design space according to tree-based method as shown in Algorithm 1. Secondly, we randomly sample some points from the design space to initialize all the models. The points run in the higher fidelities are subsets of the lower fidelities, *i.e.*, $\mathcal{X}_{impl} \subseteq \mathcal{X}_{syn} \subseteq \mathcal{X}_{hls} \subseteq \mathcal{X}$. These points are then fed into the FPGA design tool to get real reports. For each fidelity, we initialize a multivariate correlated multi-objective GP model. The candidate Pareto set is CS. In each optimization time step, for each fidelity i, we will select a configuration $\hat{\boldsymbol{x}}_i$ which maximizes the expected improvement EI_i. $\hat{\boldsymbol{x}}_i$ is regarded as the candidate Pareto point in this fidelity. Then a node-fidelity pair $(\boldsymbol{x}^*, h)$ is selected from these three $\hat{\boldsymbol{x}}_i$ points which achieves the highest

Algorithm 2 The Optimization Flow Based on Bayesian Optimization Method

Inputs: High-level programming language source code, and optimization steps N_{iter};
Outputs: Pareto set CS, initially $CS \leftarrow \emptyset$;
Run tree-based pruning method to get pruned design space $\mathcal{X}$;
Randomly sample initial sets $\mathcal{X}_{impl} \subseteq \mathcal{X}_{syn} \subseteq \mathcal{X}_{hls} \subseteq \mathcal{X}$;
Initialize a GP model GP_i and an EI function EI_i for each fidelity;
for $t \leftarrow 1$ to N_{iter} **do**
  for all fidelity $i \in \{hls, syn, impl\}$ **do**
    Update GP_i and EI_i according to $\mathcal{X}_i$;
    $\hat{\boldsymbol{x}}_i \leftarrow \arg\max_{\boldsymbol{x} \in \mathcal{X}} EI_i(\boldsymbol{x})$;
  end for
  $(\boldsymbol{x}^*, h) \leftarrow \arg\max_{(\hat{\boldsymbol{x}}_i, i)} \mathrm{EI}_i(\boldsymbol{x})$, for $i \in \{hls, syn, impl\}$;
  Run FPGA tool with $\boldsymbol{x}^*$ up to fidelity h, to get $\boldsymbol{y}_h$;
  $\mathcal{X}_i \leftarrow \mathcal{X}_i \cup \boldsymbol{x}^*$, with i up to h;
  $CS \leftarrow CS \cup \{\boldsymbol{x}^*, \boldsymbol{y}_h\}$, $\mathcal{X} \leftarrow \mathcal{X} \setminus \boldsymbol{x}^*$;
end for
return Pareto set CS.



(a) (b)

Fig. 5 Normalized delay values in three fidelities. Y-axis is the configuration index. (a) GEMM. (b) SPMV_ELLPACK.

expected improvements. Here h denotes the fidelity of $\boldsymbol{x}^*$. $\boldsymbol{x}^*$ is our final choice of Pareto point in current optimization step and will be added into CS. A one-dimension toy example on the GP model and EI function is shown in Fig. 4. We run the FPGA design tool up to fidelity h to get real reports and then update all the corresponding GP models and EI functions. The final output set is CS.

IV. Proposed Models

In this section, non-linear multi-fidelity models and correlated multi-objective models which are used in Algorithm 2 are described in detail.

A. Non-linear Multi-Fidelity Model

Traditionally, in HLS directive designs, the relationship among the multiple fidelities is assumed to be linear, *e.g.*, [12]. However, it is not suitable for situations when these three FPGA design stages exhibit strong non-linear correlations. Therefore, a non-linear multi-fidelity model is proposed in this paper to further exploit the corresponding non-linear correlations between the low- and high-fidelity models. The non-linear model can be formulated as Equation (5).

$$f_{i+1}(\boldsymbol{x}) = z(f_i(\boldsymbol{x}), \boldsymbol{x}) + f_e(\boldsymbol{x}), \forall i \in \{1, \ldots, L-1\}, \tag{5}$$

where $z(\cdot)$ is the non-linear function and is modeled by a Gaussian process model in this paper, and $f_e(\boldsymbol{x})$ is the error term which is also defined as Gaussian process model. The outputs of the low fidelity are concatenated with the directive encoding features as the input features to the high fidelity GP model.

