



BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration

CHEN BAI, The Chinese University of Hong Kong, Hong Kong SAR

QI SUN, Zhejiang University, China

JIANWANG ZHAI, Beijing University of Posts and Telecommunications, China

YUZHENG MA, Hong Kong University of Science and Technology (Guangzhou), China

BEI YU, The Chinese University of Hong Kong, Hong Kong SAR

MARTIN D. F. WONG, Hong Kong Baptist University, Hong Kong SAR

Microarchitecture parameters tuning is critical in the microprocessor design cycle. It is a non-trivial design space exploration (DSE) problem due to the large solution space, cycle-accurate simulators' modeling inaccuracy, and high simulation runtime for performance evaluations. Previous methods require massive expert efforts to construct interpretable equations or high computing resource demands to train black-box prediction models. This article follows the black-box methods due to better solution qualities than analytical methods in general. We summarize two learned lessons and propose BOOM-Explorer accordingly. First, embedding microarchitecture domain knowledge in the DSE improves the solution quality. Second, BOOM-Explorer makes the microarchitecture DSE for register-transfer-level designs within the limited time budget feasible. We enhance BOOM-Explorer with the diversity-guidance, further improving the algorithm performance. Experimental results with RISC-V Berkeley-Out-of-Order Machine under 7-nm technology show that our proposed methodology achieves an average of 18.75% higher Pareto hypervolume, 35.47% less average distance to reference set, and 65.38% less overall running time compared to previous approaches.

CCS Concepts: • **Computer systems organization** → *Superscalar architectures*; • **Computing methodologies** → **Machine learning approaches**;

Additional Key Words and Phrases: Microprocessor, microarchitecture, design space exploration

This work is supported by National Key R&D Program of China (2022YFB2901100), The Research Grants Council of Hong Kong SAR (No. CUHK14210723), and The Zhejiang University Education Foundation Qizhen Scholar Foundation.

Authors' addresses: C. Bai, Room 122, Ho Sin Hang Engineering Building, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong SAR; e-mail: cbai@cse.cuhk.edu.hk; Q. Sun, Bd A04, No. 2118, Pinglan Rd., ZJU-Hangzhou Global Scientific and Technological Innovation Center, Xiaoshan District, Hangzhou, China; e-mail: qisunchn@zju.edu.cn; J. Zhai, Room 111, Scientific-Research Building, School of Integrated Circuits, Beijing University of Posts and Telecommunications, Beijing, China; e-mail: zhajw@bupt.edu.cn; Y. Ma, W4-511, No. 1 Duxue Road, Nansha, Guangzhou, China; e-mail: yuzhema@hkust-gz.edu.cn; B. Yu, Room 907, Ho Sin Hang Engineering Building, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong SAR; e-mail: byu@cse.cuhk.edu.hk; M. D. F. Wong, Room 801B, Shaw Tower, Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Kowloon, Hong Kong SAR; e-mail: prov@hkbu.edu.hk.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

1084-4309/2023/12-ART20 \$15.00

<https://doi.org/10.1145/3630013>

ACM Reference format:

Chen Bai, Qi Sun, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin D. F. Wong. 2023. BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration. *ACM Trans. Des. Autom. Electron. Syst.* 29, 1, Article 20 (December 2023), 23 pages.
<https://doi.org/10.1145/3630013>

1 INTRODUCTION

Recently, RISC-V, an open-source **instruction set architecture (ISA)**, has gained much attention and received strong support from academia and industry [1]. The RISC-V microprocessor design cycle requires a high workforce and design time input due to the co-optimization with architecture and physical implementation. Tuning microarchitecture parameters is crucial in the design cycle, as it can reduce non-recurring engineering efforts [2] and meet the product delivery deadline [3].

It is non-trivial to determine promising RISC-V microarchitecture parameters combinations within the limited time budget. The reason lies in three factors. First, the design space size can be billions or even trillions as architects consider more components such as emerging functional units or choices of speculation techniques [4–6]. Second, it costs high runtime to estimate the performance of a microarchitecture with high-fidelity simulations. Third, architects often use **cycle-accurate simulators (CAS)** [7–11] in the decision procedure, but relatively low CAS performance modeling accuracy is a concern. A CAS builds a microprocessor model with a high-level programming language to trade off the modeling efficiency and accuracy. The distinction between the model and actual circuit implementation cannot be eliminated. And positive correlations between modeled performance results and *de facto* performance values remain unclear. Hence, a practical, accurate, and efficient **design space exploration (DSE)** algorithm is necessary for architects.

Many DSE algorithms have been proposed to solve the problem [16–22] and can be categorized as analytical and black-box methodologies. The analytical methods require massive expert efforts to construct interpretable equations or microexecution graph representations to model microarchitecture performance [19, 22–25]. The equations or graph representations are then leveraged to prune the design space or enable fast DSE. Karkhanis and Smith construct analytical models to explore an out-of-order superscalar microprocessor [19]. Golestani et al. [22] enhance the dynamic event-dependence graph for DSE with an Alpha21264-like microprocessor [26]. When domain knowledge cannot be accessed, black-box methods emerge to confront the dilemma [18, 20, 21, 27]. Ipek et al. [18] adopt the **artificial neural network (ANN)**. Li et al. [21] employ statistical sampling with AdaBoost.RT black-box models [28] to conduct DSE. When it comes to solution quality, black-box methods often outperform analytical methods due to the better prediction and searching capability of machine-learning models.

Nonetheless, previous black-box methods have two limitations in improving the DSE algorithm further. *First, purely driven by the algorithm rather than tightly coupled with expert knowledge restricts DSE efficiency improvement.* Architects already know the characteristics of many microarchitectures. For example, the **return address stack (RAS)** size is relative to the programs' nested function call depth. A large-size RAS causes resource waste when the microprocessor executes benchmarks containing few nested function calls. The number of physical registers depends on the number of instructions in the pipeline, which is upper bound by the pipeline width and depth. Embedding the prior knowledge in the DSE procedure helps the algorithm probe interesting regions of the design space with potentially more promising solutions straightly. *Second, the inaccuracy of the cycle-accurate simulators' modeling leads to sub-optimal DSE solutions.* The lack of detailed circuit implementation, i.e., a **register-transfer-level (RTL)** design, such as hardware components connections, signal handshake delays between pipeline stages, speculation mechanisms, and so on, can result in incorrect simulations. Consequently, sub-optimal DSE solutions are generated.

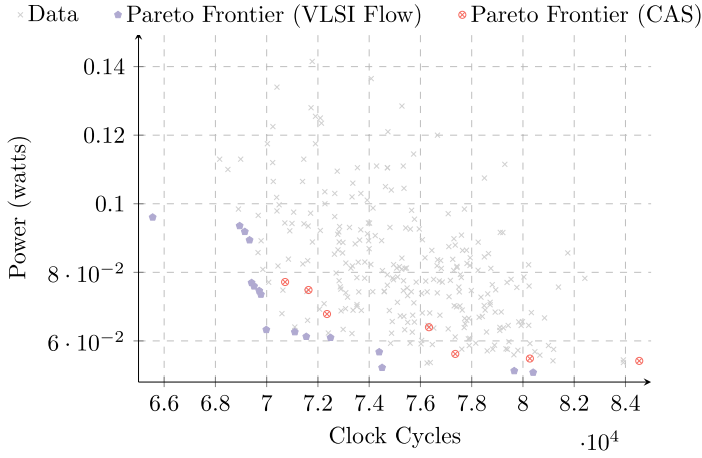


Fig. 1. Comparison of Pareto frontiers between the exploration w.r.t. the VLSI flow and CAS. We use around 300 BOOM microarchitecture designs [12–14]. Each design is estimated with the same benchmarks [15]. The Pareto frontier attained by CAS differs from that of VLSI flow. Specifically, the design colored in red deviates from the true Pareto frontier (highlighted with a purple pentagon) by a margin.

