PPATuner: Pareto-driven Tool Parameter Auto-tuning in Physical Design via Gaussian Process Transfer Learning

Hao Geng
ShanghaiTech & CUHK

Qi Xu
USTC

Tsung-Yi Ho
CUHK

Bei Yu
CUHK

Abstract

Thanks to the amazing semiconductor scaling, incredible design complexity makes the synthesis-centric very large-scale integration (VLSI) design flow increasingly rely on electronic design automation (EDA) tools. However, invoking EDA tools especially the physical synthesis tool may require several hours or even days for only one possible parameters combination. Even worse, for a new design, oceans of attempts to navigate high quality-of-results (QoR) after physical synthesis have to be made via multiple tool runs with numerous combinations of tunable tool parameters. Additionally, designers often puzzle over simultaneously considering multiple QoR metrics of interest (e.g., delay, power, and area). To tackle the dilemma within finite resource budget, designing a multi-objective parameter auto-tuning framework of the physical design tool which can learn from historical tool configurations and transfer the associated knowledge to new tasks is in demand. In this paper, we propose PPATuner, a Pareto-driven physical design tool parameter tuning methodology, to achieve a good trade-off among multiple QoR metrics of interest (e.g., power, area, delay) at the physical design stage. By incorporating the transfer Gaussian process (GP) model, it can autonomously learn the transfer knowledge from the existing tool parameter combinations. The experimental results on industrial benchmarks under the 7nm technology node demonstrate the merits of our framework.

1 Introduction

Modern synthesis-centric chip design flow benefits a lot from the continuous scaling of feature size. However, just like a double-edged sword, the aggressive scaling also brings side effects. Ever-increasing design complexity challenges the current electronic design automation (EDA) solutions. Accordingly, EDA tools have been upgraded in terms of quality of outcomes by ceaselessly integrating sophisticated algorithms and optimization techniques at all stages of the design flow. Concurrently, countless tunable options that can remarkably affect the quality-of-results (QoR) are also provided as hints for designers. This makes the parameter tuning process more intractable in turn.

On the other hand, physical design (PD) flow comprising partitioning, floorplanning, placement, clock tree synthesis, and routing via corresponding tools, heavily impacts the post-layout QoR.

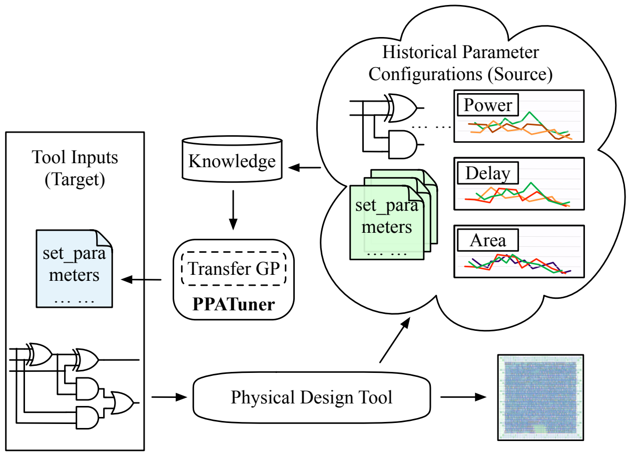


Figure 1: The Overview of PPATuner.

Simultaneously, it costs much more time than other stages in the design flow. Another status quo is that the tool used in such momentous step of the whole design flow typically has over ten thousand parameter-option combinations. Applying the proposed methodology in the physical tool parameter tuning has considerably practical significance. In this paper, we focus on the PD tool parameter auto-tuning issue.

An analogous problem to the parameter tuning issue of a PD tool is design space exploration (DSE) [1] which navigates the optimal solutions (e.g., values, designs, or tool configurations) in large design space. Likewise, DSE in our field often relates to multi-objective "black-box" optimization. Note that a surrogate function is proposed in lieu of the objective function in such optimization issues because there are no explicit mathematical expressions of the latter. Therefore, Pareto-driven "black-box" optimization approaches proposed under the DSE problem formulation may be prospective recipes for the parameter auto-tuning issue.

Recently, loads of prior arts [2–9] start to focus on the EDA tool parameter tuning. Powered by machine learning techniques, [6–9] have attained phenomenal success.

Nevertheless, the defects emerge as well. The performance of machine learning models is jeopardized due to a lack of training parameter configurations with associated QoR metrics values reported by the tool when handling new designs. As mentioned, running EDA tools to obtain data for model calibration is costly during the exploration of a new design or new preference of QoR metrics of interest. On the other hand, since similar parameter configurations have been already tried quite a few times in navigation of previous designs, there are quantities of prior data available. Although designs differ from one to another, the impact of architecture properties of similar designs like pipelining may have little change. Some studies like [9] discover that the influence of parameters can

DAC '22, July 10–14, 2022, San Francisco, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530602

be consistent for different designs, let alone similar designs. This paves the way for transferring knowledge from prior tool parameter configurations.