Delay values of two benchmarks are shown in Fig. 5 as examples to illustrate the complex non-linear relationships among the three fidelities. In GEMM, delay values of one configuration in three fidelities are highly overlapping. For SPMV_ELLPACK, delay values in the three fidelities show high divergences. The high divergences of various applications make it hard to regress the relationships with traditional linear models. Obviously, using non-linear models is a wise and general choice to handle various applications.

B. Correlated Multi-Objective Model

To learn and measure the Pareto set for the multi-objective optimization problem, we introduce the expected improvement of Pareto hyper-volume (EIPV) [16] and define it as the acquisition function, *i.e.*, the EI function in Algorithm 2. Firstly, we will clarify the concept of the improvement of Pareto hyper-volume. Secondly, we will define the probability model and compute the value of expected improvement.

Assume that in current optimization step $t+1$, we already have a Pareto-optimal HLS configuration set $\mathcal{D} = \{\boldsymbol{x}_s, \boldsymbol{y}_s\}_{s=1}^t$ and its value set $\mathcal{P}(\mathcal{Y}) = \{\boldsymbol{y}_s\}_{s=1}^t$. A virtual configuration point $\boldsymbol{v}_{ref} \in \mathbb{R}^M$ is defined as the reference point, which is dominated by $\mathcal{P}(\mathcal{Y})$, *i.e.*, $\boldsymbol{y}_s \succeq \boldsymbol{v}_{ref}$ for $\forall \boldsymbol{y}_s \in \mathcal{P}(\mathcal{Y})$. The values of $\boldsymbol{v}_{ref}$ are the extremely large values of the multiple design objectives. The Pareto hyper-volume with respect to $\boldsymbol{v}_{ref}$ is defined as Equation (6).

$$\mathrm{PV}_{\boldsymbol{v}_{ref}}(\mathcal{P}(\mathcal{Y})) = \int_{\mathbb{R}^M} \mathbb{I}[\boldsymbol{y} \succeq \boldsymbol{v}_{ref}] \left[1 - \prod_{\boldsymbol{u} \in \mathcal{P}(\mathcal{Y})} \mathbb{I}[\boldsymbol{u} \not\succeq \boldsymbol{y}]\right] \mathrm{d}\boldsymbol{y}, \tag{6}$$

where $\mathbb{I}(\cdot)$ is the indicator function, which outputs 1 if its argument is true and 0 otherwise. This equation measures the volume of the value space composed of configurations which dominate $\boldsymbol{v}_{ref}$ but are dominated by at least one configuration in $\mathcal{P}(\mathcal{Y})$. The greater the volume is, the better the Pareto set is.

Greedily, in each operation step, we want to sample a configuration which can lead to the highest improvement of the Pareto hyper-volume. We will traverse the un-sampled configuration space and estimate the expected improvement for each configuration. The expected improvement is defined as Equation (7).

$$\begin{aligned}&\mathrm{EIPV}(\boldsymbol{x}_{t+1}|\mathcal{D}) = \\ &\mathbb{E}_{p(\boldsymbol{y}(\boldsymbol{x}_{t+1})|\mathcal{D})}\left[\mathrm{PV}_{\boldsymbol{v}_{ref}}(\mathcal{P}(\mathcal{Y} \cup \boldsymbol{y}(\boldsymbol{x}_{t+1}))) - \mathrm{PV}_{\boldsymbol{v}_{ref}}(\mathcal{P}(\mathcal{Y}))\right].\end{aligned} \tag{7}$$

We can decompose the whole value space into grid cells to simplify the integration of Equation (6), as shown in Fig. 6(a). The decomposition is according to the locations of Pareto-optimal configurations. The corresponding objective values at these two axes are b_i^1 and b_i^2. We denote the non-dominated cells as $\mathcal{C}_{\mathrm{nd}}$. Then Equation (7) is simplified as Equation (8), where $\Delta_C(\boldsymbol{x})$ is the volume of cell $C \in \mathcal{C}_{\mathrm{nd}}$.

$$\mathrm{EIPV}(\boldsymbol{x}_{t+1}|\mathcal{D}) = \sum_{C \in \mathcal{C}_{\mathrm{nd}}} \Delta_C(\boldsymbol{x}) = \sum_{C \in \mathcal{C}_{\mathrm{nd}}} \int_C \mathrm{PV}_{\boldsymbol{v}_c}(\boldsymbol{y}) p(\boldsymbol{y}|\mathcal{D}) \mathrm{d}\boldsymbol{y}. \tag{8}$$

In Fig. 6(b), the green node maximizes the expected improvement and therefore it is the candidate Pareto point.