Given two limitations, we summarize two learned lessons to solve the problem based on prior works accordingly. *First, embedding microarchitecture domain knowledge in the DSE algorithm is necessary.* It can directly guide the exploration to the regions with potentially more performance-power Pareto-optimal microarchitectures, reducing dispensable simulations. *Second, applying the DSE algorithm to RTL designs prevents sub-optimal solutions.* Figure 1 elucidates the claim. We use the **very-large-scale integration (VLSI)** flow to evaluate the performance and power of RTL designs. The VLSI flow leverages commercial **electronic design automation (EDA)** tools to conduct logic synthesis, RTL simulations, power analysis, and so on. Regrettably, the VLSI flow incurs higher runtime costs compared to CAS simulations. While performing DSE w.r.t. RTL designs directly is an ideal solution, the significant runtime cost associated with the VLSI flow poses a restriction on the method.

This article proposes BOOM-Explorer, an automated DSE flow for RISC-V **Berkeley Out-of-Order Machine (BOOM)** [12–14] microarchitecture parameters tuning. BOOM is fully in compliance with RV64GC and demonstrates comparative performance efficiency in academia. The highlight of BOOM-Explorer is that our methodology makes DSE for RISC-V BOOM RTL designs feasible. The recipes are based on a combination of domain knowledge and dedicated algorithms. BOOM-Explorer is based on Bayesian optimization. Specifically, it integrates four key ingredients to help achieve better DSE algorithm performance compared to previous approaches and even state-of-the-art methods for RTL designs. Our contributions can be summarized as follows:

- (1) MicroAL, a microarchitecture-aware initialization algorithm based on active learning, is proposed to sample the potential most valuable microarchitectures via domain knowledge injection. The formulated initialization dataset benefits the construction of the surrogate model in Bayesian optimization.
- (2) DKL-GP, the Gaussian process with deep kernel learning, is proposed as an effective surrogate model to characterize the microarchitectures' performance and power.
- (3) The **expectation improvement on Pareto hypervolume (EIPV)** is introduced to handle the negatively correlated multi-objective (performance and power) optimization, providing reliable DSE guidance progressively.

- (4) A diversity-guided exploration strategy is proposed and coupled with the batch Bayesian optimization flow, enhancing the solution quality. Better performance-power Pareto-optimal microarchitecture is achieved, outperforming the previous solution and human implementations.
- (5) The evaluation results show that under 7-nm technology [29], BOOM-Explorer achieves an average of 18.75% higher Pareto hypervolume, 35.47% less average distance to reference set, and 65.38% less overall running time compared to previous approaches.

The remainder of this article is organized as follows: Section 2 introduces the related work. Section 3 provides the problem formulation. Section 4 details BOOM-Explorer. Section 5 is for experimental evaluations. Section 6 concludes this article.

2 RELATED WORK

2.1 RISC-V BOOM Core

Figure 2 illustrates the overall pipeline organization of BOOM [12–14]. BOOM is mainly composed of a **front end (FrontEnd)**, an **instruction decoding unit (IDU)**, an **execution unit (EU)**, and a **load-store unit (LSU)**. FrontEnd fetches instructions from I-cache, predicting branch target addresses, handling return addresses, and packing consecutive instructions as a fetch packet to the fetch buffer. IDU decodes instructions retrieved from the fetch buffer as micro-ops and dispatches, schedules, and issues them according to instruction types. EU integrates various functional units, including dividers, multipliers, accelerator interfaces, and so on. LSU interacts with EU and D-cache, deciding when to fire memory instructions to D-cache. BOOM implements explicit renaming logics, short-forwards branch optimizations, and integrates a two-level branch predictor [30] to improve its overall performance. With high-level hardware description language like Chisel [31], many components and their connections can be parameterized to support various BOOM microarchitectures. For example, the parameterization of the size of the branch target buffer, RAS, branch prediction history tables, and so on, broadens BOOM’s potential to balance the performance and power dissipation in FrontEnd. We can achieve divergent tradeoffs by adjusting these parameters to meet design requirements involving low-power and embedded computation applications.

A microarchitecture design space of BOOM is constructed according to BOOM’s overall architecture, as listed in Table 1. The design space of each module is composed of various components, the structure of which can affect the performance power tradeoff and deserve to be optimized. As shown in Table 1, different **entries of RAS (RasEntry)**, **branch counters (BranchCount)**, organization of I-cache/D-cache (associativity, block width, and size), **translation lookaside buffer (TLB)** structures are considered in this article. The column of “Candidate values” in Table 1 denotes supported hardware resources. For example, the reorder buffer is provided to support 32, 64, or 96 entries, and so on. Performance and power are negatively correlated. Assigning more hardware resources often improves performance and brings considerable power dissipation. Hence, a good microarchitecture demands an appropriate compromise across all components’ hardware resources.

Some combinations in Table 1 are illegal. A legal combination should observe the constraints of BOOM design specifications as in Table 2. Otherwise, it would fail to generate reasonable RTL designs. For example, DecodeWidth defines the maximal instructions to be decoded simultaneously. If a BOOM microarchitecture breaks rule 2 in Table 2, then the reorder buffer may not reserve enough entries for each decoded instruction or contain redundant entries that we cannot fully utilize. The last three rules in Table 2 are added to simplify the design space. Their incorporations do not affect the design of a DSE algorithm. After we prune the design space w.r.t. rules in Table 2, the size of the legal microarchitecture design space is approximately 1.6×10^8 .

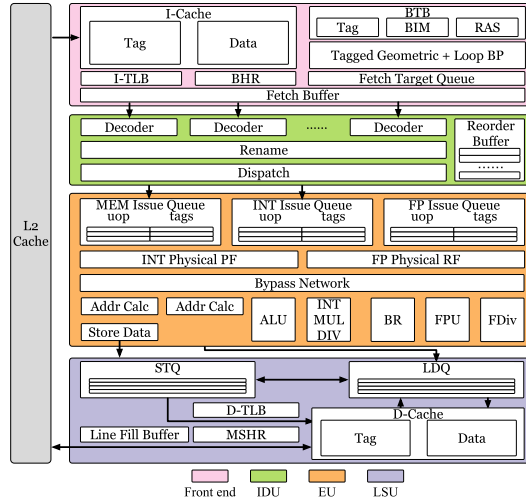


Fig. 2. BOOM implements a ten-stage pipeline including Fetch, Decode, Register Rename, Dispatch, Issue, Register Read, Execute, Memory, Writeback, and Commit.

Table 1. Microarchitecture Design Space of BOOM

Module	Component	Descriptions	Candidate values
FrontEnd	FetchWidth	The number of instructions the instruction fetch unit can retrieve once	4, 8
	FetchBufferEntry	The entries of the instruction fetch buffer	8, 16, 24, 32, 35, 40
	RasEntry	The entries of the return address stack (RAS)	16, 24, 32
	BranchCount	The maximal number of branches can be speculated simultaneously	8, 12, 16, 20
	ICacheWay	The associative sets of L1 I-cache	2, 4, 8
	ICacheTLB	The ways of look-aside buffer (TLB) for L1 I-cache	8, 16, 32
IDU	ICacheFetchBytes	The L1 I-cache line capacity	2, 4
	DecodeWidth	The maximal number of instructions the decoding unit can decode once	1, 2, 3, 4, 5
	RobEntry	The entries of the reorder buffer	32, 64, 96, 128, 130
	IntPhyRegister	The number of physical integer registers	48, 64, 80, 96, 112
EU	FpPhyRegister	The number of physical floating-point registers	48, 64, 80, 96, 112
	MemIssueWidth	The width of memory-related instructions issue slot	1, 2
	IntIssueWidth	The number of integer-related instructions issue slot	1, 2, 3, 4, 5
	FpIssueWidth	The number of floating point-related instructions issue slot	1, 2
LSU	LDQEntry	The entries of the load queue (LDQ)	8, 16, 24, 32
	STQEntry	The entries of the store queue (STQ)	8, 16, 24, 32
	DCacheWay	The associative sets of L1 D-cache	2, 4, 8
	DCacheMSHR	The numbers of miss status handling register (MSHR)	2, 4, 8
	DCacheTLB	The ways of look-aside buffer (TLB) for L1 D-cache	8, 16, 32

2.2 Bayesian Optimization

CAS offers advantages in terms of fast performance estimations compared to the VLSI flow mentioned earlier in Section 1. Thus, it becomes convenient for previous approaches to construct extensive training datasets using CAS [18, 21]. These datasets allow black-box models to learn the relationships between microarchitectures and their corresponding performance or power values. DSE is then conducted by sweeping the design space using these trained black-box models. However, given the benefits of DSE at the RTL level instead of using CAS (see Figure 1), such a new solution should aim to reduce the call for the expensive VLSI flow effectively. We find that the inherent structure of the problem fits Bayesian optimization well. So, we present the corresponding preliminary in this section.