To address the above concern, in this paper, we propose PPATuner, a Pareto-driven parameter tuning method of the PD tool, which harnesses the transfer Gaussian Process (GP) as the surrogate model. The transfer GP model learns the historical data and transfers the useful knowledge in an automatic fashion. For a better understanding of the role PPATuner plays, we visualize the tuner for a physical design tool in Figure 1. Our contributions are summarized as follows:

- A Pareto-driven iterative auto-tuning framework is proposed, which provides a good trade-off among multiple QoR metrics of interest (e.g., power dissipation, delay, and area utilization).
- The tuning framework incorporates a transfer Gaussian process to achieve knowledge transfer from old tasks to new tasks.
- Our framework takes advantage of the knowledge from existing parameter configurations so that optimal or near-optimal parameter configurations can be found and computational resources are conserved.

The rest of this paper is organized as follows. In Section 2, we will give the problem formulation and the backgrounds of the Gaussian process and transfer learning. Section 3 will describe the proposed PPATuner in detail. Section 4 will show the experimental results, followed by the conclusion in Section 5.

2 Preliminaries

In this section, we give our problem formulation and some background knowledge about the Gaussian process and transfer learning in our context, for a better understanding.

2.1 Gaussian Process

A Gaussian process (GP) over a real function $f(\boldsymbol{x})$ can be described by its mean function $u(\cdot)$ and covariance function $k(\cdot,\cdot)$. In our multiple QoR metrics case, we model each QoR metric as a draw from an independent GP distribution. Assume we try to apply the Gaussian process (GP) model to approximate the "black-box" function of power. $\boldsymbol{X}$ is the training set filled with tool parameter configurations and $\boldsymbol{Y}$ is the power consumption of trained set $\boldsymbol{X}$ (obtained by invoking a PD tool). Given a new parameter configuration $\boldsymbol{x}$, the GP will make the inference to predict the mean and variance as follows.

$$\begin{aligned} u(\boldsymbol{x}) &= k(\boldsymbol{x},\boldsymbol{X})^\top(k(\boldsymbol{X},\boldsymbol{X})+\sigma^2\boldsymbol{I})^{-1}Y, \\ \sigma^2(\boldsymbol{x}) &= k(\boldsymbol{x},\boldsymbol{x}) - k(\boldsymbol{x},\boldsymbol{X})^\top(k(\boldsymbol{X},\boldsymbol{X})+\sigma^2\boldsymbol{I})^{-1}k(\boldsymbol{x},\boldsymbol{X}), \end{aligned} \tag{1}$$

where the mean value $m(\boldsymbol{x})$ is the predicted value, while the variance $\sigma^2(\boldsymbol{x})$ refers to the uncertainty of the prediction. This is the advantage of utilizing the GP that it provides a well-calibrated prediction as well as a predictive uncertainty. By virtue of this, GP is frequently employed in the learning-based optimization framework [10]. Another merit of GP is data efficiency. Opposite to other learning models like neural networks, GP requires less training data and has a smaller number of (hyper-)parameters to optimize. The above properties make GP be suitable for the parameter tuning task.

2.2 Transfer Learning

Transfer learning represents a set of techniques aiming at improving the learning of a target domain by using the latent similarity among the data from the source and target domains. It is widely used in scenarios when the data from the target domain is insufficient and the data from the source domain is adequate. In our parameter tuning case, various parameter configurations have been tried, while the exploration for new designs or parameter configurations cannot be intensively performed due to the resource limitation and stringent time-to-market constraint. Ergo, transfer learning-like schemes are in desire. A large quantity of literature has studied the effectiveness of knowledge transfer in other domains like image processing [11], while few works have discussed the transferable issue in PD tool parameter auto-tuning.

2.3 Problem Formulation

Our parameter tuning problem is to optimize multiple QoR metrics (or design objectives) in as few evaluations by the PD tool as possible so that the total expense is minimized. Typically, the associated solutions (a.k.a. Pareto-optimal solutions) whose QoR metrics can be optimized without worsening at least one of the rest constitute a Pareto set. Our tuning framework is to derive an approximated set for Pareto-optimal parameter configurations. The quality of these parameter settings (or performance of the proposed tuning framework) is often evaluated with the hyper-volume error and the average distance from reference set (ADRS) metrics.

The hyper-volume error for a Pareto set approximation $\hat{\mathcal{P}}$ is defined as

$$e = \frac{H(\mathcal{P}) - H(\hat{\mathcal{P}})}{H(\mathcal{P})}, \tag{2}$$

where $\mathcal{P}$ is the golden Pareto-optimal set, and $H(\mathcal{P})$ is the ground-truth of hyper-volume. If a solution set $\mathcal{P}'$ is better than $\mathcal{P}''$, $H(\mathcal{P}')$ is greater than $H(\mathcal{P}'')$ in the QoR metrics minimization case.