Now we have clarified the concept of the improvement of Pareto hyper-volume. The next step is to define the probability model $p(\boldsymbol{y}|\mathcal{D})$ and to further deduce the concrete form of the expected improvements.

In traditional works [11], [12], the multiple design objectives are defined as independent Gaussian process models, though in real applications, they are mostly correlated. For example, to reduce system latency, we may want to increase the system parallelism which means we will consume much more on-board resources, especially LUTs. Therefore, latency and resource consumption are negatively correlated. But power and resource consumption are positively correlated, because more resource consumptions may lead to higher power costs. $p(\boldsymbol{y}|\mathcal{D})$ is modeled as a correlated multi-objective Gaussian model [17] in this paper, as shown in Equation (9).

$$p(\boldsymbol{y}|\mathcal{D}) = \mathcal{N}(y^1, \ldots, y^M; \boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{9}$$

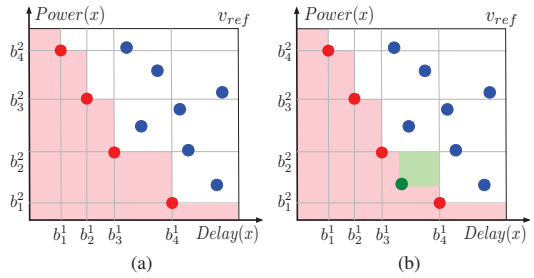


Fig. 6 An example of minimizing power and delay. The value space is divided into cells according to the locations of the current Pareto set. (a) Red points are Pareto points and blue points are dominated. Blank cells are dominated while light red cells are not. Volume of the blank cells is the current Pareto hyper-volume. (b) Green point is predicted to be the Pareto-optimal configuration and the light green cell is the corresponding EIPV.

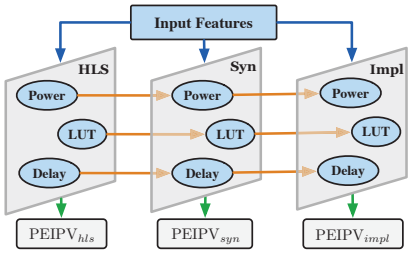


Fig. 7 The combined models, with three fidelities and three objectives. The orange lines represent the non-linear relationships. The blue lines represent the inputs. Each fidelity has a function PEIPV.

where $\boldsymbol{\mu}$ is the mean vector with length M and each element μ_i in it is the mean value of objective f^i. The covariance matrix $\boldsymbol{\Sigma}$ is non-diagonal. Specifically, the covariance value definition is $\boldsymbol{\Sigma}_{i,j} = \mathrm{Cov}(f^i(\boldsymbol{x}),\ f^j(\boldsymbol{x}')) = \mathrm{K}_{i,j}k_C(\boldsymbol{x},\boldsymbol{x}')$. $\mathrm{K}_{i,j}$ is the similarity between objectives i and j and can be obtained by maximizing likelihood estimation. k_C is a covariance function over $\mathcal{X}$ and can be defined as ARD Matérn $5/2$ kernel to avoid unrealistic smoothness.

