ROI-HIT: Region of Interest-Driven High-Dimensional Microarchitecture Design Space Exploration

Xuyang Zhao, Tianning Gao, Aidong Zhao, Zhaori Bi, *Member, IEEE*, Changhao Yan, *Member, IEEE*, Fan Yang, *Member, IEEE*, Sheng-Guo Wang, *Life Senior Member, IEEE*, Dian Zhou, *Senior Member, IEEE*, and Xuan Zeng, *Senior Member, IEEE*

***Abstract*—Exploring the design space of RISC-V processors faces significant challenges due to the vastness of the high-dimensional design space and the associated expensive simulation costs. This work proposes a region of interest (ROI)-driven method, which focuses on the promising ROIs to reduce the over-exploration on the huge design space and improve the optimization efficiency. A tree structure based on self-organizing map (SOM) networks is proposed to partition the design space into ROIs. To reduce the high dimensionality of design space, a variable selection technique based on a sensitivity matrix is developed to prune unimportant design parameters and efficiently hit the optimum inside the ROIs. Moreover, an asynchronous parallel strategy is employed to further save the time taken by simulations. Experimental results demonstrate the superiority of our proposed method, achieving improvements of up to 43.82% in performance, 33.20% in power consumption, and 11.41% in area compared to state-of-the-art methods.**

***Index Terms*—Asynchronous parallel optimization, high-dimensional design space exploration (DSE), region of interest (ROI), RISC-V microarchitecture, variable selection (VS).**

I. Introduction

CUSTOMIZED microprocessors in embedded systems are in demand for a wide range of applications, such as mobile devices, medical equipment, and automotive systems [1]. Application-specific performance requirements necessitate tailored microprocessor designs. For instance, battery-powered devices prioritize low-power consumption for extended operation. Besides, the integration of microprocessors into system-on-chips (SoCs) requires them to harmonize with other system components and achieve target performance within constrained resources.

However, a tradeoff between performance metrics exists: enhancements in clock frequency often incur increased power consumption or enlarged die areas. So, when application requirements change, designs should be reoptimized. To address this challenge, multiobjective optimization considering performance, power, and area (PPA) is introduced. Multiobjective optimization empowers designers to select the most suitable solution based on their specific performance requirements. When these requirements change, designers can simply select a different design from the Pareto optimal set without the need for reoptimization.

In industry, it is crucial to meet strict and predetermined deadlines for product delivery. In recent years, RISC-V, an emerging instruction set architecture (ISA), has gained lots of attention in both academia and industry because of its open-source licenses and flexibility [2]. To achieve the agile design of the RISC-V SoC, the Berkeley architecture research (BAR) Group develops the Chipyard [3] framework. It uses Chisel [4] hardware construction language to generate the synthesizable and parameterizable processor cores (such as BOOM [5], [6] and Rocket [7]) in the SoC. This enables the generation of corresponding register-transfer level (RTL) designs for processor cores based on specified architecture parameters. So, there is a growing emphasis on identifying the Pareto optimal design parameters, namely, design space exploration (DSE) [1], in the early stages of the design process. By utilizing multiobjective optimization, DSE provides insights for subsequent design iterations.

Usually, processor simulators, such as GEM5 [8], are used to obtain the PPA metrics for microarchitectures. However, there exists a disparity between simulations and actual results. To achieve PPA results that closely resemble the actual hardware, the Chipyard framework leverages the HAMMER [9] framework for automated VLSI flow. This framework utilizes electronic design automation (EDA) tools for RTL simulation, logic synthesis, power analysis, and other tasks, ensuring the accuracy of PPA results. Chipyard supports automated generation from Chisel to layout. The flow incorporates the

Manuscript received 4 August 2024; accepted 4 August 2024. This work was supported in part by the National Natural Science Foundation of China (NSFC) Research Projects under Grant 62141407, Grant 62304052, and Grant 92373207. This article was presented at the International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS) 2024 and appeared as part of the ESWEEK-TCAD Special Issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding authors: Zhaori Bi; Xuan Zeng.)*

Xuyang Zhao, Aidong Zhao, Zhaori Bi, Changhao Yan, Fan Yang, and Xuan Zeng are with the School of Microelectronics, State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai 200433, China (e-mail: zhaori_bi@fudan.edu.cn; xzeng@fudan.edu.cn).

Tianning Gao is with the Department of Electrical Engineering, University of Texas at Dallas, Dallas, TX 75080 USA.

Sheng-Guo Wang is with the School of Microelectronics, State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai 200433, China and also with the College of Engineering, The University of North Carolina at Charlotte, Charlotte, NC 28223 USA.

Dian Zhou is with the School of Microelectronics, State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai 200433, China, and also with the Department of Electrical Engineering, University of Texas at Dallas, Dallas, TX 75080 USA.

Digital Object Identifier 10.1109/TCAD.2024.3443006

impact of various parameters of EDA tools, resulting in a parameter space exceeding 50 dimensions and a design space of approximately 10^{30}.

However, the high dimensionality poses challenges for multiobjective optimization algorithms, as their computational complexity typically increases linearly or exponentially with the number of dimensions. This high dimensionality leads to long modeling and optimization time. Moreover, given a flow, including RTL netlist generation with Chipyard BOOM generator, RTL simulation with Verilator [10], and logic synthesis with Cadence Genus, it typically requires 1-6 h of execution time. Consequently, the DSE process with hundreds of evaluation would take months of time. Therefore, it is imperative to develop high-dimensional microarchitecture DSE methods that can achieve satisfactory results with minimal evaluation counts.

Evolutionary-based approaches, such as NSGA-II [11] and MOEA/D [12], are commonly used multiobjective optimization methods. These methods generate a population of candidate solutions and iteratively refine them through crossover, mutation, and selection operators. However, they require a substantial number of simulation evaluations, which is impractical for the DSE based on the VLSI flow. In order to conduct the DSE within an acceptable time, researchers [13], [14] have utilized Bayesian optimization (BO). BO trains surrogate models to approximate the PPA results. These models are then used to optimize an acquisition function and identify promising candidate solutions for actual simulation. This approach significantly reduces the number of required simulations and improves sampling efficiency. BOOM-Explorer [13] proposes an efficient initial sampling method MicroAL via the characteristics of microarchitecture and utilizes the Gaussian process with deep kernel learning (DKL-GP) as surrogate models. BOOM-Explorer leverages the Chipyard framework, making it a valuable tool for exploring not only the BOOM core but also other RISC-V cores. GRL-DSE [14] introduces graph representation learning, which leverages a graph neural network (GNN) to project the microarchitecture design space into graph embedding space. While these methods effectively model the microarchitecture design space and reduce simulation requirements, both MicroAL and the GNN require training on the entire design space. Consequently, they struggle to handle the vast design space encountered in high-dimensional DSE problems.

Recent researches [15], [16], [17], [18], [19] have been proposed to address high-dimensional optimization problems. Trust region-based methods [15], [16] aim to tackle the vast design space. TuRBO [15] employs local models within trust regions instead of a global model to mitigate overemphasized global exploration. MORBO [16] leverages hypervolume contribution (HVC) to evaluate data point quality and extends TuRBO from single-objective to multiobjective optimization. Despite reducing the design space size, these local region-based algorithms still face high dimensionality. The modeling and optimization time complexity increases linearly or faster with the number of dimensions. Moreover, the existing data point distribution is not considered when determining the trust region size. This can result in trust regions with insufficient data points, leading to inaccurate models.

Embedding-based methods [17], [18] have been proposed to reduce the dimensionality of the search space. REMBO [17] leverages a random embedding matrix to map the high-dimensional space to a lower-dimensional one, resulting in faster optimization. BODi [18] employs dictionary embedding based on Hamming distance for dimensionality reduction and improved optimization efficiency. However, these methods require setting a fixed effective dimension for the embedding space in advance, which can be challenging in practice. Additionally, different microarchitecture design parameters have varying influences on performance outcomes. The aforementioned methods only consider the design space when training the embedding matrix, neglecting the correlations between design space and performance space and the different importance of different design variables. Recently, REMOTune [19] combines MORBO [16] and REMBO [17] to reduce both size and dimensionality of design space. Nonetheless, it still focuses solely on the design space, overlooking the distribution and performance of existing data points.

Moreover, with the popularity of multicore devices and cloud computing, parallel/batch simulation techniques can be used to conduct more simulations within a limited time budget, leading to better solutions. However, the simulation time can vary significantly based on different microarchitecture parameters. Existing batch optimization methods, such as MORBO [16] and REMOTune [19], are synchronous. It means these methods must wait for the slowest simulation in the batch to return before starting the next optimization iteration, which results in idle devices and wasted time.

This article proposes ROI-HIT, a region of interest (ROI)-driven high-dimensional microarchitecture DSE algorithm. By focusing on the promising ROIs, we reduce the over-exploration on the vast design space and advance the Pareto front (PF) more quickly. Consequently, we can obtain superior results within a constrained time budget. To further shorten the optimization time, we prune unimportant variables via a sensitivity matrix and reduce the number of dimensions used for modeling and optimization. For time-consuming VLSI flow simulations, an asynchronous parallel strategy is employed. Our contributions are summarized as follows.

1) We propose a novel space partitioning method based on a self-organizing map (SOM) tree to address the vast design space of high-dimensional microarchitecture DSE. A SOM network maps the high-dimensional design space into a 2-D feature space maintaining neighbor relationships and density. We define the feature regions covering the current Pareto set (PS) as ROIs, which are more promising to advance the PF. By exploiting the ROIs, we reduce the exploration for poor designs on the vast high-dimensional design space and improve the exploration efficiency.
2) We propose a variable selection (VS) method based on a sensitivity matrix to address the high dimensionality of DSE. To reduce the number of dimensions for modeling and optimization, a sensitivity matrix is calculated inside an ROI. ROI-HIT identifies important design variables according to the sensitivity matrix and merges unimportant variables into a 1-D auxiliary variable. Because

of the reduced dimensions, the runtime of ROI-HIT is shortened.