Assume two of the QoR metrics of interest are power P and area A. The ADRS performance indicator can be calculated as follows. Given the Pareto-optimal set $\mathcal{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \dots \mid \boldsymbol{a} = (P, A)\}$ as reference and an approximated Pareto-optimal set $\hat{\mathcal{P}} = \{\hat{\boldsymbol{p}}_1, \hat{\boldsymbol{p}}_2, \dots \mid \hat{\boldsymbol{p}} = (\hat{P}, \hat{A})\}$ in bi-objective optimization,

$$\mathrm{ADRS}(\mathcal{A}, \hat{\mathcal{P}}) = \frac{1}{|\mathcal{A}|}\sum_{\boldsymbol{a}\in\mathcal{A}} \min_{\hat{\boldsymbol{p}}\in\hat{\mathcal{P}}} \delta(\boldsymbol{a}, \hat{\boldsymbol{p}}), \tag{3}$$

where $\delta(\boldsymbol{a}, \hat{\boldsymbol{p}}) = \max\left\{\left|\frac{P-\hat{P}}{P}\right|, \left|\frac{A-\hat{A}}{A}\right|\right\}$.

With the aforementioned knowledge, our problem can be formulated as follows.

Problem 1 (Pareto-driven Parameter Auto-tuning). Given a specific amount of tool parameter configurations from a target domain, the Pareto-driven parameter auto-tuning for the physical design tool is expected to autonomously seek for the Pareto-optimal parameter configurations in target parameter space so that high QoR can be achieved.

3 The Proposed Tuning Framework

In a spirit of Bayesian optimization, our tuning framework provides a solution to the PD tool parameter tuning issue: an acquisition procedure, which takes as input cheap probabilistic surrogate models of the "black-box" functions (i.e., the mappings from PD tool parameter configurations to post-layout QoR metrics), repeatedly scores promising tool parameter configurations by performing an explore-exploit trade-off. In the following, we will first introduce the surrogate model: the transfer Gaussian process. Then, the working flow of the whole tuning framework is characterized detailedly.

3.1 The Transfer Gaussian Process

As aforementioned, the Gaussian process belongs to non-parametric models. It is suitable to be the core of our transfer learning scheme in the PD tool parameter tuning issue. The main idea underneath is to learn a transfer kernel to model the correlation of the output predictive QoR metric values when the input parameter configurations come from prior trials (i.e., the source domain or task) and current parameter tuning jobs (i.e., the target domain or task), which can be considered as a measure of similarity between tasks. What to transfer is based on how similar the source is to the target. If the tasks are very similar, the knowledge would be directly transferred from the source data. By contrast, if the tasks are dissimilar, the model would only transfer the shared coefficients in the kernel function.

Suppose $y_i^{\mathcal{S}}$ denotes the observed output (e.g., one of QoR metric of interest like power value) corresponding to the i-th input tool parameter configuration vector $\boldsymbol{x}_i^{\mathcal{S}}$ in the source domain, while $y_j^{\mathcal{T}}$ refers to the observed QoR metric value of the j-th input $\boldsymbol{x}_j^{\mathcal{T}}$ in the target domain. There exists the underlying "black-box" function $f^{\mathcal{S}}$ between the input and output for the source task. Assume we have N tool parameter configurations for the source task and M instances for the target, then $\boldsymbol{f}^{\mathcal{S}}$ is an N-dimensional vector and $\boldsymbol{f}^{\mathcal{T}}$ an M-dimensional vector.

In training, the conditional distribution $p\left(\boldsymbol{y}^{\mathcal{T}} \mid \boldsymbol{y}^{\mathcal{S}}, \boldsymbol{X}^{\mathcal{T}}, \boldsymbol{X}^{\mathcal{S}}\right)$ is what we need to consider. Let $\boldsymbol{f} = \left(\boldsymbol{f}^{\mathcal{S}}, \boldsymbol{f}^{\mathcal{T}}\right)$ and we can define a Gaussian process over $\boldsymbol{f}$ like

$$p(\boldsymbol{f} \mid \boldsymbol{X}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{f} \mid \boldsymbol{0}, \boldsymbol{K}), \tag{4}$$

with the kernel matrix $\boldsymbol{K}$ for transfer learning and $\boldsymbol{\theta}$ the whole hyper-parameters. More specifically,

$$\boldsymbol{K}_{nm} \sim k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right)\left(2e^{-\eta\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right)\phi} - 1\right), \tag{5}$$

where $\eta\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = 0$ if $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$ are from the same task domain, otherwise $\eta\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = 1$. The parameter ϕ represents the dissimilarity between $\mathcal{S}$ and $\mathcal{T}$. The intuition behind the kernel definition is that the additional factor makes the correlation between points of the different tasks are less or equal to the correlation between the ones in the same task. To estimate the (dis)similarity with limit amount of tool runs, a Bayesian approach is exploited to combat this difficulty. Reasonably assume ϕ follows a Gamma distribution, i.e., $\phi \sim \Gamma(b, a)$. We can get the kernel by computing the formulation as

$$\widetilde{\boldsymbol{K}}_{nm} = \mathbb{E}\left[\boldsymbol{K}_{nm}\right] = k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right)\int\left(2e^{-\eta\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right)\phi} - 1\right)\phi^{b-1}\frac{e^{-\phi/a}}{a^b\Gamma(b)}\mathrm{d}\phi. \tag{6}$$

Then, integrating out ϕ, we obtain the kernel

$$\widetilde{\boldsymbol{K}}_{nm} = \begin{cases} k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right)\left(2\left(\frac{1}{1+a}\right)^b - 1\right), \eta\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = 1 \\ k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right), \quad \text{otherwise} \end{cases} \tag{7}$$

As can be observed in Equation (7), the proposed transfer kernel models the correlation of outputs based on not only the similarity between inputs but also the similarity between tasks. In addition, it measures both positive and negative correlations between tasks and thus has a stronger expression ability.