C. Combined Model

Our optimization model has two dimensions, one for objectives and one for fidelities. Fig. 7 visualizes the structures of combined models. At each fidelity, all the objectives form a correlated multi-objective optimization model. Its expected improvement function of Pareto hyper-volume is denoted as $\mathrm{EIPV}_i(\boldsymbol{x}_{t+1}|\mathcal{D})$. Obtaining results at different fidelities cost different running times. To characterize this kind of inter-stage differences, an additional penalty term is applied to augment $\mathrm{EIPV}_i(\boldsymbol{x}_{t+1})$ as $\mathrm{PEIPV}_i(\boldsymbol{x}_{t+1}|\mathcal{D})$, as shown in Equation (10).

$$\mathrm{PEIPV}_i(\boldsymbol{x}_{t+1}|\mathcal{D}) = \mathrm{EIPV}_i(\boldsymbol{x}_{t+1}|\mathcal{D}) \cdot \frac{T_{impl}}{T_i}, i \in \{hls, syn, impl\}, \tag{10}$$

where T_i is the time of running the FPGA design tool from scratch to fidelity i. Finally, PEIPV in Equation (10) is used as the EI function in Algorithm 2. For the designs that violate the placement or routing rules, no valid reports are returned from the FPGA tool. Their simulation performance is set to be $10\times$ worse than the current worst-case, to punish the illegal designs and teach the models.

V. Experimental Results

In our experiments, the initial design space is defined by specifying all of the possible locations of directives and their factors in YAML files. We parse the YAML files, and convert the directives to feature vectors and HLS TCL files. The target FPGA board is Xilinx Virtex-7 VC707. The FPGA tool is Xilinx Vivado HLS 2018.2.

A. Benchmarks and Baselines

We conduct experiments on six benchmarks. Five benchmarks are from open-source FPGA application benchmark MachSuite [18], *i.e.*, `GEMM`, `SORT_RADIX`, `SPMV_ELLPACK`, `SPMV_CRS`, and `STENCIL3D`. Another benchmark is `iSmart2` [19], an object detection Deep Neural Network model deployed on FPGA. These benchmarks are composed of some for-loops, several matrix computations, and memory communications. For each for-loop, we consider unrolling and pipelining (with initialization interval). For each array, we consider array partitioning. Each benchmark contains a large number of possible configurations. Our tree-based method can prune the design space by a lot. Take `SORT_RADIX` as an example, the design space is pruned from more than 3.8×10^{12} to 20000 configurations.

Four popular and representative methods are compared with our method. [12], shorted as FPL18, is also based on Bayesian methods and Gaussian process. The authors build linear multi-fidelity and independent multi-objective models. [20], abbreviated as DAC19, defines several regression models to guide the FPGA HLS designs with existing ASIC designs. Although our starting points are different, their methods are transferable. Post-HLS reports can be regarded as the ASIC implementations in their implementation, to predict the post-Implementation reports. Artificial neural network (ANN) and Boosting tree (BT) methods have been used in [7]–[9] to guide the back-end designs and achieved good performances. For these regression algorithms, some configurations are randomly sampled from the design space to initialize these models. We use the post-implementation reports as the regression targets. For each objective, we build one model. After all of the models are trained, the whole design space is fed into these models to predict the Pareto points. For fairness, all of these algorithms use the same feature encodings and design spaces with our method.

B. Experimental Settings

For our method and FPL18 [12], we run 10 tests on each benchmark and the results reported in this paper are the averages. For each benchmark, 8 configurations are randomly sampled to initialize the models and the maximum optimization step is 40.

For ANN, we design a model with 2 hidden layers. We train the model with $\{500, 1000, \ldots, 5000\}$ times. For the Boosting method used in [7]–[9], we run a group of experiments, with tree depth from 1 to 6, and learning rate in $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. In DAC19 [20], different numbers of initial sets are sampled to build the models. Therefore, the number of initial sets is also considered as a hyperparameter, *i.e.*, $\{3, 4, \ldots, 11\}$. In experiments of ANN, Boosting, and DAC19, for each benchmark, the number of initialization configurations is 48.

Two metrics are used to measure the performance: average distance to reference set (ADRS) and overall running time. ADRS computes the distance between the learned Pareto set and the real Pareto set [20].

$$\mathrm{ADRS}(\Gamma, \Omega) = \frac{1}{|\Gamma|}\sum_{\gamma\in\Gamma}\min_{\omega\in\Omega} f(\gamma, \omega), \tag{11}$$

where Ω is the learned Pareto set, Γ is the real Pareto set, $f(\gamma, \omega)$ is the distance between two points, $\gamma \in \Gamma$ and $\omega \in \Omega$, $|\Gamma|$ is the number of points in Γ. Overall running time is the total time needed to get all results, including initialization and iterative optimization.