3) We propose an asynchronous parallel/batch framework to address the time-consuming simulations of DSE. Considering the tradeoffs between the PPA results, the expected hypervolume improvement (EHVI) acquisition function is optimized to choose the points to be simulated. Since the simulations based on the VLSI flow are independent and spend lots of time, we will simulate the next design if there is an idle worker, regardless of whether all the simulations have been completed. This mechanism further enhances the optimization efficiency.

The experimental results demonstrate that ROI-HIT outperforms the state-of-the-art methods, achieving 3.47%–9.19% improvement of hypervolume (HV) within the same time budget and 3.58×–341.68× speed-up of time in reaching the same HV. In terms of PPA metrics, ROI-HIT achieves improvements of up to 43.82% in performance, 33.20% in power consumption, and 11.41% in area. Compared with the official designs of RISC-V BOOM and Rocket microarchitecture, ROI-HIT can obtain 18.13%–30.05% improvement of power with higher performance and smaller area.

The remainder of this article is structured as follows: Section II introduces the problem formulation and the basics of BO. Section III details the ROI-HIT algorithm. Section IV presents the experimental results and compares the performances of our proposed algorithm with state-of-the-art algorithms. Section V concludes this article.

II. Background

A. Problem Formulation

The design space of microarchitecture is explored for better PPAs. So, the DSE problem can be formulated as a multiobjective optimization problem in

$$\text{minimize} \quad f^{(1)}(\boldsymbol{x}), \ldots, f^{(M)}(\boldsymbol{x}) \tag{1}$$

where $\boldsymbol{x}$ represents input parameters and $f^{(i)}(\cdot)$ represents objective functions. In this article, $\boldsymbol{x}$ involves the design parameters, such as *FetchWidth* and *DecodeWidth*, while $f^{(i)}$ includes power consumption, area, and the average cycle per instruction (CPI).

For DSE problems, it is not straightforward to determine the superiority of one design over another based solely on individual objective metrics, such as CPI, power consumption, or area. A design with a lower CPI may have higher-power consumption and a larger area, while a design with lower-power consumption and a smaller area may have a higher CPI. To assess the quality of multiobjective optimization, the concept of *domination* is introduced. For arbitrary data points $\boldsymbol{a}$ and $\boldsymbol{b}$, $\boldsymbol{a} \prec \boldsymbol{b}$ ($\boldsymbol{a}$ *dominates* $\boldsymbol{b}$) if

$$\begin{aligned} &\forall i \in \{1, \ldots, M\} \quad f^{(i)}(\boldsymbol{a}) \leq f^{(i)}(\boldsymbol{b}) \\ \text{and} \quad &\exists i \in \{1, \ldots, M\} \quad f^{(i)}(\boldsymbol{a}) < f^{(i)}(\boldsymbol{b}). \end{aligned} \tag{2}$$

The multiobjective optimization yields a PS, as defined in

$$\mathrm{PS} = \left\{\boldsymbol{x} \in X \mid \nexists\, \boldsymbol{x}' \in X,\ \boldsymbol{x}' \prec \boldsymbol{x}\right\} \tag{3}$$

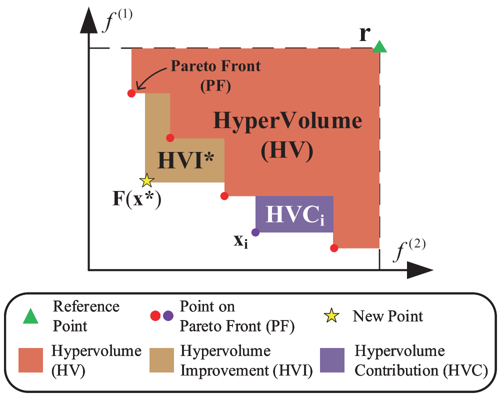

Fig. 1. Illustration of HV, HVI, and HVC.

where X denotes input space. The entire nondominated performance space is referred to as PF in

$$\mathrm{PF} = \left\{f^{(i)}(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathrm{PS},\ i = 1, \ldots, M\right\}. \tag{4}$$

Namely, all the points in the PS are (Pareto) optimal for DSE problems.

To evaluate the quality of PF in multiobjective optimization, HV is a commonly used performance indicator. As Fig. 1 illustrates, given a reference point $\boldsymbol{r}$ and a finite PF, the HV is the M-dimensional Lebesgue measure λ_M of the space dominated by PF, as defined in

$$\mathrm{HV}(PF, \boldsymbol{r}) = \lambda_M\left(\bigcup_{j=1}^{|\mathrm{PF}|} [\boldsymbol{r}, \boldsymbol{y}_j]\right) \tag{5}$$

where $[\boldsymbol{r}, \boldsymbol{y}_j]$ denotes the hyper-rectangle bounded by $\boldsymbol{r}$ and $\boldsymbol{y}_j$, with $\{\boldsymbol{y}_j\}_{j=1}^{|\mathrm{PF}|}$ comprising the PF. A larger HV corresponds to a better PF, namely a better optimization result.

To compare the importance of various points in the PS, HVC is defined as

$$\mathrm{HVC}(\boldsymbol{x}_i|\mathrm{PF}, \boldsymbol{r}) = \mathrm{HV}(\mathrm{PF}, \boldsymbol{r}) - \mathrm{HV}(\mathrm{PF} \setminus \boldsymbol{f}(\boldsymbol{x}_i), \boldsymbol{r}). \tag{6}$$

Similarly, to select the best-candidate point, HV improvement (HVI) of a candidate point $\boldsymbol{x}^*$ is defined as follows:

$$\mathrm{HVI}\big(\boldsymbol{x}^*|\mathrm{PF}, \boldsymbol{r}\big) = \mathrm{HV}\big(\mathrm{PF} \cup \boldsymbol{f}(\boldsymbol{x}^*), \boldsymbol{r}\big) - \mathrm{HV}(\mathrm{PF}, \boldsymbol{r}). \tag{7}$$

Fig. 1 is an example of a bi-objective optimization. The purple area is the HVC of $\boldsymbol{x}_i$, which is the exclusive contribution of $\boldsymbol{x}_i$ to the PF, and the yellow area is the HVI of $\boldsymbol{x}^*$, which is the improvement brought by $\boldsymbol{x}^*$ to the PF.

B. Bayesian Optimization

BO comprises two fundamental components: 1) the surrogate model and 2) the acquisition function. The surrogate model integrates existing prior knowledge and newly acquired data points to estimate a posterior distribution with predictive mean and uncertainty. Subsequently, the acquisition function is optimized with respect to the model to identify the most promising candidate for simulation, striking a balance between exploring unknown regions and exploiting local optima.

One commonly used surrogate model is Gaussian process regression (GPR) model [20]. Given a d-dimensional input

```
Algorithm 1: Multiobjective BO Framework
  Input: The size of the initial dataset $N_{init}$, and the
         maximum number of iterations $N_{iter}$
1 Generate an initial dataset $D_0 = \{X, Y\}$ by randomly
  sampling, where $Y = \{\boldsymbol{y}^{(i)} | i = 1, ..., M\}$;
2 for $t = 0 \rightarrow N_{iter} - 1$ do
3     Construct and train $M$ GPR models with $D_t$;
4     Select the next candidate point $\boldsymbol{x}_t$ by optimizing the
      acquisition function $\alpha(\boldsymbol{x}; D_t)$;
5     Simulate the candidate $\boldsymbol{x}_t$ and get the result $\boldsymbol{y}_t$;
6     Combine $\{D_t, \{\boldsymbol{x}_t, \boldsymbol{y}_t\}\}$ to form the dataset $D_{t+1}$;
7 end
  Output: Pareto set $PS$
```

vector $\boldsymbol{x}$, assume that $y = f(\boldsymbol{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ denotes the observation noise. The dataset $D = \{X, \boldsymbol{y}\}$ is formed by N observations, where X is a set of input vectors $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, and $\boldsymbol{y}$ is the corresponding outputs $\boldsymbol{y} = \{y_1, \ldots, y_N\}$.

A Gaussian process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution. A GP is fully characterized by its mean function $m(\cdot)$ and its kernel or covariance function $k(\cdot, \cdot)$. If f follows a GP, the GPR model can provide posterior distribution for an arbitrary location $\boldsymbol{x}^*$ as follow [20]:

$$\begin{cases} \mu(\boldsymbol{x}^*) &= m(\boldsymbol{x}) + k(\boldsymbol{x}^*, X)\left[K + \sigma_n^2 I\right]^{-1}(\boldsymbol{y} - m(\boldsymbol{x})) \\ \sigma^2(\boldsymbol{x}^*) &= k(\boldsymbol{x}^*, \boldsymbol{x}^*) - k(\boldsymbol{x}^*, X)\left[K + \sigma_n^2 I\right]^{-1} k(X, \boldsymbol{x}^*) \end{cases} \tag{8}$$

where $\mu(\boldsymbol{x}^*)$ is the predictive mean, $\sigma(\boldsymbol{x}^*)$ represents the uncertainty estimation, i.e., the standard deviation, $k(\boldsymbol{x}^*, X) = \{k(\boldsymbol{x}^*, \boldsymbol{x}_i) | i = 1, \ldots, N\}$, and $k(X, \boldsymbol{x}^*) = k(\boldsymbol{x}^*, X)^T$ and $K = k(X, X)$. When no prior information about $f(\boldsymbol{x})$ is available, $m(\boldsymbol{x})$ is always set to zero. In this work, we set $m(\boldsymbol{x}) = 0$ and the kernel function as the Matern52 kernel [20].