Table 2. Constraints of BOOM Design Specifications

Rule	Descriptions
1	$\text{FetchWidth} \geq \text{DecodeWidth}$
2	$\text{RobEntry} \mid \text{DecodeWidth}^+$
3	$\text{FetchBufferEntry} > \text{FetchWidth}$
4	$\text{FetchBufferEntry} \mid \text{DecodeWidth}$
5	$\text{fetchWidth} = 2 \times \text{ICacheFetchBytes}$
6	$\text{IntPhyRegister} = \text{FpPhyRegister}$
7	$\text{LDQEntry} = \text{STQEntry}$
8	$\text{MemIssueWidth} = \text{FpIssueWidth}$

⁺“ \mid ” means RobEntry should be divisible by DecodeWidth.

Bayesian optimization [32, 33] is widely applied in such optimization problems when an evaluation flow is expensive, as shown in Equation (1), where we use the minimization as an example.

$$x^* = \arg \min_{x \in \mathcal{X}} f(x), \quad (1)$$

where \mathcal{X} is the solution space (design space), and f represents the evaluation flow, which maps x to a metric value. The optimal solution x^* attains the minimal value of $f(x)$. The main idea of Bayesian optimization is only to select the potentially promising microarchitectures for evaluation using the VLSI flow. These solutions are predicted promising based on what we have known in previous searched samples. A distinction between DSE using Bayesian optimization and previous black-box methods is the absence of the need to construct a large dataset for training a black-box model. We instead progressively explore the design space in an online fashion.

Bayesian optimization consists of a *surrogate model* and an *acquisition function*. A surrogate model is often constructed from the **Gaussian process (GP)** [34]. It models f in Equation (1) as a generated probability distribution from the GP. The acquisition function is designed to characterize the relative rankings between different solutions based on the probability distribution. In other words, the acquisition function answers whether x_1 is better than x_2 without considering exactly how much better x_1 performs than x_2 . The solution that achieves the acquisition function's optimum is selected as a potential promising/optimal solution. The potential optimal solutions are then evaluated using the VLSI flow and their corresponding performance and power values are utilized to tune the surrogate model. Better solutions are expected to be explored using the tuned surrogate model with a more accurate modeled probability distribution. Suppose \mathbf{x} denotes a microarchitecture embedding (Section 3), τ denotes the current explored best performance/power value. GP characterizes the evaluation flow with the mean $\mu(\mathbf{x})$ and the covariance $\sigma^2(\mathbf{x})$. The **expected improvement (EI)**, a popular acquisition function, is shown in Equation (2).

$$\begin{aligned}
 \text{EI}(\mathbf{x}) &= \mathbb{E}[\min(f(\mathbf{x}) - \tau, 0)] \\
 &= \mathbb{E} \left[\min \left(\frac{f(\mathbf{x}) - \mu(\mathbf{x})}{\sqrt{\sigma^2(\mathbf{x})}} - \frac{\tau - \mu(\mathbf{x})}{\sqrt{\sigma^2(\mathbf{x})}}, 0 \right) \right] \cdot \sqrt{\sigma^2(\mathbf{x})} \\
 &= \sigma(\mathbf{x})(\lambda \Phi(\mathbf{x}) + \phi(\mathbf{x})), \\
 \lambda &= \frac{\tau - \xi - \mu(\mathbf{x})}{\sigma(\mathbf{x})},
 \end{aligned} \quad (2)$$

where $\Phi(\mathbf{x})$ and $\phi(\mathbf{x})$ are the cumulative distribution function and the probability density function of GP, and ξ is a coefficient to improve the numerical ability. Figure 3 visualizes the optimization procedure for maximizing $f(x)$ with Bayesian optimization. A surrogate model with GP is built in

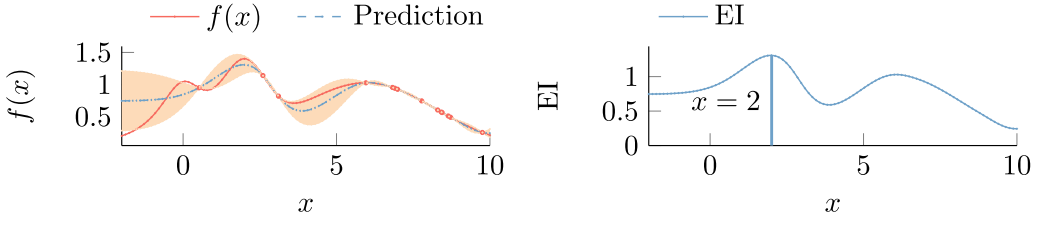


Fig. 3. An example of Bayesian optimization (with the EI function): find x , which attains the maximal value of $f(x)$.

the $x - f(x)$ space from already-known samples represented by red dots. The red curve denotes the golden values (ground truths) for each input x , and the dashed blue curve denotes GP predictions. The band colored in orange shows the GP prediction uncertainty of each x . If the band is wider, then the uncertainty is larger. The potential optimal solution is selected according to EI. Namely, point $x = 2$, which achieves the maximal EI, is chosen as a potential optimal solution, as shown in the $x - EI$ space.

3 PROBLEM FORMULATION

Definition 1 (Microarchitecture Embedding). Microarchitecture embedding is a feature vector \mathbf{x} , denoting a combination of candidate values given in Table 1, and it satisfies all constraints, as referred to in Table 2.

Definition 2 (Clock Cycle). The clock cycle is defined as the clock cycles spent when a BOOM executes a benchmark. It serves as a proxy for performance measurement.

Definition 3 (Power). Power is defined as the summation of dynamic, short-circuit, and leakage power dissipation.

Given the same benchmark, performance, and power conflict, as mentioned in Section 2.1. Clock cycles and the power of a BOOM microarchitecture are denoted as a vector \mathbf{y} .

Definition 4 (Pareto Optimality). For an n -objective minimization problem, a vector of objective values $f(\mathbf{x})$ is said to dominate $f(\mathbf{x}')$ if

$$\begin{aligned} \forall i \in [1, n], \quad f_i(\mathbf{x}) &\leq f_i(\mathbf{x}'); \\ \exists j \in [1, n], \quad f_j(\mathbf{x}) &< f_j(\mathbf{x}'), \end{aligned} \quad (3)$$

where \mathbf{x} and \mathbf{x}' are two microarchitecture embeddings. Hence, we denote $f(\mathbf{x}) \geq f(\mathbf{x}')$. Otherwise, $f(\mathbf{x}) \not\geq f(\mathbf{x}')$. A set of objective values that are not dominated by any other is called the Pareto frontier. Their corresponding microarchitectures are termed Pareto-optimal set.

We aim to explore microarchitecture embeddings, whose objective values are the Pareto-optimal set in the design space. A microarchitecture whose objective values belong to the Pareto frontier cannot improve performance without sacrificing power, and vice versa. Based on the above definitions, we formulate the problem.

PROBLEM 1 (BOOM MICROARCHITECTURE DESIGN SPACE EXPLORATION). Given a design space \mathcal{D} , each microarchitecture embedding $\mathbf{x} \in \mathcal{D}$ corresponds to objective values \mathbf{y} in the clock cycle-power space \mathcal{Y} . BOOM microarchitecture design space exploration is to find the Pareto-optimal microarchitecture embeddings set $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, whose objective values $Y = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n\} \subset \mathcal{Y}$ formulate the Pareto frontier.

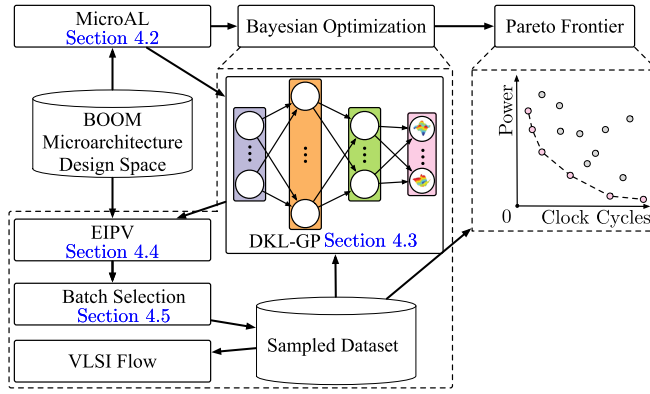


Fig. 4. Overview of the proposed BOOM-Explorer.