When doing inference, given a test parameter configuration $\boldsymbol{x}$ in the target task, the predictive distribution $p\left(y \mid \boldsymbol{X}, \boldsymbol{y}\right)$ needs to determine. Note that the inference process of the model is the same as that in standard GP models. Hence, the mean and variance of the predictive distribution of the target task parameter configuration are

$$\begin{aligned} u(\boldsymbol{x}) &= k(\boldsymbol{x}, \boldsymbol{X})^\top\left(\widetilde{\boldsymbol{K}} + \Lambda\right)^{-1}\boldsymbol{y}, \\ \sigma^2(\boldsymbol{x}) &= c - k(\boldsymbol{x}, \boldsymbol{X})^\top\left(\widetilde{\boldsymbol{K}} + \Lambda\right)^{-1}k(\boldsymbol{x}, \boldsymbol{X}), \end{aligned} \tag{8}$$

where $\Lambda = \begin{pmatrix} \beta_s^{-1}\boldsymbol{I}_N & \boldsymbol{0} \\ \boldsymbol{0} & \beta_t^{-1}\boldsymbol{I}_M \end{pmatrix}$ and $c = k(\boldsymbol{x}, \boldsymbol{x}) + \beta_t^{-1}$ and $k(\boldsymbol{x}, \boldsymbol{X})$ can be computed by the transfer kernel (Equation (6)). β_s and β_t are two hyper-parameters which can be learned by maximizing the marginal likelihood of data of the target task during training.

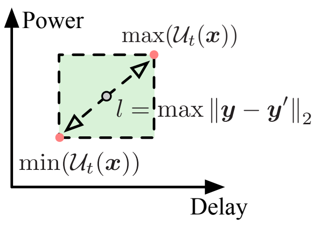
In general, the kernel function in transfer GP expresses that for parameter configurations $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$ that are similar, the corresponding QoR values $y(\boldsymbol{x}_n)$ and $y(\boldsymbol{x}_m)$ will be more strongly correlated than for dissimilar points. Surely, the correlation between $y(\boldsymbol{x}_n)$ and $y(\boldsymbol{x}_m)$ also depends on which tasks the inputs parameter configurations $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$ come from and how similar the tasks are. As a result, the transfer kernel represents that for parameter configurations $\boldsymbol{x}_n$ and $\boldsymbol{x}_m$ from different tasks, how the associated QoR values y_n and y_m are correlated.

3.2 The Key Steps of PPATuner

PPATuner is to find the Pareto set by navigating a parameter space. After initialization, it iterates over several stages: model calibration, decision-making, and selection. When any stop criteria (e.g., all parameter configuration points have been either selected, dropped, or predicted as one of the Pareto set) is met, the proposed tuner will be terminated.

3.2.1 ***Initialization*** During the initialization, the proposed tuning framework interacts with the PD tool to acquire the golden QoR metric (e.g., area, power, and delay) values of a small number of parameter configurations which are randomly sampled from the parameter space $\mathbb{E}^{\mathcal{S}}$ of the source task and the space $\mathbb{E}^{\mathcal{T}}$ of the target task. In the multiple QoR metrics of interest scenario, we model each parameter-to-performance mappings as a draw from an independent GP distribution. These transfer Gaussian processes are first calibrated with the initial data. Afterward, PPATuner will start working iteratively.

3.2.2 ***Model Calibration*** The calibrated transfer GPs are exploited to predict the QoR metric values as well as corresponding uncertainties of the parameter configurations. It is worth mentioning that such inference is only for the parameter configurations that have not been discarded yet. Specifically, a vector $\boldsymbol{\mu}(\boldsymbol{x})$ where QoR


Power
$\max(\mathcal{U}_t(\boldsymbol{x}))$
$l = \max \|\boldsymbol{y} - \boldsymbol{y}'\|_2$
$\min(\mathcal{U}_t(\boldsymbol{x}))$
Delay
(a)

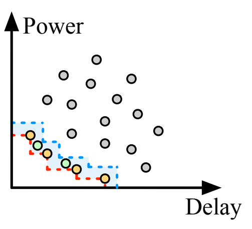


Figure 2: The visualizations of concepts in PPATuner frameworks. (a) The uncertainty region of a parameter configuration and its diameter; (b) δ-accurate Pareto frontier (blue dash lines) vs. golden Pareto frontier (red dash lines).