BO optimizes the acquisition function to acquire the next simulation point. Several commonly used single-objective acquisition functions include lower-confidence bound (LCB), probability of improvement (PI), and expected improvement (EI) [20]. EHVI [21] is an acquisition function that is specified for multiobjective optimization. The EHVI represents the expectation of HVI over the posterior of the GPR models, typically approximated using Monte Carlo integration [22]

$$\alpha_{\text{EHVI}}(X_{\text{cand}} | \text{PF}, \boldsymbol{r}) \approx \frac{1}{N} \sum_{t=1}^{N} \text{HVI}\big(\boldsymbol{f}_t(X_{cand}) | \text{PF}, \boldsymbol{r}\big) \tag{9}$$

where $\boldsymbol{f}_t(\cdot), t = 1, \ldots, N$ are sampled from the GPR models.

To sum up, a general multiobjective BO framework is shown in Algorithm 1.

III. Proposed Method

A. Overview of ROI-HIT Framework

Our proposed ROI-HIT framework is presented in Fig. 2. The green part represents the stage where ROI-HIT acquires ROIs in the global design space. The red part represents the local BO inside an ROI, and the blue part represents the asynchronous parallel strategy. In the global stage, a SOM tree is constructed for space partitioning. At each level of the SOM tree, a SOM network is trained on the current dataset to map the high-dimensional design space into a 2-D feature space. The SOM network ensures that neighboring points in the design space remain adjacent in the feature space. The regions associated with the PS in the feature space are defined as ROIs. By focusing on ROIs, ROI-HIT can reduce the over-exploration in the vast design space and advance the PF faster. Inside each ROI, ROI-HIT calculates a sensitivity matrix via the data points and uses it to prune the unimportant variables. Because of the reduced dimensions for modeling and optimization, the optimization time to choose candidates is shortened. In order to save the simulation time, ROI-HIT supports asynchronous parallel simulation. A new round of optimization will begin when there are idle workers, regardless of the status of the ongoing simulation. After reaching the maximum runtime budget, ROI-HIT returns the optimal PS. The combination of ROI exploitation, unimportant variable pruning, and asynchronous parallel simulation significantly enhances the exploration efficiency.

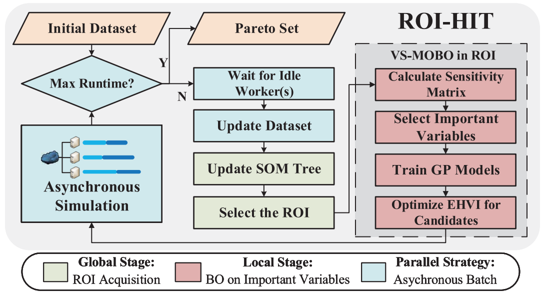
Initial Dataset
Pareto Set
ROI-HIT
Max Runtime?
Y
N
Wait for Idle Worker(s)
Update Dataset
Update SOM Tree
Select the ROI
Asynchronous Simulation
VS-MOBO in ROI
Calculate Sensitivity Matrix
Select Important Variables
Train GP Models
Optimize EHVI for Candidates
Global Stage: ROI Acquisition
Local Stage: BO on Important Variables
Parallel Strategy: Asychronous Batch

Fig. 2. Framework of ROI-HIT method.

B. SOM-Tree-Based ROI Acquisition Method

For the high-dimensional DSE, the simulation points are sparse relative to the vast design space. Besides, the commonly used GPR models have the implicit homogeneity [15], while the design space is heterogeneous. So, it is hard for the global models on simulation results to give accurate estimates of the entire space. At the same time, the space far from the simulated points has a large uncertainty, which makes the optimization method tend to over-explore. To address these problems, ROI-HIT exploits the more promising ROIs to reduce the over-exploration and advance the PF faster.

RTL netlists of RISC-V SoCs can be generated by a dedicated generator based on specific architecture parameters. So, significant variations in these parameters can lead to substantial differences in the generated netlists, resulting in considerable changes to the corresponding PPA characteristics. On the other hand, similar design parameters exhibit similar performance, as Fig. 3(a) and (b) show. Meanwhile, models can give more accurate estimates of the regions near the sampled points than the others. As a result, we designate the regions near the PS as ROIs and conduct local BO inside the ROIs to generate the candidates to be simulated.

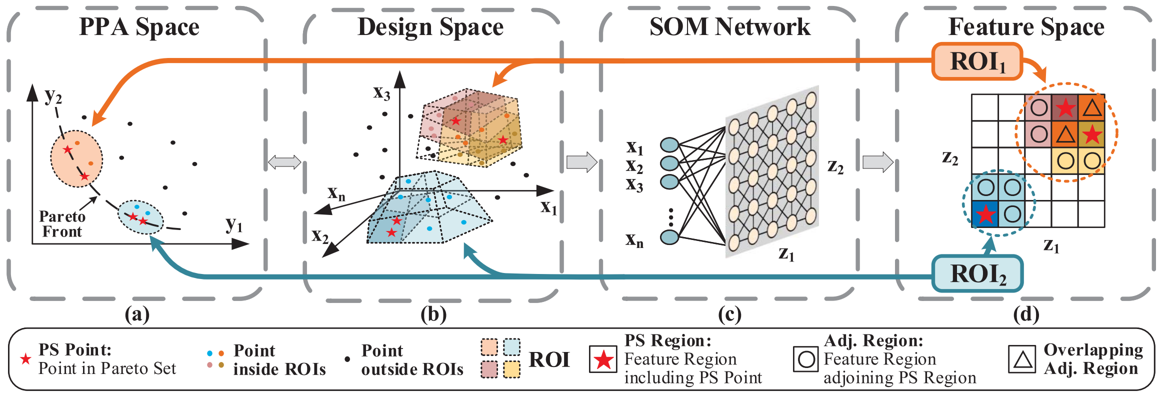

Fig. 3. Illustration of (a) PPA space, (b) high-dimensional design space, (c) SOM network, and (d) ROIs in 2-D feature space. In (a), we prioritize the regions near the current PF, shown as orange and blue regions, because they are more promising to advance the PF. These regions correspond to the orange and blue irregular design spaces in (b). To acquire the ROIs, ROI-HIT employs a SOM network, as shown in (c), to map the original design space in (b) to a 2-D feature space in (d), while the neighbors in the former are still neighbors in the latter. In (d), the feature regions, including the points in the PS (PS regions, shown as the regions with stars) and their adjoining areas (Adj. Regions, shown as the regions with circles), are designated as ROIs. If different ROIs have overlapping feature regions, shown as the regions with triangles, these ROIs will be combined into a single ROI covering all regions. By exploiting the ROIs, ROI-HIT is able to reduce the over-exploration in the vast design space and advance the PF faster.

1) SOM Network: To acquire the ROIs, a SOM network [23], [24] is used to map the high-dimensional design space into a 2-D feature space. SOM is an artificial neural network commonly utilized for high-dimensional visualization. It can ensure that neighbors in the high-dimensional space remain neighbors in the low-dimensional feature space [24], as shown in Fig. 3.

Moreover, the existing local BO methods based on trust regions [15], [16], [19] solely modify the range of the local region without considering the distribution of simulated data points. This may result in an insufficient number of simulated points in the local region, leading to inaccurate modeling of that area. Consequently, more iterations of optimization may be required to adjust the range of the trust region, potentially wasting time and computational resources. However, SOM networks consider the distribution of the dataset and are trained to have similar densities in each region [24]. So, the ROI-HIT based on SOM networks can provide a better guarantee of sufficient local dataset size and accurate modeling compared to methods based on trust regions.

After the SOM network maps the original design space to a 2-D space, we label the feature regions, including the current PS and their adjacent regions as an ROI, as presented in Fig. 3(d). If different ROIs have overlapping regions, they are merged into a single ROI covering all regions. Because the feature regions obtained by the SOM network maintain the neighbor relationships in the design space, ROI-HIT can exploit the promising local regions covering the PS.

Any new point in the high-dimensional input space can be categorized by the trained SOM network to a region in the 2-D feature space. Candidate points obtained through acquisition function optimization should be assessed by the SOM network, and only those inside the ROIs are retained for subsequent optimization.

Illustrated in Fig. 3(c), the neurons in a SOM are arranged in a 2-D lattice, with each neuron having full connections to all the source nodes in the input layer. Each neuron has a vector $\boldsymbol{w}_i$ of weights associated

$$\boldsymbol{w}_i = \{w_{ij},\ j = 1, \ldots, n\} \tag{10}$$

where n is the number of source nodes. A SOM involves the concept of a winning neuron. For a input $\boldsymbol{x}$, the winning neuron is defined as

$$i_{\text{win}} = \arg\min_i \|\boldsymbol{x} - \boldsymbol{w}_i\|. \tag{11}$$

The weight vector of the winning neuron is more similar to $\boldsymbol{x}$ than the others, and it will undergo weight adjustment to enhance its responsiveness to the input when encountered again. Meanwhile, the weights of the neurons close to the winning one need to be adjusted to produce a stronger response to $\boldsymbol{x}$ in order to preserve the topological order. So the weights of each neuron are adjusted according to the following rule:

$$\boldsymbol{w}_i = \boldsymbol{w}_i + \eta(t)h(i_{\text{win}})(\boldsymbol{x} - \boldsymbol{w}_i) \tag{12}$$

where t is the number of iterations, $\eta(t)$ is a learning rate, which decreases as t increases, and $h(i_{\text{win}})$ is a neighborhood function which has high values for i_{win} and the neurons close to i_{win} on the lattice. Usually, a Gaussian centered on i_{win} is chosen as the neighborhood function. Algorithm 2 outlines the training process of a SOM.