C. Results and Analyses

Two examples are plotted in Fig. 8 to show the learned Pareto points. For easy visualizations, three objectives are plotted in two figures. The right side parts of the design spaces are not plotted to help zoom in the figures. The results demonstrate that our learned Pareto points are

TABLE I Normalized Experimental Results

Model	Normalized ADRS					Normalized Standard Deviation of ADRS					Normalized Overall Running Time				
	Ours	FPL18	ANN	BT	DAC19	Ours	FPL18	ANN	BT	DAC19	Ours	FPL18	ANN	BT	DAC19
GEMM	**0.27**	0.50	1.00	0.65	1.08	**0.12**	0.46	1.00	0.37	0.90	**0.68**	0.83	1.00	1.00	7.00
iSmart2	**0.65**	0.68	1.00	1.28	1.49	**0.20**	0.75	1.00	1.10	1.24	**0.42**	0.88	1.00	1.00	7.00
SORT_RADIX	**0.64**	0.72	1.00	1.09	0.94	**0.48**	0.57	1.00	1.72	2.28	**0.34**	0.47	1.00	1.00	7.00
SPMV_ELLPACK	**0.19**	0.47	1.00	0.22	1.21	0.09	0.24	1.00	**0.06**	0.99	0.65	**0.42**	1.00	1.00	7.00
SPMV_CRS	**0.22**	0.29	1.00	2.09	1.15	**0.03**	0.26	1.00	2.09	1.52	**0.72**	0.90	1.00	1.00	7.00
STENCIL3D	**0.39**	0.41	1.00	0.40	0.41	0.03	0.57	1.00	**0.00**	0.05	0.44	**0.41**	1.00	1.00	7.00
Average	**0.39**	0.51	1.00	0.96	1.05	**0.16**	0.47	1.00	0.89	1.16	**0.54**	0.65	1.00	1.00	7.00

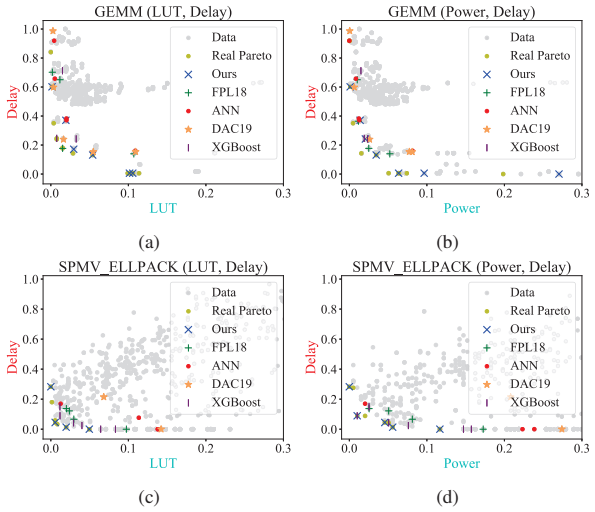


Fig. 8 Learned Pareto points of GEMM and SPMV_ELLPACK.

much more closer to the reference points. All of the statistical results are listed in TABLE I, while expressed as ratios to the results of ANN.

As shown in TABLE I, our method outperforms all of these baselines by a lot. Firstly, compared with FPL18 [12], our method can achieve much better results on all benchmarks. That is because we consider practical non-linear and correlated relationships in real applications. Secondly, the other three methods are also worse than ours, because they cannot handle complex relationships between multiple fidelities. Thirdly, for benchmarks with complicated code structures, the models without GP models are inferior. For example, the irregular memory accesses of SORT_RADIX bring great challenges to ANN, Boosting tree, and DAC19. The results prove that our model is general enough to handle various applications Our method also achieves much better stability according to the standard deviations of ADRSs.

More samplings (longer running times) would introduce more information into learning models. The averages of overall running time are listed in TABLE I to show that our method can also save time. For ANN and Boosting tree, the sizes of their training data set are fixed and equal. For DAC19, the size of one training data set equals to ANN. But it has $3 \sim 11$ training sets, therefore the average running time is 7 times (*i.e.*, $(3 + 11)/2 = 7$) greater than ANN and Boosting tree. Our method runs fewer Vivado flows compared with FPL18 on four benchmarks. Although FPL18 costs less average times on SPMV_ELLPACK and STENCIL3D, the corresponding running times of our method are comparative. Our average running time is the best one. Considering the improvements in ADRS, these running time costs are worthy and satisfying.