metric predictions (e.g., area, power, delay) of a parameter vector $\boldsymbol{x}$ are concatenated and a vector of corresponding standard deviations $\boldsymbol{\sigma}(\boldsymbol{x})$ are acquired. Consequently, a hyper-rectangle $\mathcal{R}(\boldsymbol{x})$ is constructed to imply the predictive uncertainty in QoR metric space for a parameter configuration $\boldsymbol{x}$, which is defined as follows.

$$\mathcal{R}(\boldsymbol{x}) \triangleq \{\boldsymbol{y} \mid \boldsymbol{\mu}(\boldsymbol{x}) - \tau^{\frac{1}{2}}\boldsymbol{\sigma}(\boldsymbol{x}) \preceq \boldsymbol{y} \preceq \boldsymbol{\mu}(\boldsymbol{x}) + \tau^{\frac{1}{2}}\boldsymbol{\sigma}(\boldsymbol{x})\}, \tag{9}$$

where y_i in $\boldsymbol{y}$ indicates the value of associated QoR metrics (area, power, or delay), and τ is a coefficient which identifies how large this region is in proportion to σ. As continuously updates transfer GPs with tool parameters and their golden QoR metric values through the PD tool, the predictive uncertainty level diminishes. On the back of ever-shrinking uncertainty, the uncertainty region of a parameter configuration $\boldsymbol{x}$ in the t-th iteration can be formulated like

$$\mathcal{U}_t(\boldsymbol{x}) \triangleq \mathcal{U}_{t-1}(\boldsymbol{x}) \cap \mathcal{R}(\boldsymbol{x}). \tag{10}$$

We can define that the initial $\mathcal{U}_{-1}$ is the entire QoR metric space $\mathbb{R}^n$ (of target task) with n the number of QoR metric of interests. The iterative intersection ensures that uncertainty regions will not increase. In one uncertainty region $\mathcal{U}_t(\boldsymbol{x})$ (e.g., light green area in Figure 2(a)), the upper-bound of QoR is the optimistic prediction $\min(\mathcal{U}_t(\boldsymbol{x}))$ and the lower-bound is the pessimistic prediction $\max(\mathcal{U}_t(\boldsymbol{x}))$, as showed in Figure 2(a). Such uncertainty information will be applied in the follow-up decision-making and selection steps.

3.2.3 *Decision-making*

One goal of this stage is to determine whether the undecided configuration points are dominated by others. More specifically, the undecided point $\boldsymbol{x}$, which is δ-dominated by any other point $\boldsymbol{x}'$ (undecided or determined to be Pareto-optimal), will be dropped and off the table in the following iterations. Above dropping rule can be written in Equation (11).

$$\max(\mathcal{U}_t(\boldsymbol{x}')) \preceq \min(\mathcal{U}_t(\boldsymbol{x})) + \boldsymbol{\delta}, \tag{11}$$

where $\boldsymbol{\delta}$ is the relaxation coefficient vector. This kind of coefficient provides users a precision controller of final Pareto solutions.

Another goal is to predict whether the undecided configuration points are Pareto-optimal with the least amount of points selected for the PD tool evaluation. If the undecided point $\boldsymbol{x}$ is not δ-dominated by any other point $\boldsymbol{x}'$ (undecided or determined to be Pareto-optimal), it will be regarded as the Pareto-optimal point. The mathematical expression of Pareto-optimal judging is shown in Equation (12).

$$\max(\mathcal{U}_t(\boldsymbol{x})) \preceq \min(\mathcal{U}_t(\boldsymbol{x}')) + \boldsymbol{\delta}. \tag{12}$$

Equation (12) indicates that the found parameter configurations are at most δ-worse than any Pareto-optimal points according to all of the QoR metrics. In other words, the solutions are δ-accurate. For a better understanding, we sketch the related concepts in Figure 2(b). In Figure 2(b), we exemplify a power vs. delay case. The orange points constitute the Pareto frontier, while the green points are the QoR metrics of parameter configurations we find. The parameter configurations whose QoR metric points (e.g., green dots) are in the light blue area are at least δ-accurate.

Algorithm 1 The PPATuner

Input: the historical parameter configuration dataset $\mathcal{D}^{\mathcal{S}}$, the target parameter configuration dataset $\mathcal{D}^{\mathcal{T}}$ for initialization, the number of maximum iterations T_{max}.
Output: the predicted Pareto-optimal parameter configuration set.

```
1:  # Initialization:
2:  Initializing the transfer GP models with $\mathcal{D}^{\mathcal{S}}$ and $\mathcal{D}^{\mathcal{T}}$;
3:  while existing not decided configurations or $t < T_{max}$ do
4:      # Model Calibration:
5:      The transfer GP models further calibration with $\boldsymbol{x}_t^s$ and associated golden QoR metric values;
6:      Uncertainty regions construction; ▷ Equations (9) and (10)
7:      # Decision-making:
8:      Dropping $\delta$-dominated configurations;    ▷ Equation (11)
9:      Pareto-optimal configurations deciding;    ▷ Equation (12)
10:     # Selection:
11:     Choosing and sending the configurations $\boldsymbol{x}_t^s$ for the PD tool evaluation;    ▷ Equation (13)
12:     $t \leftarrow t + 1$;
13: end while
```