2) SOM Tree: To further improve optimization efficiency and obtain the optimal solutions, we propose a SOM tree. Through the SOM tree, we identify the most suitable ROI for simulation. When optimization results cease to improve, ROI-HIT will choose the most promising ROI for further subdivision and explore inside the smaller ROIs. If ROI-HIT converges to the local optimal, which means the SOM tree reaches the maximum depth, the current ROI will return to its parent node and perform a random restart. As a result, the SOM tree improves optimization efficiency in the early stage and yields a global optimization performance guarantee.

As shown in Fig. 4(a), each node in the SOM tree represents an ROI. As the overlapped ROIs should be merged into the same ROI, the ROIs are independent of each other. At each level, all ROIs are acquired by the same SOM network, which is trained during the construction of nodes at that level. The size of this SOM network is determined by the point number and dimension of the dataset. In this work, the initial dataset size is 100, and dimensionality reduction yields a design space

Algorithm 2: Train SOM Network

```
Input: The dataset $X$, the maximum iterations $N_{iter}$
Initialize the weights of each neuron at random;
for $t = 0 \rightarrow N_{iter} - 1$ do
    Randomly choose an input $\boldsymbol{x} \in X$;
    Determine the winning neuron by (11);
    Adapt the weights of each neuron by (12);
end
```

of around 10 dimensions. Since an ROI occupies at least 4 neighboring feature regions mapped by the SOM network, and each feature region contains a similar number of data points, we set the SOM network size as 5×5. This ensures that each ROI contains at least 16 data points, facilitating subsequent BO [25].

At each level, we select the ROI for optimization based on its figure of merit (FOM)

$$\text{FOM} = N_{\text{Succ}} - N_{\text{Fail}} + \text{HVC}/\text{HVC}_{\max} \tag{13}$$

where N_{Succ} is the number of times that the ROI finds a new Pareto point and N_{Fail} is the number of times that the ROI fails to find a new Pareto point. HVC is the largest HVC of the Pareto points inside the ROI, $\text{HVC}_{\max}$ is the largest HVC among all Pareto points. The SOM tree updates N_{Succ} and N_{Fail} as follows:

$$\begin{cases} N_{\text{Succ}} \mathrel{+}= 1,\ N_{\text{Fail}} = 0, \text{ if ROI gets a new PF} \\ N_{\text{Succ}} = 0,\ N_{\text{Fail}} \mathrel{+}= 1, \text{ otherwise.} \end{cases} \tag{14}$$

By the FOM, ROI-HIT favors exploring regions that are more likely to improve the PF.

If an ROI fails to find a new Pareto point for $N_{\text{Fail}_{\max}}$ consecutive times, namely, $N_{\text{Fail}} \geq N_{\text{Fail}_{\max}}$, we define that the ROI is *well explored*. As Fig. 4(b) shows, when all ROIs at the current level are well explored, we will split the best ROI, namely, the ROI with $\text{HVC}_{\max}$. A new SOM network will be trained inside this ROI, and new ROIs will be acquired. These ROIs will be child nodes of the original ROI. In our approach, the new SOM network has the same size as the original one. To ensure sufficient data points in the new ROI, the split is only performed when the number of data points in the ROI exceeds the number of data points in the initial dataset; otherwise, ROI-HIT continues to select ROIs based on the FOM value.

When the range of ROIs has become sufficiently small, i.e., the depth of the SOM tree has reached $Depth_{\max}$, and all ROIs are well explored, we consider these ROIs to have converged. As is shown in Fig. 4(c), ROI-HIT will return to the parent node and perform N_{RS} rounds of random search to explore unexplored regions. If new Pareto points are found, the ROIs will be updated and a new optimization iteration will begin. If no new Pareto points are found, the search will proceed to an upper-level node until a random search is performed globally.

C. Variable Selection Based on Sensitivity Matrix

For high-dimensional DSE, the time complexity for modeling and optimization increases linearly or faster with the

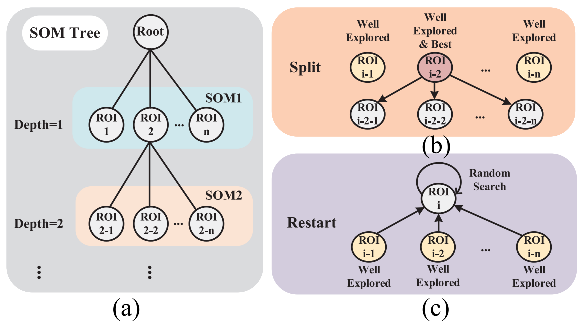

Fig. 4. Illustration of SOM Tree. (a) Each node represents an ROI. Since the number of ROIs obtained from ROI-HIT is not fixed, the number of child nodes at each level varies. At the same level, all ROIs are acquired by the same SOM network. The next ROI to be optimized is selected based on its FOM value. (b) If all the ROIs at the current level are well explored, the best ROI will be split. A new SOM network will be trained inside the split ROI, and the smaller ROIs will be acquired via the new SOM for more detailed exploration. (c) If the ROIs are small enough and well explored, ROI-HIT will restart, which means the current ROI returns to its parent node and conducts a random search.

number of dimensions. The existing methods [17], [18], [19] use embeddings to map the original high-dimensional space to a low-dimensional intrinsic space, thus reducing the dimensionality. However, these methods require specifying the effective dimension number in advance, which can be challenging since determining the accurate intrinsic dimension of the problem is often difficult.

Meanwhile, different microarchitecture design parameters have distinct influences on the PPA results. For instance, a modification in *DecodeWidth* significantly affects the PPA results, whereas the Boolean value of *EnableSFBOpt* makes a minor impact. However, the embedding-based methods ignore the different importance of design parameters on performances. Moreover, the relative importance of different design parameters varies across applications. Even seasoned design experts face challenges in accurately identifying the key parameters for a specific scenario due to the increasing complexity of RISC-V SoCs and the opaque optimization methods employed by EDA tools.

Our proposed ROI-HIT obtains a sensitivity matrix to evaluate the importance of different variables according to the input variables and their performances. Unlike existing methods that require specifying the effective dimension number, ROI-HIT eliminates unimportant variables and automatically adjusts the number of dimensions to reduce optimization time.

A sensitivity matrix S [26] is calculated to measure the importance of variables

$$S = \left\{ s_{ij} \mid i = 1, \ldots, n,\ j = 1, \ldots, M \right\} \tag{15}$$

where n denotes the number of variables, namely, the dimension d of input vector $\boldsymbol{x}$, and M is the number of objectives. The sensitivity coefficient s_{ij} is defined as

$$s_{ij} = \frac{\text{Var}\big[E\big(y_j|x_i\big)\big]}{\text{Var}\big(y_j\big)} \approx \frac{\sum_{l=1}^{r_i} |A_i^l| \left[\left(\bar{y}_j\right)_i^l - \bar{y}_j \right]^2}{\sum_{\boldsymbol{x} \in X} \left[y_j(\boldsymbol{x}) - \bar{y}_j \right]^2}$$

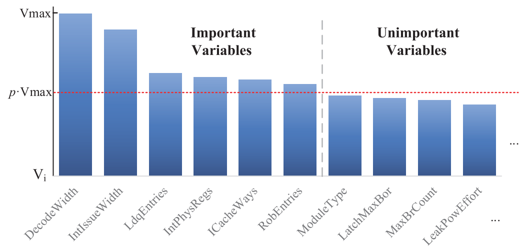

Fig. 5. Illustration of the VS based on the sensitivity matrix for RISV-V BOOM microarchitecture [6]. The maximum importance value ($V_{\max}$) represents the peak importance, and the threshold for VS is determined by the product of p and $V_{\max}$.

$$\bar{y}_j = \frac{1}{|X|}\sum_{\boldsymbol{x}\in X} y_j(\boldsymbol{x}), \quad (\bar{y}_j)_i^l = \frac{1}{|A_i^l|}\sum_{\boldsymbol{x}\in A_i^l} y_j(\boldsymbol{x}) \tag{16}$$

where X represents the dataset, and r_i represents the number of value which x_i can be and A_i^l represents the set of points for which the variable x_i is set to its lth value.

By definition, s_{ij} is a value in the range of 0 to 1. The value closer to 1 indicates that variable x_i has a more significant effect on objective y_j, whereas the value closer to 0 indicates that variable x_i has a less significant effect on variable y_j.

For multiobjective optimization, the importance of different objectives should be integrated. In order to obtain a better PF, we hope to improve the shortcomings of the current optimal results. So, the weight to the jth objective w_j and the importance value V_i of the ith variable are defined as

$$w_j = \frac{y_{j_{\text{best}}} - y_{j_{\min}}}{y_{j_{\text{avg}}}}, \quad j = 1, \ldots, M \tag{17}$$

$$V_i = \sum_{j=1}^{m}\left(w_j \cdot s_{ij}\right) + C_p\sqrt{1/n_i} \tag{18}$$

where $y_{j_{\text{best}}}$ is the jth objective value of the Pareto point with the largest HVC inside the ROI, $y_{j_{\min}}$ is the minimum jth objective value among all points, $y_{j_{\text{avg}}}$ is the average jth objective value of the whole dataset, n_i is the number that the variable x_i has been selected as the important variable, and C_p is a hyper-parameter to make a tradeoff between exploitation and exploration. For microarchitecture DSE, C_p is set as 0.01.