ALGORITHM 1: TED (\mathcal{U}, μ, b)

Input: \mathcal{U} is the unsampled microarchitecture design space, μ is a normalization coefficient, and b is the number of samples to draw.

Output: X : the sampled set with $|X| = b$.

1: $X \leftarrow \emptyset, K_{uu'} \leftarrow f(u, u'), \forall u, u' \in \mathcal{U};$

2: **for** $i = 1 \rightarrow b$ **do**

3: $\mathbf{x}_* \leftarrow \arg \max_{\mathbf{x} \in \mathcal{U}} \text{Tr}[K_{\mathcal{U}\mathbf{x}}(K_{\mathbf{x}\mathbf{x}} + \mu I)^{-1} K_{\mathbf{x}\mathcal{U}}];$

w.r.t. corresponding columns in K .

▷ $K_{\mathcal{U}\mathbf{x}}, K_{\mathbf{x}\mathbf{x}}$ and $K_{\mathbf{x}\mathcal{U}}$ are calculated via f

4: $X \leftarrow X \cup \mathbf{x}_*, \mathcal{U} \leftarrow \mathcal{U} \setminus \mathbf{x}_*;$

5: $K \leftarrow K - K_{\mathcal{U}\mathbf{x}_*}(K_{\mathbf{x}_*\mathbf{x}_*} + \mu I)^{-1} K_{\mathbf{x}_*\mathcal{U}};$

6: **end for**

7: **return** The sampled set X ;

4 BOOM-EXPLORER

4.1 Overview of BOOM-Explorer

Figure 4 shows an overview of BOOM-Explorer. First, the active learning algorithm MicroAL (Section 4.2) is adopted to sample a set of initial microarchitectures from the design space. Domain-specific knowledge is embedded in the initialization based on the first learned lesson. Second, a **Gaussian process model with deep kernel learning (DKL-GP)** (Section 4.3) is then built on the initial set as a surrogate model. Third, the expectation improvement of Pareto hypervolume (Section 4.4) is applied as the acquisition function. During the DSE procedure, BOOM-Explorer interacts with the VLSI flow to acquire performance and power values for the selected microarchitectures. Due to the customized algorithm flow, such interactions make the DSE at the RTL level feasible. Moreover, we improve the exploration with diversity-guided sampling to handle different sub-regions (Section 4.5). Finally, The outputs of BOOM-Explorer are the predicted Pareto frontier and corresponding Pareto-optimal microarchitectures.

4.2 Microarchitecture-aware Active Learning Algorithm

The initial microarchitectures sampled from the design space are critical to constructing a surrogate model for later exploration. A naive solution is to sample microarchitecture embeddings randomly [18, 27]. Some advanced sampling techniques with statistic analysis like orthogonal design [21] are also applied in previous works. Nevertheless, the methods mentioned above are less

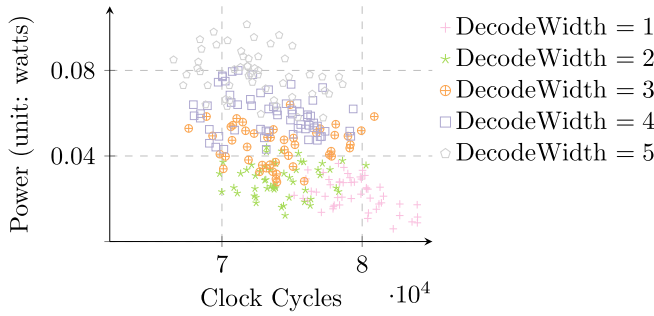


Fig. 5. Visualization of clusters w.r.t. DecodeWidth.

effective, since they require many VLSI flow interactions while the runtime cost of the VLSI flow is much higher. We follow two guidelines to design the initialization algorithm. First, the initial microarchitecture embeddings should uniformly scatter in the entire design space. Different PPA tradeoffs should be captured via uniform scattering. Second, the diversity of these microarchitecture embeddings should represent different characteristics of the design space as much as possible. Simple characteristics can be described with a few samples, while complex ones can be described with more samples. Besides, combining with prior knowledge helps us remove samples unworthy to be evaluated with the VLSI flow.

In MicroAL, we first perform clustering w.r.t. the **decode width (DecodeWidth)**, then we conduct transductive experimental design with samples per each cluster. DecodeWidth in IDU, as referred to in Table 1, decides the maximal numbers of instructions that can be decoded simultaneously, i.e., it determines the width of the pipeline, as shown in Figure 2. It allows more instruction execution parallelism in the pipeline if DecodeWidth is assigned a considerable value in BOOM, and we also allocate appropriate hardware resources to other components accordingly. Although large DecodeWidth leads to remarkable performance improvement, in most cases, power dissipation increases correspondingly due to more transistors integrated. We find that the impact of a PPA tradeoff by configuring DecodeWidth is significant.

Figure 5 visualizes the objective space by clustering w.r.t. DecodeWidth on sampled microarchitecture embeddings. Better microarchitectures will be closer to the original point in Figure 5, indicating higher performance and lower power dissipation. Figure 5 demonstrates that the distributions of clusters are highly correlated with the potential Pareto frontier. The entire design space is discrete and non-smooth. Nonetheless, many microarchitectures with the same DecodeWidth achieve similar performance-power characteristics within their clusters. We embed this domain knowledge in the initialization algorithm. Additionally, within each cluster, we adopt the **transductive experimental design (TED)** [35] to sample microarchitectures that best reflect the characteristics of their groups.

The method of TED is a widely used algorithm to construct samples that deserve to be evaluated with an expensive flow. It tends to choose microarchitecture embeddings that can spread across the design space to retain the most information [35]. We can acquire a pool of representative samples with high mutual divergences by iteratively maximizing the trace of the distance matrix constructed on newly sampled and unsampled microarchitecture embeddings. Algorithm 1 shows the backbone of TED, where f represents the distance function used in computing the distance matrix K . It should be noted that there are no restrictions on the choice of distance functions. The microarchitecture embeddings can be clustered based on DecodeWidth, allowing us to identify clusters that are most closely related to the Pareto frontier. Within each cluster, TED is performed

ALGORITHM 2: MicroAL (\mathcal{U}, μ, b, n)

Input: \mathcal{U} is the unsampled microarchitecture design space, μ is a normalization coefficient, b is the number of samples to draw, n is the number of pre-determined iterations.

Output: X : the initial set with $|X| = b$.

```

1:  $X \leftarrow \emptyset$ ;
2: Randomly initialize  $k$  centroids  $\{c_1, c_2, \dots, c_k\}$  from  $\mathcal{U}$  with  $k$  equal to the number of candidate values of DecodeWidth.
3: while  $i = 1 \rightarrow n$  do
4:    $c^i \leftarrow \arg \min_{j \in \{1, 2, \dots, k\}} \Phi(\mathbf{x}_i - c_j), \forall \mathbf{x}_i \in \mathcal{U};$  ▷  $\Phi$  is a designed distance function considering DecodeWidth.
5:   Assign  $\mathbf{x}_i$  to  $\mathcal{C}_{c^i}$ , the centroid of which is  $c_{c^i}$ ;
6:    $c_j \leftarrow \frac{\sum_{i=1}^{|\mathcal{U}|} \mathbb{1}\{c^i=j\} \mathbf{x}_i}{\sum_{i=1}^{|\mathcal{U}|} \mathbb{1}\{c^i=j\}}, \forall j \in \{1, 2, \dots, k\};$ 
7: end while
8: Clusters  $\mathcal{C} \leftarrow \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k\}$  are formulated.
9: for  $\mathcal{C}_i \in \mathcal{C}$  do
10:   $\hat{X} \leftarrow \text{TED}(\mathcal{C}_i, \mu, \lfloor \frac{b}{k} \rfloor);$  ▷ Algorithm 1
11:   $X \leftarrow X \cup \hat{X};$ 
12: end for
13: return The initial microarchitecture embedding set  $X$ ;
```

to create representative samples. This process leads to the formulation of MicroAL, as described in detail in Algorithm 2.