3.2.4 *Selection*

After the finish of the decision-making step, a parameter configuration $\boldsymbol{x}_t^s$ (Pareto-optimal or undecided) with the longest diameter l (the concept of diameter is illustrated in Figure 2(a)) of its uncertainty region $\mathcal{U}_t(\boldsymbol{x})$ is sampled to feed the PD tool for golden QoR metric values. The sampling rule is illustrated in Equation (13).

$$\boldsymbol{x}_t^s \triangleq \underset{\boldsymbol{y},\boldsymbol{y}' \in \mathcal{U}_t(\boldsymbol{x})}{\arg\max} \left\|\boldsymbol{y} - \boldsymbol{y}'\right\|_2. \tag{13}$$

The selection scope is not limited to undecided parameter configurations but also covering the predictive Pareto-optimal ones. Therefore, it is not hard to imagine that the intuitive idea of the selection is choosing the points that may benefit searching the Pareto set.

3.3 The Overall Tuning Framework

For a better understanding, the whole framework is summarized in Algorithm 1. The historical data in the source task is exploited with a few parameter configurations in the target task to initialize the transfer GP model (lines 1-2). The whole auto-tuning algorithm works by iteratively performing the transfer GP model calibration, decision-making, and selection (lines 3-13). The iterative refinement will stop when reaches the maximum iterations or all the parameter

Table 1: The statistics of parameters of the PD tool on benchmarks. “-" in the table means the parameter is not considered in this benchmark.

Parameters	Source1		Target1		Source2		Target2	
	Min	Max	Min	Max	Min	Max	Min	Max
freq	950	1050	1000	1300	-	-	-	-
place_rcfactor	-	-	-	-	1.00	1.30	1.00	1.30
place_uncertainty	50	200	20	100	-	-	-	-
flowEffort	standard	extreme	standard	extreme	standard	extreme	standard	extreme
timing_effort	-	-	-	-	medium	high	medium	high
clock_power_driven	-	-	-	-	FALSE	TRUE	FALSE	TRUE
uniform_density	FALSE	TRUE	FALSE	TRUE	-	-	-	-
cong_effort	AUTO	HIGH	AUTO	HIGH	-	-	-	-
max_density	0.65	0.90	0.65	0.90	-	-	-	-
max_Length	160	310	160	300	250	350	250	350
max_Density	0.65	0.90	0.65	0.90	0.50	1.00	0.50	1. 00
max_transition	0.19	0.34	0.10	0.35	-	-	-	-
max_capacitance	0.08	0.13	0.08	0.20	0.07	0.12	0.05	0.15
max_fanout	25	50	25	50	25	40	25	39
max_AllowedDelay	0.00	0.25	0.00	0.25	0.06	0.12	0.00	0.12

configurations have been decided. Eventually, the predicted Pareto-optimal parameter combinations will be fed into the PD tools and then go through the physical design flow for evaluation.

It is worth mentioning that our approach also supports batch trials. For the side of the physical design tool, we have several software licenses so that the parallels trials are supported when enquiring the physical design tool (lines 10-11). The generation of benchmarks is also based on parallel invoking.

4 Experimental Results and Analyses

We implement our algorithm based on `Python` and the certain PD tool employed in the paper is Cadence Innovus Implementation System (version 16.2). The experimental platform is a Xeon Silver with the 4114 CPU processor. To evaluate the whole performance state of our framework, we compare it with the cutting-edge frameworks [6, 7, 9, 12] by exploring the parameter space of the PD tool on the target industrial benchmarks.

4.1 Benchmark Statistics

In our experiments, we exploit industry benchmarks under 7nm technology node, where three of them (i.e., Source1, Source2, and Target1) are generated by the same multiply accumulate (MAC) design (around 20k cells after placement) with different parameters and the Target2 is by the larger MAC design (about 67k cells).

Collecting configuration points with the golden QoR metric values is arduous due to the time-consuming running of a PD tool. More specially, 3 hours are required for smaller MAC design going through the PD flow, while the larger design costs 2 days by invoking the Innovus tool. Hence, exhaustively enumerating the feasible tool parameter configurations is unrealistic. Necessary parameter space pruning is inevitable. Recommended by the sophisticated IC designers, several vital parameters of the PD tool which impact final design quality are considered. Then, to effectively represent the parameter space, the Latin hyper-cube selecting scheme is exploited to choose the parameter configuration points. After that, parameter points will be given to the tool for the actual PD flow running so that their QoR metric values can be obtained. Until now, we have obtained offline benchmarks. With sorting and comparing, the golden QoR metric values of the Pareto frontier are achieved. To abstain from misunderstanding, “the golden values” or “real Pareto set” is defined as the best that can be found in the benchmarks.