A variable is considered important if its importance value V_i surpasses the threshold V_{th} which is determined by the product of a hyper-parameter p and the maximum importance value observed $V_{\max}$, as shown in (19) and Fig. 5

$$D_{ipt} = \{i \mid i \in D,\ V_i \geq V_{th}\}, \quad V_{th} = p \cdot V_{\max} \tag{19}$$

where $D = \{i | i = 1, \ldots, n\}$. Moreover, if the number of important variables is less than a lower bound N_{ipt_L}, the largest N_{ipt_L} variables are selected as important variables to ensure the effectiveness of modeling. Conversely, if the number of important variables exceeds an upper bound N_{ipt_U}, the N_{ipt_U} variables with the highest-importance values are chosen as important variables to avoid excessively long optimization time. In this work, the number of important variables is constrained to be between 3 and 20. According to the theoretical analysis in [27], the greater the importance of the unselected variables, the higher the cumulative regret. Therefore, to enhance optimization efficiency, p in (19) is set to 0.5, thereby limiting the importance of the unselected variables.

Besides, unimportant variables are fixed to the values of the Pareto points inside the current ROI. As there may be more than one Pareto point, unimportant variables are transformed into a discrete auxiliary variable var_{aux}, which is the index of the closest Pareto point

$$var_{aux} = \arg\min_k \|\boldsymbol{x} - \boldsymbol{x}_{PS_k}\|. \tag{20}$$

We utilize both the important variables and the auxiliary variable to construct models for subsequent optimization.

D. Asynchronous Batch Multiobjective BO

To yield precise PPA results for RISC-V processors, a VLSI flow is employed. However, a single VLSI flow can take 1-6 h to complete. Since the simulations are independent, simulations can be run in parallel. However, various design parameters result in different simulation time, so the synchronous parallel/batch method must wait for the slowest simulation in a batch to complete before starting a new iteration. Consequently, an asynchronous batch strategy is employed to reduce simulation time and further enhance exploration efficiency.

The asynchronous parallelism means that a new optimization iteration starts before all the simulation points are returned. Thus, it is essential to carefully select candidate points to avoid repeated simulations. The EHVI [22] acquisition function is utilized to choose the candidate points.

When selecting candidates, if the simulation of $\boldsymbol{x}_i$ does not return, $\boldsymbol{x}_i$ will be placed into the pending point set to create a pseudo PS. The GPR models are left unchanged. At this time, the EHVI value of $\boldsymbol{x}_i$ will become 0 according to the definition of EHVI in (9). Then, the next candidate $\boldsymbol{x}_{i+1}$ will be obtained by the EHVI acquisition function. So, $\boldsymbol{x}_{i+1}$ is distinct from existing points.

Since the SOM network is unable to perform a reverse transformation from the feature space to the design space, hill-climbing local search [18] is employed for acquisition function optimization. When it comes to the auxiliary dimension, the points in the PS are chosen as the corresponding discrete input values to guarantee that the candidate points correspond to the points in the original high-dimensional design space.

IV. Experimental Results

A. Experimental Setup

We explore the design spaces for a RISC-V BOOM [6] core and a Rocket core [7]. Both of them are RV64GC configurations. To assess the PPA results, all circuits in the experiments are SoCs featuring either the BOOM core or Rocket core as their processor cores. The peripheral modules of these SoCs are standard components provided by the Chipyard [3] framework. A 7nm ASAP7 PDK [28] is utilized in the VLSI flow. Since the designs based on ASAP7 are not manufacturable, this article only achieves the front-end design in the experiments. The RTL designs are generated by the Chipyard framework, and the designs are simulated

TABLE I
DESIGN PARAMETERS

Name	Value	Name	Value
BOOM (22)		Genus (31)	
BPD	Boom2,TAGEL,Alpha21264	time_recovery_arcs	false,true
fetchWidth	4,8	auto_partition	true,false
numFetchBufferEntries	8,16,24,32,35,40	dp_analytical_opt	off,standard,extreme
numRasEntries	16,24,32	dp_area_mode	false,true
maxBrCount	8,12,16,20	dp_csa	basic,none
decodeWidth	1,2,3,4,5	dp_rewriting	basic,advanced,none
numRobEntries	32,64,96,128,130	dp_sharing	advanced,basic,none
numIntPhysRegisters	48,64,80,96,112	exact_match_seq_async_ctrls	false,true
memIssueWidth	1,2	exact_match_seq_sync_ctrls	false,true
intIssueWidth	1,2,3,4,5	iopt_enable_floating_output_check	false,true
numLdqEntries	8,16,24,32	iopt_force_constant_removal	false,true
enablePrefetching	false,true	iopt_lp_power_analysis_effort	low,medium,high
enableSFBOpt	false,true	lbr_seq_in_out_phase_opto	false,true
numRXQEntries	2,4,8	optimize_net_area	true,false
numRCQEntries	8,16,32	retime_effort_level	low,medium,high
nL2TLBEntries	128,256,512,1024	retime_optimize_reset	false,true
nL2TLBWays	1,2,4,8	syn_generic_effort	low,medium,high
nICacheWays	2,4,8	syn_map_effort	low,medium,high
nICacheTLBWays	8,16,32	syn_opt_effort	low,medium,high
nDCacheWays	2,4,8	leakage_power_effort	low,medium,high
nDCacheMSHRs	2,4,8	lp_power_analysis_effort	low,medium,high
nDCacheTLBWays	8,16,32	enable_latch_max_borrow	false,true
Rocket (9)		latch_max_borrow	1:500:1
mulUnroll	1,2,4,8	enable_max_fanout	false,true
divUnroll	1,2,4,8	max_fanout	2:64:1
mulEarlyOut	false,true	enable_lp_clock_gating_max_flops	false,true
divEarlyOut	false,true	lp_clock_gating_max_flops	8,16,32
nDCacheWays	1,2,4	enable_lp_power_optimization_weight	false,true
nDCacheTLBWays	4,8,16	lp_power_optimization_weight	0.1:0.9:0.1
nDCacheMSHRs	1,2,4	max_dynamic_power	3.0:150.0:0.1
nICacheWays	1,2,4	max_leakage_power	5.0:100.0:0.1
nICacheTLBWays	4,8,16		

via an open-source simulator Verilator [10] to get CPI as the performance metric. Cadence Genus Synthesis Solution 19.10 is employed for logic synthesis. We set the clock frequency to 250 MHz for low-power embedded system applications. The area and power are obtained by Genus. The power metric is the total power. In this article, we refer to the time elapsed from inputting design parameters to obtaining PPA results as simulation time; the time spent running the algorithm is referred to as optimization time. The sum of these two time constitutes the total time.

To evaluate the performance of the designs, 13 benchmarks are selected, including *dhrystone, median, mm, mt-matmul, mt-vvadd, multiply, qsort, rsort, spmv, towers, vvadd, fir2dim*, and *whetstone*. The first 11 benchmarks are from riscv-test,[1] a benchmark suite by RISC-V International designed to evaluate RISC-V processor performance. These benchmarks can well cover the RISC-V instructions and are employed in BOOM-Explorer [13] and GRL-DSE [14]. BOOM-Explorer also utilizes *fir2dim*[2] and *whetstone*[3] to evaluate digital signal processing (DSP) and floating-point performance, respectively. So, we adopt them for assessing our processor's capabilities. The optimizations are to minimize the average CPI of benchmarks, power consumption, and area of the generated design. All experiments take place on a Linux workstation equipped with 80 Intel Xeon Gold 6230 CPUs @2.10 GHz.

As shown in Table I, the design parameters encompass the 22 parameters of BOOM, the nine parameters of Rocket, and the 31 parameters of Genus, all represented as discrete values. Namely, the dimension numbers of design spaces for BOOM and Rocket are 53 and 40, respectively. The sizes of the design space for BOOM and Rocket are around 6.96×10^{32} and 4.06×10^{25}, respectively.

We compare our ROI-HIT method with five state-of-the-art methods: 1) BOOM-Explorer [13], a sequential BO method for microarchitecture DSE based on an efficient initial sampling method MicroAL; 2) GRL-DSE [14], another sequential BO method for microarchitecture DSE that utilizes a GNN to map the design space into graph embedding space; 3) BODi [18], a BO method for high-dimensional discrete optimization problem that employs dictionary embedding; 4) MORBO [16], an efficient multiobjective BO method that is based on trust regions; and 5) REMOTune [19], a recent BO method for high-dimensional DSE that combines the trust regions and random embedding. Since the MORBO and REMOTune support the parallel simulation in a batch, these two methods are compared in both sequential mode and batch mode. For clarity, we label the batch size after the name of the algorithms. In these experiments, the batch size is set as 4. All algorithms are implemented in Python.

[1]https://github.com/riscv-software-src/riscv-tests

[2]https://www.ice.rwth-aachen.de/research/tools-projects/closed-projects/dspstone

[3]https://netlib.org/benchmark/whetstone.c

TABLE II
ABLATION STUDY OF ROI-HIT BASED ON RISC-V BOOM MICROARCHITECTURE DSE

Algorithm	↑ HV ($\times10^5$)	Impro.	Opt. Time	Speed up	# Sim	Multi.
Vanilla BO	1.7240	0.00%	23493s	1.00×	13	1.00×
BO+ROI	1.7291	0.30%	22990s	1.02×	13	1.00×
BO+VS	1.7565	1.89%	**324s**	**72.51×**	15	1.15×
ROI-HIT-1	1.7887	3.76%	876s	26.83×	15	1.15×
ROI-HIT-4S	1.8015	4.50%	2880s	8.16×	44	3.38×
ROI-HIT-4	**1.8379**	**6.61%**	/	/	**53**	**4.08×**

Due to the untraversable high-dimensional design space, BOOM-Explorer and GRL-DSE randomly select 15 000 sample points from the entire design space to train MicroAL and GNN. Additionally, we extend the original BODi from single-objective optimization to multiobjective optimization by using the EHVI acquisition function [22]. In order to gain an insight into the high-dimensional design space, we set the initial dataset size to 100. Since the points in the initial dataset are independent, they can be simulated in parallel. Consequently, through a 10-process parallel execution, the initial dataset can be acquired in one day. For a fair comparison, all algorithms are initiated with an identical set of designs sampled from the design space utilizing the MicroAL algorithm in [13]. The maximum time budget for the experiments is one day (86400s). The reference point to calculate the HV is set as the maximum CPI, power, and area values in the initial dataset. The experiments are conducted five times, and the average results are reported.