In Algorithm 2, first, we cluster the entire design space according to Φ , which is the distance function with a higher penalty along the dimension of DecodeWidth. One possible alternative can be $\Phi = (\mathbf{x}_i - c_j)^\top \Lambda (\mathbf{x}_i - c_j)$, with $i \in \{1, \dots, |\mathcal{U}|\}$ and $j \in \{1, \dots, k\}$, where Λ is a pre-defined diagonal weight matrix. The procedure runs n rounds to make the design space sufficiently clustered. Second, we apply TED to each cluster to perform sampling, i.e., line 10 in Algorithm 2. Finally, the initial microarchitectures that are worth to be estimated with the VLSI flow are constructed. Each microarchitecture is estimated using the VLSI flow. Our acquisition function is built based on these microarchitectures and their respective performance and power values.

4.3 Gaussian Process with Deep Kernel Learning

It is common to build GP models for the initial microarchitectures due to the non-parametric approximation in terms of reliability in uncertainty estimation and robust performance for many applications [36, 37].

Without loss of generality, let $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ denote a set of microarchitecture embeddings. The corresponding objective values formulate a matrix:

$$Y = \begin{pmatrix} y_{11} & y_{12} & \dots & y_{1m} \\ y_{21} & y_{22} & \dots & y_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \dots & y_{nm} \end{pmatrix}, \quad (4)$$

where m denotes the number of objectives, and Y is an $n \times m$ matrix. The objective functions $\{f_1, f_2, \dots, f_m\}$, which characterize how we get the objective values, map an \mathbf{x} to $\mathbf{y} = \{y_1, y_2, \dots, y_m\}$. We place an appropriate GP prior on each f , i.e., $f_i(\mathbf{x}) \sim \mathcal{GP}(\mu_i, \sigma_i^2)$, where μ_i and σ_i^2 are the mean value function and the covariance function for the i th objective, respectively. A non-trivial problem is that the objectives in our problem are not orthogonal, since higher

performance often comes with higher power dissipation. Characterizing individual objectives using independent GP models could degrade the overall modeling performance. Hence, we introduce multi-task GP models to handle the difficulty [38]. The main idea of multi-task GP is to model the hidden mappings based on the *objective identities* and the observed objective values for each microarchitecture. Objective identities are scalars utilized to distinguish different objectives. Hence, a GP prior is also placed between each objective function, as shown in Equation (5).

$$\text{cov}(f_i(\mathbf{x}), f_j(\mathbf{x}')) = \mathbf{K}_{ij}^f \mathbf{K}^x(\mathbf{x}, \mathbf{x}'), \quad (5)$$

where $\text{cov}(f_i(\mathbf{x}), f_j(\mathbf{x}'))$ is a covariance between f_i and f_j on two microarchitecture embeddings \mathbf{x} and \mathbf{x}' , \mathbf{K}_{ij}^f is a positive semi-definite matrix specifying the similarities between objective i and j , and \mathbf{K}^x is a covariance function over \mathbf{x} and \mathbf{x}' .

Given a newly sampled microarchitecture embedding \mathbf{x}^* , the mean value representing the prediction of an objective value is shown in Equation (6).

$$\mu_i(\mathbf{x}^*) = \mu_i + \left(\mathbf{K}_i^f \otimes \mathbf{K}^x(\mathbf{x}^*, X) \right)^\top \Sigma^{-1} (Y_i - \mu_i), \quad (6)$$

where \mathbf{K}_i^f is the i th column of \mathbf{K}^f defined in Equation (5), μ_i denote a vector of i th objective mean values, and Y_i is the i th column of Y denoting the i th objective values in the training data set. The uncertainty of the prediction $\mu_i(\mathbf{x}^*)$ in Equation (6) is calculated by Equation (7).

$$\begin{aligned} \sigma_i^2(\mathbf{x}^*) = & \left(\mathbf{K}_i^f \otimes \mathbf{K}^x(\mathbf{x}^*, \mathbf{x}^*) \right) - \\ & \left(\mathbf{K}_i^f \otimes \mathbf{K}^x(\mathbf{x}^*, X) \right) \Sigma^\top \left(\mathbf{K}_i^f \otimes \mathbf{K}^x(X, \mathbf{x}^*) \right), \end{aligned} \quad (7)$$

where Σ in Equations (6) and (7) is obtained from Equation (8).

$$\Sigma = \left(\mathbf{K}_i^f \otimes \mathbf{K}^x(X, X) \right) + D \otimes I. \quad (8)$$

The operator \otimes denotes Kronecker product. In Equation (8), D is an $m \times m$ diagonal matrix with σ_k^2 as the k th diagonal element, and I is the identity matrix. Therefore, the posterior distribution is $f_i(\mathbf{x}^* | X, Y_i, \mathbf{K}_i^f, \mathbf{K}^x) \sim \mathcal{GP}(\mu_i(\mathbf{x}^*), \sigma_i^2(\mathbf{x}^*))$. The likelihood estimation of the multi-task GP is as shown in Equation (9).

$$\begin{aligned} l = & -\frac{mn}{2} \log 2\pi - \frac{n}{2} \log |\mathbf{K}^f| - \frac{m}{2} \log |\mathbf{K}^x| \\ & - \frac{1}{2} \text{Tr}[(\mathbf{K}^f)^{-1} \Gamma^\top (\mathbf{K}^x)^{-1} \Gamma] \\ & - \frac{n}{2} \sum_{i=1}^m \log \sigma_i^2 - \frac{1}{2} \text{Tr}[(Y - \Gamma) D^{-1} (Y - \Gamma)^\top], \end{aligned} \quad (9)$$

where Γ is a matrix of μ_{ij} corresponding to Y . The parameters describing \mathbf{K}^f and \mathbf{K}^x are optimized via the maximization of Equation (9).

We use deep kernels [39] stacked by **multiple linear perceptrons (MLP)** with non-linear transformations to construct the Gaussian kernels, i.e., \mathbf{K}^f and \mathbf{K}^x . We term our surrogate model DKL-GP due to the construction of the GP and deep kernel functions. The use of deep kernels can result in improved performance due to their strong modeling ability. Thus far, \mathbf{K}^f and \mathbf{K}^x are described by Equation (10).

$$\mathbf{K} \rightarrow (\varphi(g(\mathbf{x}, \theta)), \varphi(g(\mathbf{x}', \theta))), \quad (10)$$

where φ denotes non-linear transformation functions, e.g., the ReLU function, and so on, and the MLP g is parameterized by θ .

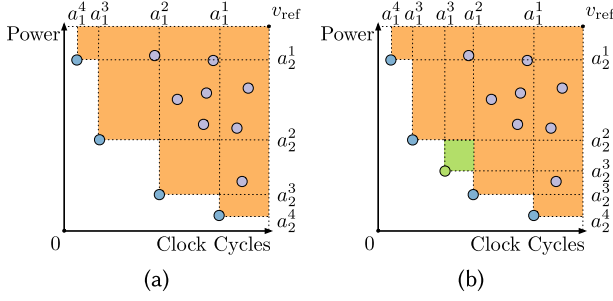


Fig. 6. An example of hypervolume is shown in the power-performance space. (a) The region covered in orange is dominated by the currently explored Pareto-optimal objective values denoted as circles in blue. Circles in purple denote dominated objective values. (b) The circle in green denotes an explored potential point belonging to the Pareto-optimal set among the entire design space. EIPV is represented as the area of the sub-region colored in light green.

4.4 Correlated Multi-objective Optimization

Although DKL-GP, as demonstrated in Section 4.3, can be utilized to characterize the uncertainty of predicted clock cycles and power, designing a suitable acquisition function to find the Pareto frontier remains an unsolved problem. Since clock cycles and power are a pair of negatively correlated metrics, we introduce the **expected improvement of Pareto hypervolume (EIPV)** [40] to model the tradeoff in the performance-power space.