For a better understanding, the statistics of parameters are listed in Table 1. There are 5000 parameter configuration points built upon the distinct combinations of 12 PD tool parameters in Source1 and Target1, respectively. Source2 and Target2 have 1440 and 727 points considering 9 parameters respectively. The data types of parameters include floating and integer. We supplement descriptions of some parameters for further reference. `flowEffort` configures the flow to give a trade-off between best-quality result and best turnaround time, and `uniform_density` enables even cell distribution for designs with less than 70% utilization, and `cong_effort` specifies the effort level of relieving congestion, and `max_density` controls the maximum density of local bins during global placement, while `max_Length` belongs to DRV rule parameters including `max_capacitance`/ `max_transition`/ `max_fanout`, and `max_Density` defines the maximum value for density (area utilization).

4.2 Whole Performance Against the SOTA

Four outstanding prior works are selected as baselines. TCAD’19 [12] is an active learning-based optimization framework, while MLCAD’19 [6] uses the classical Bayesian optimization flow with the lower confidence bound (LCB) function as the acquisition function. DAC’19 [7] which is inspired by the solution to matrix completion issue in recommender systems proposes the tensor decomposition and recommender system-based tuning algorithm. ASPDAC’20 [9] integrates the feature impotence-guided sampling strategy and utilizes ensemble boosting tree-based regressor to navigate the optimal parameter configurations. Among them, we have requested the source code of TCAD’19 and MLCAD’19, while DAC’19 and ASPDAC’20 are reimplemented by ourselves. Three metrics, hyper-volume error, the ADRS, and the number of tool runs, are used for performance indicators. Because the modeling time (i.e., calibrating model) of all methods requires far less time (e.g., usually in a few seconds) than tool runs (hours or days). We employ the number of tool runs to count the runtime overhead, which is a replacement to the conventional running time. To test the PPATuner, two practical scenarios requiring transferring knowledge are set up. Experimental results are shown quantitatively and qualitatively. Unfortunately, due to the space limitation, only the power vs. delay QoR metric space in Scenario Two is used as a visual example.

4.2.1 ***Scenario One: Same Design*** Given the same design, according to the reason like different designers’ preference of final design quality (e.g., area over power), distinct parameters will be considered to tune. We conduct the experiment to test the performance in this Scenario (denoted as **Scenario One**). Source1 is harnessed as the source task and Target1 is the target. 200 data points in the source task and no more than 5% of the data in the target task are used for the training of PPATuner. Table 2 records the associated results on Target1. Column “Multi-objective" lists three QoR metric spaces to be explored: area vs. delay, power vs. delay, and area vs. power vs. delay, and columns “HV”, “ADRS”, and “Runs” are the evaluation metrics referring to hyper-volume error, the average distance from the reference set, and the number of tool runs, respectively. As Table 2 suggests, with less runtime overhead (i.e., tool runs) on Target1, PPATuner averagely outperforms TCAD’19 with a 57.4% decrease on the hyper-volume error and a 41.0% drop on the ADRS value and reduces half the hyper-volume

Table 2: The whole performance comparison on Target1 benchmark.

Multi-objective	TCAD'19 [12]			MLCAD'19 [6]			DAC'19 [7]			ASPDAC'20 [9]			PPATuner		
	HV	ADRS	Runs	HV	ADRS	Runs	HV	ADRS	Runs	HV	ADRS	Runs	HV	ADRS	Runs
Area-Delay	0.142	0.068	509	0.122	0.061	400	0.129	0.075	627	0.133	0.063	400	0.050	0.041	251
Power-Delay	0.237	0.235	512	0.199	0.256	400	0.243	0.266	596	0.190	0.178	400	0.095	0.099	259
Area-Power-Delay	0.185	0.064	503	0.160	0.058	400	0.214	0.100	577	0.195	0.086	400	0.096	0.075	247
Average	0.188	0.122	508	0.160	0.125	400	0.195	0.147	600	0.173	0.109	400	**0.080**	**0.072**	**252.333**
Ratio	2.350	1.694	2.013	2.000	1.736	1.585	2.438	2.042	2.378	2.163	1.514	1.585	**1.000**	**1.000**	**1.000**

Table 3: The whole performance comparison on Target2 benchmark.

Multi-objective	TCAD'19 [12]			MLCAD'19 [6]			DAC'19 [7]			ASPDAC'20 [9]			PPATuner		
	HV	ADRS	Runs	HV	ADRS	Runs	HV	ADRS	Runs	HV	ADRS	Runs	HV	ADRS	Runs
Area-Delay	0.103	0.099	95	0.140	0.104	70	0.133	0.092	132	0.114	0.116	70	0.053	0.049	65
Power-Delay	0.113	0.092	93	0.121	0.094	70	0.127	0.077	125	0.122	0.119	70	0.039	0.050	63
Area-Power-Delay	0.107	0.084	88	0.099	0.074	70	0.105	0.103	137	0.140	0.086	70	0.059	0.042	58
Average	0.108	0.092	92	0.120	0.091	70	0.122	0.091	131.333	0.125	0.107	70	**0.050**	**0.047**	**62**
Ratio	2.160	1.957	1.484	2.400	1.936	1.129	2.440	1.936	2.118	2.500	2.277	1.129	**1.000**	**1.000**	**1.000**

error and 42.4% the ADRS compared with MLCAD'19. Besides, our method is better than DAC'19 with a drop of 59.0% hyper-volume error and more than 50.0% ADRS value and also excels ASPDAC'20 on all evaluation indicators.