B. Ablation Study

To verify the effectiveness of the ROI acquisition method and VS, we first conduct ablation experiments of our ROI-HIT method on the RISC-V BOOM microarchitecture DSE problem. We conduct a comparison between a vanilla BO, a BO method with ROI (BO+ROI), a BO method based on VS (BO+VS), a sequential ROI-HIT-1, a synchronous batch ROI-HIT-4S, and an asynchronous batch ROI-HIT-4. The experimental results are presented in Table II.

1) Effectiveness of ROI and VS: ROIs offer a more promising local scope, while VS focuses on important variables, such as *DecodeWidth* and *IntIssueWidth*, and mitigates interference

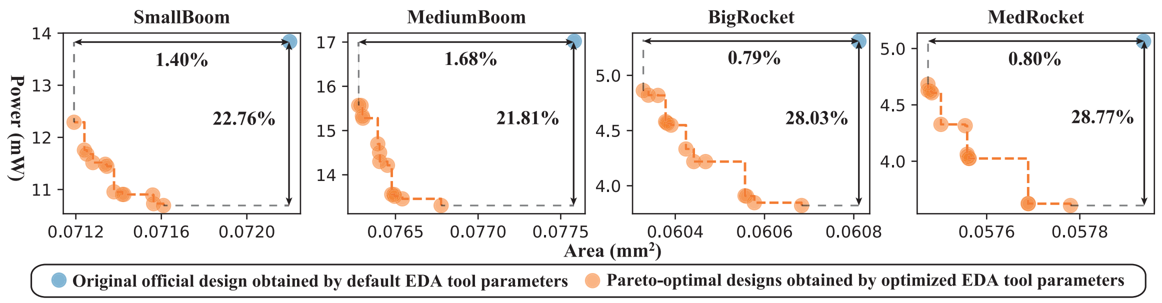

Fig. 6. Optimization results obtained by adjusting the EDA tool parameters while keeping the architecture parameters unchanged. The blue points denote the original official designs (*SmallBoom* and *MediumBoom* are from [6]. *BigRocket* and *MedRocket* are from [7].) The orange points denote the Pareto-optimal designs from optimizing the EDA tool (Genus) parameters by ROI-HIT-4. The original design and the optimized designs have the same architecture parameters, so their RTL netlists and CPIs are the same.

TABLE III
OPTIMIZATION RESULTS AND CORRESPONDING TIME FOR RISC-V BOOM MICROARCHITECTURE DSE

Algorithm	BOOM-Explorer [13]	GRL-DSE [14]	BODi [18]	MORBO-1 [16]	REMOTune-1 [19]	**ROI-HIT-1**	MORBO-4 [16]	REMOTune-4 [19]	**ROI-HIT-4**
↑ HV	172391	172540	172871	172781	172428	**178872**	175585	172551	**183795**
Improvement	0.00%	0.09%	0.28%	0.23%	0.02%	**3.76%**	1.85%	0.09%	**6.61%**
↓ $\widehat{\text{ADRS}}$	35280	55300	47351	54191	50092	**29569**	37934	56007	**21538**
Improvement	0.00%	-56.75%	-34.21%	-53.60%	-41.98%	**16.19%**	-7.52%	-58.75%	**38.95%**
↓ Min. CPI	1.3919	0.9080	0.9312	**0.7924**	1.0531	0.8946	0.8159	0.9501	**0.7820**
Improvement	0.00%	34.77%	33.10%	**43.07%**	24.34%	35.73%	41.38%	31.74%	**43.82%**
↓ Min. Power	10.763mW	13.440mW	10.048mW	10.170mW	11.945mW	**9.102mW**	9.341mW	12.274mW	**8.978mW**
Improvement	0.00%	-24.88%	6.64%	5.51%	-10.98%	**15.43%**	13.21%	-14.04%	**16.58%**
↓ Min. Area	0.0688mm^2	0.0772mm^2	0.0710mm^2	0.0709mm^2	0.0734mm^2	**0.0688mm^2**	0.0692mm^2	0.0743mm^2	**0.0684mm^2**
Improvement	0.00%	-12.16%	-3.20%	-2.93%	-6.59%	**0.08%**	-0.58%	-7.97%	**0.65%**
Opt. Time	33065s	4291s	10456s	1175s	1227s	**876s**	1416s	1152s	/
Speed up	1.00×	7.71×	3.16×	28.13×	26.94×	**37.76**×	23.35×	28.70×	/
# Sim	11	8	11	14	13	**15**	52	42	**53**
Multiple	1.00×	0.73×	1.00×	1.27×	1.18×	**1.36**×	4.73×	3.82×	**4.82**×

from noncritical dimensions during ROI-driven optimization. ROIs and VS improve, respectively, 0.30% and 1.89% HV compared with vanilla BO. The combination of ROIs and VS thus greatly enhances the results of ROI-HIT. The sequential ROI-HIT-1 achieves 3.76%, 3.45%, and 1.83% improvement in HV compared with vanilla BO, BO with ROI, and BO with VS.

2) Effectiveness of Batch Simulation: The synchronous and asynchronous batch simulation methods, through increased exploration points, further enhance the exploration results within the limited time budget. The asynchronous ROI-HIT-4 with a batch of 4 achieves, respectively, 6.61%, 2.75%, and 2.02% improvement in HV compared with vanilla BO, sequential ROI-HIT-1, and synchronous ROI-HIT-4S. By exploiting the idle time gap between different simulations in a batch, asynchronous ROI-HIT-4 explores 1.20× more points than synchronous ROI-HIT-4S.

3) Optimization Time: The VS notably reduces the optimization time by significantly reducing the dimensions and obtains a speedup of 72.51× in optimization time compared to vanilla BO. Benefiting from the VS, the sequential ROI-HIT-1 achieves optimization time speedups of 26.83× compared to vanilla BO. However, due to asynchronous batch simulation, where simulation time overlaps with optimization time, it is not feasible to quantify the optimization time for asynchronous ROI-HIT-4.

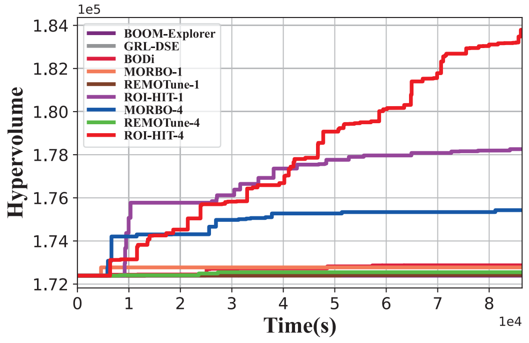

Fig. 7. Optimization results for RISC-V BOOM microarchitecture DSE.

4) Effect of EDA Tool Parameters: Although the CPI of the design is determined by the RTL netlist generated by the architecture parameters, differing EDA tool configurations result in variations in power and area. This is illustrated in Fig. 6, where the blue points represent the power and area resulting from logic synthesis using default parameters, while the orange points represent the PF achieved by utilizing ROI-HIT-4 to optimize EDA tool parameters. The experiment suggests that optimizing EDA tool parameters achieves 21.81%–28.77% improvement in power and 0.79%–1.68% improvement in area.

C. RISC-V BOOM Microarchitecture DSE

The optimization results are shown in Table III and Fig. 7. *Min. CPI, Min. Power,* and *Min. Area* in the table denote

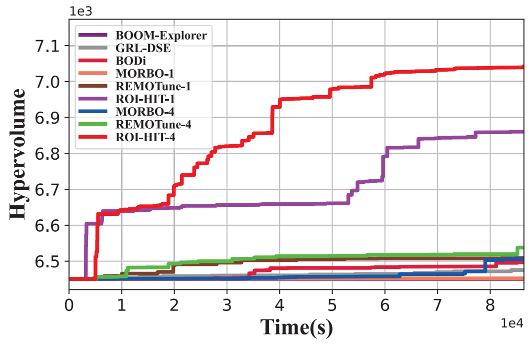

Fig. 8. Optimization results for rocket microarchitecture DSE.

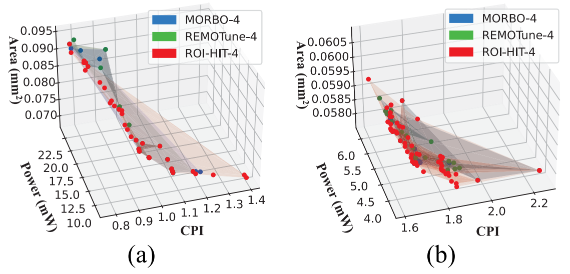

Fig. 9. PFs obtained by, respectively, MORBO-4 [16], REMOTune-4 [19], and our proposed ROI-HIT-4. (a) BOOM. (b) Rocket.

the minimum values of CPI, power, and area achieved by the optimization. In addition to HV, we also use average distance to reference set (ADRS) to provide a comprehensive evaluation of optimization results

$$\mathrm{ADRS}(\Gamma, \Omega) = \frac{1}{|\Gamma|} \sum_{\gamma \in \Gamma} \min_{\omega \in \Omega} d(\gamma, \omega) \tag{21}$$

where d is the Euclidean distance function. Γ is the real PF and Ω is the optimized PF. However, due to the vastness of our design space, it is not feasible to obtain a real PF. To address this issue, we consider all the points evaluated in the experiments as the total dataset, utilizing the PF derived from it as the reference set to calculate the $\widehat{\mathrm{ADRS}}$. Compared with the state-of-the-art methods, ROI-HIT obtains the largest HV and the least $\widehat{\mathrm{ADRS}}$ in both sequential and batch mode. The sequential ROI-HIT-1 achieves 3.47%–3.76% improvement in HV compared with sequential baselines, while asynchronous batch ROI-HIT-4 achieves 4.67% and 6.51% improvement in HV compared with MORBO-4 and REMOTune-4. ROI-HIT-1 improves $\widehat{\mathrm{ADRS}}$ by 16.19%–46.53% over sequential baselines, while ROI-HIT-4 improves $\widehat{\mathrm{ADRS}}$ by 43.22% and 61.54% over MORBO-4 and REMOTune-4. In terms of PPA metrics, ROI-HIT-4 gains improvements of 4.16%–43.82% in CPI, 3.88%–33.20% in power, and 0.65%–11.41% in area, compared to other methods. Besides, ROI-HIT-1 takes the least optimization time among all methods, and ROI-HIT-4 explores the most points within the limited time budget. As shown in Fig. 7, compared to state-of-the-art methods, ROI-HIT-4 achieves a speed-up of 3.58×–13.52× in reaching the maximum HV that the baselines achieve. Fig. 9(a) shows the PFs obtained from a sole run of MORBO-4, REMOTune-4, and ROI-HIT-4. ROI-HIT-4 gains the best PF compared to MORBO-4 and REMOTune-4.