We denote the points in the performance-power space as $Y = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n\}$. As Figure 6 shows, a better performance and power tradeoff lies closer to the origin. Pareto hypervolume is the volume of the area bounded by the Pareto frontier and a reference point. The reference point is a self-defined point dominated by all objective values. We use $\mathcal{P}(Y)$ to represent the Pareto frontier, i.e., $\mathcal{P}(Y) = \{\mathbf{y}_i \in Y \mid \mathbf{y}_j \not\preceq \mathbf{y}_i, \forall \mathbf{y}_j \in Y \setminus \{\mathbf{y}_i\}\}$. Given a reference point \mathbf{v}_{ref} in the objective space, which is dominated by $\mathcal{P}(Y)$. The Pareto hypervolume [40] bounded by \mathbf{v}_{ref} and $\mathcal{P}(Y)$, as the orange region highlighted in Figure 6(a), can be computed by Equation (11).

$$\text{PVol}_{\mathbf{v}_{ref}}(\mathcal{P}(Y)) = \int_Y \mathbb{1}[\mathbf{y} \geq \mathbf{v}_{ref}] \left[1 - \prod_{\mathbf{y}_* \in \mathcal{P}(Y)} \mathbb{1}[\mathbf{y}_* \not\preceq \mathbf{y}] \right] d\mathbf{y}, \quad (11)$$

where $\mathbb{1}(\cdot)$ is the indicator function, which outputs one if its argument is true and zero otherwise. The integral characterized by Equation (11) sums up all bounded regions. Intuitively, if a new point \mathbf{y}_{n+1} is searched out, and \mathbf{y}_{n+1} is not dominated by any points in Y , then $\text{PVol}_{\mathbf{v}_{ref}}(\mathcal{P}(Y \cup \{\mathbf{y}_{n+1}\}))$ is increased. The increased part is specified as the improvement of Pareto hypervolume. The larger the increased part, the better the improvement of Pareto hypervolume is. The EIPV is the expectation of the improvement w.r.t. potential solution candidates at the $(n+1)$ -th optimization steps. Formally, the EIPV is computed as Equation (12).

$$\text{EIPV}(\mathbf{x}_{n+1} \mid \mathcal{D}) = \mathbb{E}_{p(f(\mathbf{x}_{n+1}) \mid \mathcal{D})} [\text{PVol}_{\mathbf{v}_{ref}}(\mathcal{P}(Y) \cup f(\mathbf{x}_{n+1})) - \text{PVol}_{\mathbf{v}_{ref}}(\mathcal{P}(Y))], \quad (12)$$

where f is the DKL-GP mentioned in Section 4.3, $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$ is the dataset, and \mathbf{x}_{n+1} is a newly sampled microarchitecture embedding at the step $n+1$. Figure 6(b) visualizes the EIPV based on Figure 6(a), where the green region highlights $\text{EIPV}(\mathbf{x}_{n+1} \mid \mathcal{D})$.

In the performance-power space, we can simplify Equation (12) to make EIPV better computable by decomposing the space as grid cells. Assume $\mathbf{v}_{ref} = \{(a_1^0, a_2^0)\}$. The union of grid cells can be phrased as $\mathcal{C} = [a_1^1, a_1^0] \times [a_2^1, a_2^0] \times [a_1^2, a_1^1] \times [a_2^2, a_2^1] \times \dots \times [a_1^n, a_1^{n-1}] \times [a_2^n, a_2^{n-1}]$. Denote

$C_{\text{nd}} = \{C \in \mathcal{C} \mid \mathbf{y}' \not\preceq \mathbf{y}, \forall \mathbf{y} \in \mathcal{C}, \mathbf{y}' \in \mathcal{P}(Y)\}$ as non-dominated cells. Hence, the simplified version of EIPV computation is derived, as shown in Equation (13).

$$\text{EIPV}(\mathbf{x}_{n+1} \mid \mathcal{D}) = \sum_{C \in C_{\text{nd}}} \int_C \text{PVol}_{\mathbf{v}_C}(\mathbf{y}) p(\mathbf{y} \mid \mathcal{D}) d\mathbf{y}. \quad (13)$$

The relative relations between microarchitecture embeddings \mathbf{x}_1 and \mathbf{x}_2 , i.e., whether \mathbf{x}_1 is better than \mathbf{x}_2 , can be precisely described by EIPV. The microarchitecture embedding \mathbf{x}^* , which achieves the maximal EIPV, explicitly demonstrates that $f(\mathbf{x}^*)$ is the potential Pareto-optimal solution, as shown in Equation (14).

$$\mathbf{x}^* = \arg \max_{\mathbf{x} \in \mathcal{D}} \text{EIPV}(\mathbf{x} \mid \mathcal{D}). \quad (14)$$

We sample \mathbf{x}^* each time and evaluate its performance and power values \mathbf{y}^* via the VLSI flow. If a generous time budget is available, then we can sweep the design space \mathcal{D} using DKL-GP to obtain \mathbf{x}^* . The advantage of using DKL-GP is that it has low runtime, making the estimation highly efficient. The pair $(\mathbf{x}^*, \mathbf{y}^*)$ is added to \mathcal{D} . And we utilize the aggregated \mathcal{D} to tune DKL-GP, hoping to sample the following \mathbf{x}^* with better \mathbf{y}^* that can dominate already explored points in the design space.

4.5 Diversity-guided Parallel Exploration

Despite the benefits from MicroAL and the negatively correlated multi-objective Bayesian optimization flow, we also propose diversity-guided parallel exploration to further improve the algorithm's efficiency. With the technique, we can sample and evaluate more microarchitecture embeddings at the same time. And we also improve the overall DSE performance effectively.

Although exploring with EIPV is mostly effective, a limitation is also viewed. We notice that searching via EIPV can lead to local optimum, i.e., it cannot recover the complete Pareto frontier due to potential "outliers." Outliers have good properties in performance and power tradeoff but may not have high EIPV. The reason behind this is that the Pareto frontier tends to group in various regions due to components impacting the performance and power tradeoff in various degrees. Some objective values with relatively higher EIPV can hide outliers when we do not have a large optimization budget. In other words, exploring with EIPV misses such outliers when insufficient optimization is applied.

Two methodologies can be integrated to explore those outliers with higher Pareto hypervolume. First, improve the number of samples while maintaining an unchanged optimization budget with batch optimization. Applying parallel VLSI estimations in the batch optimization instead of original sequential optimization is necessary. And the parallelism depends on the number of available EDA tools licenses. Second, embedding domain knowledge or heuristics in the exploration is a good supplement to the exploration with original acquisition design (discussed in Section 4.4). Consequently, we propose a parallel-based exploration by combining two pathways.

In the exploration, we select multiple microarchitectures simultaneously to conduct parallel estimations using the VLSI flow. Intuitively, a straightforward way to select microarchitectures can be achieved according to the ranking of EIPV w.r.t. each design, as introduced in Section 4.4. Conversely, we provide some prior knowledge in the selection. DecodeWidth (mentioned in Section 4.2) contributes more to the performance and power tradeoff than other components, since modifying DecodeWidth can lead to distinct clusters shown in Figure 5. Except for DecodeWidth, another component IntPhyRegister, determining the number of integer physical registers, can also distinctly affect performance and power if a benchmark contains considerable integer-related instructions. Hence, the Pareto frontier is scattered in multiple sub-regions, as shown in Figure 7. In Figure 7, the DecodeWidth equals 1, and other components differ. We can observe that the Pareto

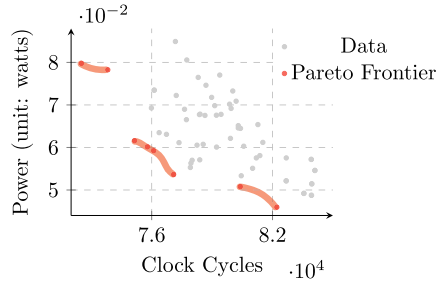


Fig. 7. The DecodeWidth equals 1 for the sampled microarchitectures, and the Pareto frontier they formulated disperses across different sub-regions, colored in red.

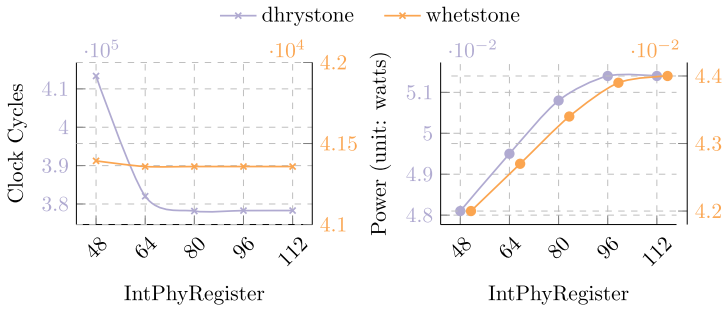


Fig. 8. The change of performance and power dissipation w.r.t. IntPhyRegister on dhrystone and whetstone.

frontier spread across multiple sub-regions, highlighted in red ribbons in Figure 7. An algorithm should inspect those sub-regions and sample microarchitecture embeddings across them, circumventing trapping into local optimum. Therefore, we propose the diversity-guided parallel exploration.