VI. Conclusion

In this paper, we solve the problem of FPGA HLS directives design optimization. A novel tree-based pruning method is proposed, which can significantly prune the design space. Non-linear multi-fidelity models can handle the strong nonlinearities among the multiple fidelities. To the best of our knowledge, correlated multi-fidelity model is introduced into the HLS directive optimization domain for the first time and has been proven to be effective. We hope this paper will stimulate new research directions in this domain.

References

[1] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proc. DAC*, 2019, pp. 1–6.

[2] J. Zhao, L. Feng, S. Sinha, W. Zhang, Y. Liang, and B. He, "COMBA: A comprehensive model-based analysis framework for high level synthesis of real applications," in *Proc. ICCAD*, 2017, pp. 430–437.

[3] ——, "Performance modeling and directives optimization for high level synthesis on FPGA," *IEEE TCAD*, 2019.

[4] Q. Sun, T. Chen, J. Miao, and B. Yu, "Power-driven DNN dataflow optimization on FPGA," in *Proc. ICCAD*, 2019, pp. 1–7.

[5] L. Ferretti, G. Ansaloni, and L. Pozzi, "Lattice-traversing design space exploration for high level synthesis," in *Proc. ICCD*, 2018, pp. 210–217.

[6] K. O'Neal, M. Liu, H. Tang, A. Kalantar, K. DeRenard, and P. Brisk, "HLSPredict: cross platform performance prediction for FPGA high-level synthesis," in *Proc. ICCAD*, 2018, pp. 1–8.

[7] 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," in *Proc. FCCM*, 2018, pp. 129–132.

[8] J. Zhao, T. Liang, S. Sinha, and W. Zhang, "Machine learning based routing congestion prediction in FPGA high-level synthesis," in *Proc. DATE*, 2019, pp. 1130–1135.

[9] E. Ustun, S. Xiang, J. Gui, C. Yu, and Z. Zhang, "Lamda: Learning-assisted multi-stage autotuning for FPGA design closure," in *Proc. FCCM*, 2019, pp. 74–77.

[10] B. C. Schafer and Z. Wang, "High-level synthesis design space exploration: Past, present and future," *IEEE TCAD*, 2019.

[11] C. Lo and P. Chow, "Model-based optimization of high-level synthesis directives," in *Proc. FPL*, 2016, pp. 1–10.

[12] ——, "Multi-fidelity optimization for high-level synthesis directives," in *Proc. FPL*, 2018, pp. 272–2727.

[13] T. Chen, B. Lin, H. Geng, and B. Yu, "Sensor drift calibration via spatial correlation model in smart building," in *Proc. DAC*, 2019, pp. 1–6.

[14] W. Lyu, F. Yang, C. Yan, D. Zhou, and X. Zeng, "Batch Bayesian optimization via multi-objective acquisition ensemble for automated analog circuit design," in *Proc. ICML*, 2018, pp. 3312–3320.

[15] Y. Ma, S. Roy, J. Miao, J. Chen, and B. Yu, "Cross-layer optimization for high speed adders: A pareto driven machine learning approach," *IEEE TCAD*, vol. 38, no. 12, pp. 2298–2311, 2018.

[16] A. Shah and Z. Ghahramani, "Pareto frontier learning with expensive correlated objectives," in *Proc. ICML*, 2016, pp. 1919–1927.

[17] E. V. Bonilla, K. M. Chai, and C. Williams, "Multi-task Gaussian process prediction," in *Proc. NIPS*, 2008, pp. 153–160.

[18] "MachSuite," https://breagen.github.io/MachSuite/.

[19] "iSmartDNN," https://github.com/onioncc/iSmartDNN.

[20] S. Liu, F. Lau, and B. C. Schafer, "Accelerating FPGA prototyping through predictive model-based HLS design space exploration," in *Proc. DAC*, 2019, p. 97.