To verify the effectiveness of the optimization, we compare the Pareto-optimal designs obtained from the experiment with the official BOOM designs (*Small BOOM* and *Medium BOOM*) from [6]. As Table IV shows, ROI-HIT-4 achieves improvements of 19.21% in CPI, 25.52% in power, and 1.61% in area compared to Small BOOM, while ROI-HIT-4 achieves improvements of 0.30% in CPI, 30.05% in power, and 2.71% in area compared to Medium BOOM. Through the VS, we identify the eight most important design parameters: 1) decodeWidth$^{\mathrm{B}}$; 2) BPD$^{\mathrm{B}}$; 3) intIssueWidth$^{\mathrm{B}}$; 4) fetchWidth$^{\mathrm{B}}$; 5) numIntPhysRegisters$^{\mathrm{B}}$; 6) memIssueWidth$^{\mathrm{B}}$; 7) numRasEntries$^{\mathrm{B}}$; and 8) leakage_power_effort$^{\mathrm{G}}$. Parameters with a superscript B belong to BOOM parameters, while the parameter with a superscript G belongs to Genus parameters. Increasing the decodeWidth$^{\mathrm{B}}$ can significantly improve CPI but at the cost of higher-power consumption and area. Similarly, the choice of prediction branch (BPD$^{\mathrm{B}}$) affects CPI, power consumption, and area. In terms of EDA tool parameters, logic synthesis with a higher-leakage_power_effort$^{\mathrm{G}}$ yields significantly better-power optimization results compared to the official design at the default setting.

D. Rocket Microarchitecture DSE

The optimization results are shown in Table V and Fig. 8. Compared with the state-of-the-art methods, ROI-HIT obtains the largest HV and the least $\widehat{\mathrm{ADRS}}$ in both sequential and batch mode. The ROI-HIT-1 improves HV by 5.49%–6.54% over sequential baselines, while ROI-HIT-4 achieves HVIs of 8.24% and 7.72% compared to MORBO-4 and REMOTune-4. ROI-HIT-1 improves $\widehat{\mathrm{ADRS}}$ by 39.11%–46.97% over sequential baselines, while ROI-HIT-4 improves $\widehat{\mathrm{ADRS}}$ by 63.95% and 59.53% over MORBO-4 and REMOTune-4. In terms of PPA metrics, ROI-HIT-4 achieves improvements of 0.00%–16.27% in CPI, 3.87%–22.72% in power, and 0.16%–0.57% in area, compared to other methods. Besides, ROI-HIT-1 takes the least optimization time among all methods, and ROI-HIT-4 explores the most points within the limited time budget. As shown in Fig. 8, compared to state-of-the-art methods, ROI-HIT-4 achieves a speed-up of 16.13×–17.23× in reaching the maximum HV that the baselines achieve. Fig. 9(b) shows the PFs obtained from a sole run of MORBO-4, REMOTune-4, and ROI-HIT-4. ROI-HIT-4 gains the best PF compared to MORBO-4 and REMOTune-4.

We also compare the Pareto-optimal designs obtained from the experiment with the official Rocket designs (*BigCore* and *MedCore*) from [7]. As Table VI shows, ROI-HIT-4 improves CPI by 0.32%, power by 18.13%, and area by 0.81% compared with BigCore, while ROI-HIT-4 improves CPI by 18.04%, power by 25.32%, and area by 0.01% compared with MedCore. Through the VS, we identify the eight most important design parameters: 1) nDCacheWays$^{\mathrm{R}}$; 2) divUnroll$^{\mathrm{R}}$; 3) nICacheWays$^{\mathrm{R}}$; 4) mulUnroll$^{\mathrm{R}}$; 5) syn_opt_effort$^{\mathrm{G}}$; 6) leakage_power_effort$^{\mathrm{G}}$; 7) nICacheTLBWays$^{\mathrm{R}}$; and 8) dp_analytical_opt$^{\mathrm{G}}$. Parameters with a superscript R belong to Rocket parameters, while the parameters with a superscript

TABLE IV
PARETO-OPTIMAL DESIGNS FOR RISC-V BOOM MICROARCHITECTURE

Design	Important Variables*	↓ CPI	Improvement	↓ Power	Improvement	↓ Area	Improvement
Small BOOM [6]	[0, 1, 0, 0, 0, 0, 2, 0]	1.3725	0.00%	13.844mW	0.00%	0.0722mm^2	0.00%
MORBO-4 [16]	[1, 1, 1, 1, 1, 1, 1, 1]	**1.0719**	**21.90%**	11.587mW	16.30%	0.0735mm^2	-1.74%
REMOTune-4 [19]	[1, 0, 1, 0, 1, 0, 1, 2]	1.0792	21.37%	10.656mW	23.03%	0.0718mm^2	0.49%
ROI-HIT-4	**[1, 0, 1, 0, 1, 0, 0, 2]**	1.1088	19.21%	**10.311mW**	**25.52%**	**0.0710mm^2**	**1.61%**
Medium BOOM [6]	[1, 1, 1, 0, 2, 0, 2, 0]	0.9906	0.00%	17.018mW	0.00%	0.0776mm^2	0.00%
MORBO-4 [16]	[1, 1, 1, 0, 1, 0, 0, 2]	0.9575	3.35%	13.579mW	20.21%	0.0772mm^2	0.44%
REMOTune-4 [19]	[2, 1, 2, 0, 1, 0, 0, 2]	**0.9488**	**4.23%**	15.374mW	9.66%	0.0790mm^2	-1.85%
ROI-HIT-4	**[1, 0, 2, 1, 2, 0, 0, 2]**	0.9877	0.30%	**11.905mW**	**30.05%**	**0.0755mm^2**	**2.71%**

* Important Variables are the 8 most important design parameters obtained by variable selection. They are decodeWidth^B, BPD^B, intIssueWidth^B, fetchWidth^B, numIntPhysRegisters^B, memIssueWidth^B, numRasEntries^B, and leakage_power_effort^G. Parameters with a superscript B belong to BOOM parameters, while the parameter with a superscript G belongs to Genus parameters.

TABLE V
OPTIMIZATION RESULTS AND CORRESPONDING TIME FOR ROCKET MICROARCHITECTURE DSE

Algorithm	BOOM-Explorer [13]	GRL-DSE [14]	BODi [18]	MORBO-1 [16]	REMOTune-1 [19]	ROI-HIT-1	MORBO-4 [16]	REMOTune-4 [19]	ROI-HIT-4
↑ HV	6451	6475	6497	6452	6515	**6872**	6507	6539	**7043**
Improvement	0.00%	0.38%	0.72%	0.03%	1.00%	**6.54%**	0.87%	1.37%	**9.19%**
↓ $\widehat{\text{ADRS}}$	2457	2140	2299	2449	2146	**1303**	2172	1935	**783**
Improvement	0.00%	12.90%	6.43%	0.33%	12.66%	**46.97%**	11.60%	21.25%	**68.13%**
↓ Min. CPI	1.9192	1.6104	1.6121	**1.6070**	1.6828	1.6163	**1.6070**	1.6768	**1.6070**
Improvement	0.00%	16.09%	16.00%	**16.27%**	12.32%	15.78%	**16.27%**	12.63%	**16.27%**
↓ Min. Power	4.756mW	3.935mW	3.919mW	3.823mW	3.877mW	**3.735mW**	3.823mW	4.108mW	**3.675mW**
Improvement	0.00%	17.26%	17.60%	19.61%	18.48%	**21.46%**	19.61%	13.63%	**22.72%**
↓ Min. Area	0.0579mm^2	0.0580mm^2	0.0580mm^2	0.0578mm^2	0.0580mm^2	**0.0577mm^2**	0.0578mm^2	0.0578mm^2	**0.0577mm^2**
Improvement	0.00%	-0.25%	-0.11%	0.12%	-0.26%	**0.32%**	0.16%	0.07%	**0.32%**
Opt. Time	50062s	14495s	20977s	2424s	3508s	**435s**	3717s	3064s	/
Speed up	1.00×	3.45×	2.39×	20.65×	14.27×	**115.02×**	13.47×	16.34×	/
# Sim	16	22	21	27	**29**	28	96	90	**98**
Multiple	1.00×	1.38×	1.31×	1.69×	**1.81×**	1.75×	6.00×	5.63×	**6.13×**