MicroAL leverages DecodeWidth as a critical factor to capture the main characteristics of the design space. Similarly, we also notice that IntPhyRegister can significantly affect the tradeoff. Figure 8 visualizes such impacts. Dhrystone and whetstone are integer instructions-intensive and floating-point instructions-intensive benchmarks, respectively. When the microarchitecture selects IntPhyRegister from 48 to 112, the performance is improved by 8.52% and dissipates 6.86% more power. Since whetstone is mainly composed of floating-point instructions, increasing IntPhyRegister introduces no change to the performance but worsens power dissipation more. BOOM implements the unified integer physical register files design, i.e., the registers hold both committed and speculative values/states.

More integer physical registers help to resolve more integer-related instructions conflicts, e.g., data dependencies, and provide more support for instruction parallelism. Most commonly used benchmarks contain many integer-related instructions, e.g., dhrystone, mm, median, and so on. Thus, the impact on the tradeoff between performance and power can highly correlate with IntPhyRegister on these benchmarks. The findings motivate us to propose the backbone for diversity guidance.

We leverage IntPhyRegister to partition the clustered design space by MicroAL, as demonstrated in Algorithm 3. In Algorithm 3, the clusters are obtained from MicroAL (line 2), and partitions within each cluster are formulated according to IntPhyRegister (line 5). Different sub-regions are constructed based on Algorithm 3. We sample microarchitecture embeddings according to the

ALGORITHM 3: Partition (\mathcal{U})**Input:** \mathcal{U} is the dataset already clustered by MicroAL.**Output:** The partitioned dataset \mathcal{U}' .

```

1:  $\mathcal{U}' \leftarrow \emptyset$ ;
2:  $\mathcal{C} \leftarrow \text{extract\_clusters}(\mathcal{U})$ ; ▷  $\mathcal{C}$  is obtained from line 8 of Algorithm 2
3: for  $\mathcal{C}_i \in \mathcal{C}$  do
4:   for  $\mathbf{x} \in \mathcal{C}_i$  do
5:      $\mathcal{U}'[\mathcal{C}_i][\mathbf{x}.\text{IntPhyRegister}] \leftarrow \mathbf{x}$ ;
6:   end for
7: end for
8: return  $\mathcal{U}'$ ;

```

ALGORITHM 4: Diversity-guided BOOM-Explorer ($\mathcal{D}, T, \mu, b, n$)**Input:** \mathcal{D} is the microarchitecture design space, T is the number of maximal iterations, μ is a normalization coefficient and b is the number of samples to draw, n is the number of pre-determined iterations for Algorithm 2.**Output:** The microarchitectures X that form the Pareto optimality in \mathcal{D} .

```

1:  $X_0 \leftarrow \text{MicroAL}(\mathcal{D}, \mu, b, n)$ ; ▷ Algorithm 2
2: Push  $X_0$  to the VLSI verification flow to obtain corresponding clock cycles and power values  $Y$ ;
3:  $\mathcal{L} \leftarrow X_0$ ;  $\mathcal{U} \leftarrow \mathcal{D} \setminus \mathcal{L}$ ;
4:  $\mathcal{P} \leftarrow \text{Partition}(\mathcal{U})$ ; ▷ Algorithm 3
5: for  $i = 1 \rightarrow T$  do
6:   Establish and train DKL-GP with  $(\mathcal{L}, Y)$ ;
7:   Select  $b$  unvisited partitions randomly from  $\mathcal{P}$ ;
8:   for  $j = 1 \rightarrow b$  do
9:      $\mathbf{x}_{j*} \leftarrow \arg \max_{\mathbf{x} \in \mathcal{P}_j} \text{EIPV}(\mathbf{x} \mid \mathcal{P}_j)$ ; ▷ Equation (13)
10:   end for
11:    $\mathbf{x}_* \leftarrow \{\mathbf{x}_{1*}, \mathbf{x}_{2*}, \dots, \mathbf{x}_{b*}\}$ ;
12:   Push  $\mathbf{x}_*$  to the VLSI flow to obtain corresponding clock cycles and power values and add results to  $Y$ ;
13:    $\mathcal{L} \leftarrow \mathcal{L} \cup \mathbf{x}_*$ ;  $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathbf{x}_*$ ;
14: end for
15: Construct  $X$  from  $\mathcal{L}$  that form the Pareto optimality, according to Equation (3);
16: return Pareto-optimal set  $X$ ;

```

highest EIPV from each sub-region, respectively, as Equation (15) shows,

$$\mathbf{x}^* = \left\{ \arg \max_{\mathbf{x}' \in \mathcal{P}_j} \text{EIPV}(\mathbf{x}' \mid \mathcal{P}_j) \mid j = 1, 2, \dots, N \right\}, \quad (15)$$

where \mathcal{P}_j denotes the partition j , and N is the total number of partitions. The sampled batch \mathbf{x}^* is evaluated with the VLSI flow in parallel.

Algorithm 4 details the whole framework. Line 4 performs the partition. Combined with Table 1 and MicroAL, the partition operation incurs 25 different sub-regions for the design space. We establish and train the surrogate model, DKL-GP, for each optimization round (line 6) and select unvisited partitions to aggregate the explored dataset (line 7). We maintain a *visited buffer* to record the access times to a specific partition in line 7. The framework extends the explored dataset gradually (line 13). Finally, the predicted Pareto frontier and corresponding microarchitecture embeddings are obtained. It is worth noting that Algorithm 4 requires at least b available EDA tools licenses to implement.

Other solutions could also be leveraged in Algorithm 3 to further improve the performance. First, some partitions can be directly pruned by the expertise. Architects have predetermined performance or power design goals. The selection of partitions can be made based on these goals. For example, we prefer to use a wider microarchitecture to pursue higher performance. So, we can only focus on partitions with decode width attaining the maximal value while neglecting other partitions, i.e., we only focus on “important” partitions. Second, the exploration-exploitation heuristic can be utilized, similar to the action selection in reinforcement learning [41]. We decide whether to exploit the current best-known partition that effectively achieves higher Pareto hypervolume or explore new partitions. The decision could be based on epsilon-greedy heuristics, upper confidence bound, and so on. We leave these discussed solutions in our future work.

5 EXPERIMENTS

We conduct comprehensive experiments to evaluate BOOM-Explorer.

5.1 Experiments Settings & Evaluation Metrics

We utilize Chipyard [42] to generate various BOOM RTL designs. And we use 7-nm ASAP7 PDK [29] for the VLSI flow. Cadence Genus 18.12-e012_1 is used to synthesize every sampled RTL design with 1 GHz timing constraints, and Synopsys VCS M-2017.03 is used to simulate the design with different benchmarks. PrimeTime PX R-2020.09-SP1 is leveraged to get power values for all benchmarks. All experiments are conducted in 80 cores of Intel(R) Xeon(R) CPU E7-4830 v2 @ 2.20 GHz with 1 TB main memory.

In the settings of BOOM-Explorer, DKL-GP is stacked with three hidden layers, each of which has 1,000, 500, and 50 hidden neurons, respectively, and it adopts ReLU as the non-linear transformation for deep kernels. The Adam optimizer [43] is used, with an initial learning rate equal to 0.001. BOOM-Explorer performs Bayesian exploration with 9 iterations. All experiments together with baselines are repeated 10 times, and we report corresponding average results.

5.2 Benchmarks, Baselines & Evaluation Metrics

To assess each microarchitecture, we employ a set of benchmarks selected from bare models [15]. These benchmarks include median, mt-vvadd, whetstone, mm, and so on. The benchmarks have covered all kinds of RV64G instructions, e.g., integer-related, floating-point numbers-related, memory-related instructions, and so on, and each benchmark focuses on testing specific instruction features. For example, mm focuses on multi-threaded memory reads and writes operations. Since a BOOM design needs to handle various applications rather than specific instruction categories, we average the clock cycles and power values from these benchmarks to denote the design’s performance and power. After warming up the design by executing the benchmark, we proceed to measure the performance and power for a specific BOOM configuration. This process allows us to obtain accurate performance and power values for the given microarchitecture, taking into account any initial variability or transient effects that may occur during the warm-up phase.