4.2.2 ***Scenario Two: Similar Designs*** With the similar but different sized designs at hand, fast transferring knowledge from smaller design to larger design is essential. Another experiment is performed to evaluate the performance under this assumption (defined as **Scenario Two**). Source2 is the source task with Target2 as the target. Similarly, 200 data points in the source task and no more than 5% of the data in the target task are used for the training of PPATuner. Table 3 shows the corresponding results on Target2. It is crystal clear that with fewer PD tool runs on Target2, the proposed algorithm surpasses TCAD'19 with the average hyper-volume error and the ADRS value of 0.050 and 0.047, while it has a less hyper-volume error and smaller ADRS value compared to MLCAD'19. In addition, PPATuner behaves better than DAC'19 by decreasing 59.0% hyper-volume error and shrinking half ADRS value. With about 56.0% average reductions on hyper-volume error and ADRS, PPATuner shows a better performance against ASPDAC'20. The visualization of Pareto frontiers in the power vs. delay space is illustrated in Figure 3, which demonstrates that our learned Pareto points are much closer to the real ones.

5 Conclusion

In this paper, for the first time, we have proposed PPATuner, a Pareto-driven parameter auto-tuning flow of the PD tool, which adaptively transfers the knowledge from prior tuning tasks to new jobs via transfer Gaussian process model. The proposed framework not only reduces the human labor in the design loop, but also makes use of the existing experience so that the computational resource is saved within a shorter time-to-market. The experimental results on industrial benchmarks under the 7nm technology node have demonstrated the efficacy and effectiveness of the framework.

Acknowledgment

This work is supported by HiSilicon Technologies Co., ACCESS – AI Chip Center for Emerging Smart Systems, Hong Kong SAR, and The Research Grants Council of Hong Kong SAR (Project No. CUHK14209420).

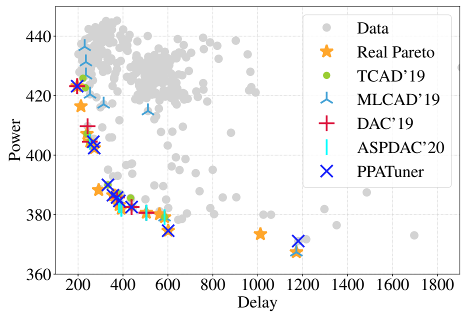


Figure 3: The visualization of Pareto frontiers in power vs. delay space on Target2 benchmark. The units for power and delay are mW and ns, respectively.

References

[1] H.-Y. Liu, I. Diakonikolas, M. Petracca, and L. Carloni, "Supervised design space exploration by compositional approximation of Pareto sets," in *Proc. DAC*, 2011, pp. 399–404.
[2] M. M. Ziegler, H.-Y. Liu, G. Gristede, B. Owens, R. Nigaglioni, and L. P. Carloni, "A synthesis-parameter tuning system for autonomous design-space exploration," in *Proc. DATE*, 2016, pp. 1148–1151.
[3] M. M. Ziegler, H.-Y. Liu, and L. P. Carloni, "Scalable auto-tuning of synthesis parameters for optimizing high-performance processors," in *Proc. ISLPED*, 2016, pp. 180–185.
[4] C. Xu, G. Liu, R. Zhao, S. Yang, G. Luo, and Z. Zhang, "A parallel bandit-based approach for autotuning FPGA compilation," in *Proc. FPGA*, 2017, pp. 157–166.
[5] C. H. Yu, P. Wei, M. Grossman, P. Zhang, V. Sarker, and J. Cong, "S2FA: An accelerator automation framework for heterogeneous computing in datacenters," in *Proc. DAC*, 2018, pp. 1–6.
[6] Y. Ma, Z. Yu, and B. Yu, "CAD tool design space exploration via Bayesian optimization," in *Proc. MLCAD*, 2019, pp. 1–6.
[7] J. Kwon, M. M. Ziegler, and L. P. Carloni, "A learning-based recommender system for autotuning design flows of industrial high-performance processors," in *Proc. DAC*, 2019, pp. 1–6.
[8] A. Agnesina, K. Chang, and S. K. Lim, "VLSI placement parameter optimization using deep reinforcement learning," in *Proc. ICCAD*, 2020, pp. 1–9.
[9] Z. Xie, G.-Q. Fang, Y.-H. Huang, H. Ren, Y. Zhang, B. Khailany, S.-Y. Fang, J. Hu, Y. Chen, and E. C. Barboza, "FIST: A feature-importance sampling and tree-based method for automatic design flow parameter tuning," in *Proc. ASPDAC*, 2020, pp. 19–25.

[10] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, “Taking the human out of the loop: A review of Bayesian optimization,” *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[11] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, “A comprehensive survey on transfer learning,” *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[12] 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, 2019.