TABLE VI
PARETO-OPTIMAL DESIGNS FOR ROCKET MICROARCHITECTURE

Design	Important Variables*	↓ CPI	Improvement	↓ Power	Improvement	↓ Area	Improvement
BigCore [7]	[2, 0, 0, 3, 0, 0, 2, 0]	1.6121	0.00%	5.309mW	0.00%	0.0608mm^2	0.00%
MORBO-4 [16]	[2, 1, 0, 2, 2, 1, 2, 2]	1.6239	-0.73%	4.914mW	7.45%	0.0594mm^2	2.32%
REMOTune-4 [19]	[1, 0, 1, 3, 1, 0, 2, 0]	1.6919	-4.95%	4.530mW	14.67%	**0.0586mm^2**	**3.57%**
ROI-HIT-4	**[2, 1, 1, 2, 2, 2, 0, 0]**	**1.6070**	**0.32%**	**4.347mW**	**18.13%**	0.0603mm^2	0.81%
MedCore [7]	[0, 0, 0, 0, 0, 0, 0, 0]	2.2670	0.00%	5.064mW	0.00%	0.0579mm^2	0.00%
MORBO-4 [16]	[0, 1, 0, 2, 1, 2, 0, 0]	1.9615	13.48%	3.823mW	24.51%	**0.0579mm^2**	**0.13%**
REMOTune-4 [19]	[0, 1, 0, 2, 2, 2, 0, 1]	1.8823	16.97%	3.870mW	23.59%	0.0581mm^2	-0.28%
ROI-HIT-4	**[0, 0, 0, 3, 1, 2, 1, 1]**	**1.8581**	**18.04%**	**3.782mW**	**25.32%**	0.0579mm^2	0.01%

* Important Variables are the 8 most important design parameters obtained by variable selection. They are nDCacheWays^R, divUnroll^R, nICacheWays^R, mulUnroll^R, syn_opt_effort^G, leakage_power_effort^G, nICacheTLBWays^R, and dp_analytical_opt^G. Parameters with a superscript R belong to Rocket parameters, while the parameters with a superscript G belong to Genus parameters.

G belong to Genus parameters. Increasing the number of nWays^R and Unroll^R improves CPI, but also increases power consumption and area. But, logic synthesis tool parameters can be adjusted to mitigate these tradeoffs, resulting in a design that achieves better-power consumption and area results.

E. Discussion

Due to the time-consuming optimization of DKL-GP in high dimensions, BOOM-Explorer's optimization time significantly lags behind other methods. GRL-DSE exhibits mediocre HV because training an accurate GNN for the entire high-dimensional design space is challenging. BODi outperforms BOOM-Explorer and GRL-DSE by employing dictionary embedding, which is more suitable for high-dimensional problems. However, it disregards the varying importance of design parameters, resulting in inferior performance compared to ROI-HIT. BODi's hill-climbing local search requires more acquisition evaluations than sampling methods in GRL-DSE, MORBO, and REMOTune, leading to a longer optimization time. Although MORBO and REMOTune are designed for high-dimensional multiobjective optimization, they achieve worse results than ROI-HIT in both sequential and batch modes because they ignore the relationship between design parameters and PPA results, ignoring the different importance of variables. Additionally, these trust region-based methods exhibit limited robustness, as demonstrated in experiments for RISC-V BOOM and Rocket microarchitectures.

Our proposed ROI-HIT achieves better PPA results by focusing on promising ROIs and important variables. The asynchronous batch technique in ROI-HIT enables it to explore more points within the limited time budget. Moreover, our approach provides designers with valuable insights into RISC-V SoCs by identifying the important variables (key design parameters). This information is beneficial for microarchitecture DSE.

V. Conclusion

In this article, we propose an ROI-driven microarchitecture DSE method called ROI-HIT. This method exploits ROIs acquired by a SOM Tree and then prunes unimportant variables based on a sensitivity matrix. By optimizing inside the more promising ROIs and reduced dimensions, this approach greatly improves the exploration efficiency in high-dimensional DSE. For time-consuming VLSI flow simulation, an asynchronous parallel strategy is employed to make ROI-HIT explore more points in the limited time budget. Experimental results conducted on the RISC-V BOOM and Rocket microarchitectures demonstrate that our proposed method achieves a 3.47%–9.19% improvement of HV within the same time budget and 3.58×–341.68× speed-up of time in reaching the same HV compared to the state-of-the-art methods. In terms of PPA metrics, ROI-HIT achieves improvements of up to 43.82% in performance, 33.20% in power consumption, and 11.41% in area. By leveraging SOMT-based space partitioning, sensitivity matrix-based VS, and an asynchronous parallel strategy, ROI-HIT can effectively handle black-box optimization problems with large parameter space, high dimensionality, and expensive simulations. We believe that ROI-HIT can find broader applications in future research. The code of ROI-HIT is available at https://github.com/zxy6541/ROI-HIT.

References

[1] A. D. Pimentel, “Exploring exploration: A tutorial introduction to embedded systems design space exploration,” *IEEE Design Test*, vol. 34, no. 1, pp. 77–90, Feb. 2017.

[2] S. Greengard, “Will RISC-V revolutionize computing?” *Commun. ACM*, vol. 63, no. 5, pp. 30–32, 2020.

[3] A. Amid et al., “Chipyard: Integrated design, simulation, and implementation framework for custom SoCs,” *IEEE Micro*, vol. 40, no. 4, pp. 10–21, Aug. 2020.

[4] J. Bachrach et al., “Chisel: Constructing hardware in a scala embedded language,” in *Proc. Design Autom. Conf. (DAC)*, 2012, pp. 1212–1221.

[5] C. Celio, D. A. Patterson, and K. Asanovic, “The Berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized RISC-V processor,” Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Rep. UCB/EECS-2015-167, 2015.

[6] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, “SonicBOOM: The 3rd generation Berkeley out-of-order machine,” in *Proc. 4th Workshop Comput. Archit. Res. RISC-V*, 2020, pp. 1–7.

[7] K. Asanovic et al., “The rocket chip generator,” Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Rep. UCB/EECS-2016-17, 2016.

[8] N. Binkert et al., “The gem5 simulator,” *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[9] H. Liew et al., “Hammer: A modular and reusable physical design flow tool,” in *Proc. 59th ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 1335–1338.

[10] W. Snyder, “Verilator 4.0: Open simulation goes multithreaded,” in *Proc. Open Source Digit. Design Conf. (ORConf)*, 2018, pp. 1–31.

[11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[12] Q. Zhang and H. Li, “MOEA/D: A multiobjective evolutionary algorithm based on decomposition,” *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, Dec. 2007.

[13] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, “BOOM-explorer: RISC-V BOOM microarchitecture design space exploration framework,” in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2021, pp. 1–9.

[14] X. Yi, J. Lu, X. Xiong, D. Xu, L. Shang, and F. Yang, “Graph representation learning for microarchitecture design space exploration,” in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.

[15] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek, “Scalable global optimization via local Bayesian optimization,” in *Proc. 33rd Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 1–12.

[16] S. Daulton, D. Eriksson, M. Balandat, and E. Bakshy, “Multi-objective Bayesian optimization over high-dimensional search spaces,” in *Proc. 38th Conf. Uncertainty Artif. Intell. (UAI)*, 2022, pp. 507–517.

[17] Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. De Feitas, “Bayesian optimization in a billion dimensions via random embeddings,” *J. Artif. Intell. Res.*, vol. 55, no. 1, pp. 361–387, 2016.

[18] A. Deshwal, S. Ament, M. Balandat, E. Bakshy, J. R. Doppa, and D. Eriksson, “Bayesian optimization over high-dimensional combinatorial spaces via dictionary-based embeddings,” in *Proc. Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2023, pp. 7021–7039.

[19] S. Zheng, H. Geng, C. Bai, B. Yu, and M. D. Wong, “Boosting VLSI design flow parameter tuning with random embedding and multi-objective trust-region Bayesian optimization,” *ACM Trans. Design Autom. Electron. Syst.*, vol. 28, no. 5, pp. 1–23, 2023.

[20] C. E. Rasmussen, “Gaussian processes in machine learning,” in *Advanced Lectures on Machine Learning*. Tübingen, Germany: Springer, 2003.

[21] M. T. Emmerich, K. C. Giannakoglou, and B. Naujoks, “Single-and multiobjective evolutionary optimization assisted by Gaussian random field metamodels,” *IEEE Trans. Evol. Comput.*, vol. 10, no. 4, pp. 421–439, Aug. 2006.

[22] S. Daulton, M. Balandat, and E. Bakshy, “Parallel Bayesian optimization of multiple noisy objectives with expected hypervolume improvement,” in *Advances in Neural Information Processing Systems*, vol. 34, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds. Red Hook, NY, USA: Curran Associates, Inc., 2021, pp. 2187–2200.

[23] T. Kohonen, “The self-organizing map,” *Proc. IEEE*, vol. 78, no. 9, pp. 1464–1480, Sep. 1990.

[24] T. Kohonen, “Essentials of the self-organizing map,” *Neural Netw.*, vol. 37, pp. 52–65, Jan. 2013.

[25] A. Zhao et al., “D3PBO: Dynamic domain decomposition-based parallel Bayesian optimization for large-scale analog circuit sizing,” *ACM Trans. Design Autom. Electron. Syst.*, vol. 29, no. 3, pp. 1–25, 2024.

[26] L. Adjengue, C. Audet, and I. Ben Yahia, “A variance-based method to rank input variables of the mesh adaptive direct search algorithm,” *Optim. Lett.*, vol. 8, pp. 1599–1610, Jun. 2014.

[27] L. Song, K. Xue, X. Huang, and C. Qian, “Monte carlo tree search based variable selection for high dimensional Bayesian optimization,” in *Proc. 36th Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2022, pp. 28488–28501.

[28] L. T. Clark et al., “ASAP7: A 7-nm finFET predictive process design kit,” *Microelectron. J.*, vol. 53, pp. 105–115, Jul. 2016.