Several representative baselines are compared with the diversity-guided BOOM-Explorer. The ANN-based method [18] (shorted as ASPLOS’06) stacks ANN as the performance model for a multiprocessor to conduct DSE. The regression-based method [27] (termed HPCA’07) leverages regression models with non-linear transformations to explore the performance-power Pareto frontier for POWER microprocessors. The AdaBoost-RT-based method [21] (abbreviated as DAC’16) utilizes the orthogonal design sampling [49] and active learning-based AdaBoost regression tree models [28] to explore an Alpha21264-like microprocessor [8, 11]. The arts mentioned above proved effective in exploring microarchitecture parameters in their works, respectively. Therefore, it is requisite to compare these methodologies with our proposed methodology. The

high-level synthesis (HLS) predictive model-based method [45] (named DAC'19) exploring the HLS design is also chosen as our baseline. Although the starting point is different, their method proved robust and transferable. We also compare diversity-guided BOOM-Explorer with traditional machine learning models, including **support vector regression (SVR)** and tree-based approaches such as random forest, XGBoost [50], and tree-structured Parzen estimator approach (abbreviated as NIPS'11) [44]. In addition, we compare previous state-of-the-art Bayesian optimization approaches [46, 47] with our proposed methodology, since both methods adopt the same optimization framework but are distinct in the design of initialization, surrogate model, and acquisition function. We name the two state-of-the-art approaches as NIPS'19 and NIPS'22, respectively. For fair comparisons, the experimental settings of the baselines are the same as those mentioned in their papers. In traditional machine learning algorithms (SVR, random forest, and XGBoost), since they are not fit for Bayesian optimization due to unavailable predictive uncertainties, we leverage simulated annealing [51] to explore the design space. We also compare the proposed methodology with the sequential optimization version of BOOM-Explorer [48]. All algorithms explore the same design space, as defined in Table 1. In our future work, we will delve into more intricate design spaces that encompass branch prediction algorithms, prefetching, and additional objectives, such as area metrics. However, one caveat is that there is a resemblance between microarchitecture and area values, akin to power values, where a larger area usually corresponds to higher power dissipation. Meeting higher performance requirements often entails employing more hardware resources, which, in turn, leads to a larger area.

To compare the diversity-guided BOOM-Explorer with all baselines, we utilize multiple metrics. These metrics include the Pareto hypervolume, the **average distance to the reference set (ADRS)**, and the **overall running time (ORT)**. Pareto hypervolume and ADRS are two widely used metrics in estimating the performance of DSE among multiple objectives. As mentioned in Section 4.4, the Pareto hypervolume measures the volume of the space enclosed by all solutions on the explored Pareto frontier and a user-defined reference point, where the computation of Pareto hypervolume is defined in Equation (11). The ADRS computes the average distance between the predicted Pareto frontier and the real Pareto frontier, providing insights into the quality and proximity of the solutions to the golden results. Equation (16) lists the computation of ADRS.

$$\text{ADRS}(\Gamma, \Omega) = \frac{1}{|\Gamma|} \sum_{\gamma \in \Gamma} \min_{\omega \in \Omega} f(\gamma, \omega), \quad (16)$$

where f is the Euclidean distance function, Γ is the real Pareto frontier, and Ω is the predicted Pareto frontier. The ORT measures the total running time of algorithms, including initialization, exploring, and the VLSI runtime cost. The higher the Pareto hypervolume, the lower the ADRS and ORT, the better the DSE algorithm is.

5.3 Evaluation Results

Figure 9 shows the predicted Pareto frontier obtained by the baselines and diversity-guided BOOM-Explorer. The results show that the Pareto frontier generated by BOOM-Explorer and its diversity-guided version is much closer to the actual Pareto frontier in general. Moreover, the diversity-guided BOOM-Explorer improves the results visually.

The normalized Pareto hypervolume, ADRS, and ORT results are listed in Table 3. Three summaries can be drawn from Table 3. First, BOOM-Explorer achieves an average of 18.75% higher Pareto hypervolume, 35.47% less ADRS, and 65.38% less ORT compared to all baselines. Specifically, BOOM-Explorer outperforms ASPLOS'06, HPCA'07, DAC'16, and DAC'19 by 70.13%, 66.55%, 28.65%, and 64.54% in ADRS, respectively. Meanwhile, it accelerates the exploration by more than 88.19% compared with DAC'16. The improvement achieved by our method over

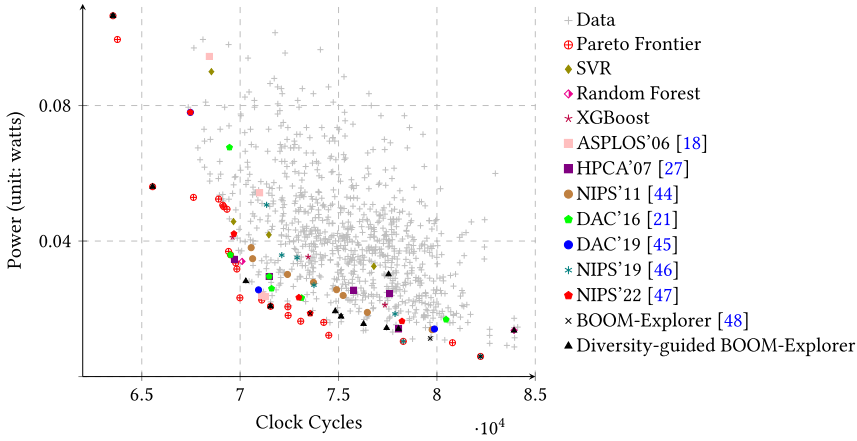


Fig. 9. Comparisons between the predicted Pareto frontier and the real Pareto frontier of BOOM microarchitectures.

Table 3. Normalized Experimental Results for Pareto Hypervolume, ADRS & ORT

Methodologies	Norm. Pareto Hypervolume		Norm. ADRS		Norm. ORT
	Val.	Ratio	Val.	Ratio	
SVR	1.1519	1.0000×	0.2400	1.0000×	1.0000×
Random Forest	1.1794	1.0238×	0.2263	0.9430×	0.9763×
XGBoost	1.3152	1.1417×	0.2171	0.9046×	1.0102×
ASPLOS'06 [18]	1.3266	1.1516×	0.1948	0.8116×	0.9436×
HPCA'07 [27]	1.3218	1.1475×	0.1907	0.7949×	0.8544×
NIPS'11 [44]	1.3547	1.1760×	0.1723	0.7181×	0.7506×
DAC'16 [21]	1.3886	1.2055×	0.1473	0.6141×	3.0102×
DAC'19 [45]	1.3395	1.1628×	0.1884	0.7852×	0.8973×
NIPS'19 [46]	1.5496	1.3452×	0.1178	0.4908×	0.3567×
NIPS'22 [47]	1.5625	1.3564×	0.1426	0.5944×	0.4436×
BOOM-Explorer w/o MicroAL [48]	1.4665	1.2731×	0.1441	0.6006×	0.3307×
BOOM-Explorer [48]	1.6280	1.4132×	0.1145	0.4773×	0.3555×
Diversity-guided BOOM-Explorer	1.6362	1.4203×	0.0915	0.3815×	0.3533×

previous approaches stems from a customized algorithm design explicitly tailored to the problem at hand. Our method asks for fewer estimations using the VLSI flow while effectively modeling the design space with as few samples as possible. Second, MicroAL contributes 11.00% and 20.53% to exalt the Pareto hypervolume and ADRS. The results are obtained from the comparison between BOOM-Explorer and BOOM-Explorer w/o MicroAL. It also illustrates that without MicroAL, the performance of BOOM-Explorer would be close to DAC'16. If more time budget is allowed, then we expect better results received from BOOM-Explorer. Third, incorporating diversity guidance as a key enhancement, we achieve a significant improvement in the ADRS, surpassing BOOM-Explorer by 20.09% with comparable ORT. The improvement on ADRS is larger than the Pareto hypervolume, demonstrating that “outliers” can be explored. As discussed in Section 4.5, these outliers pertain to similar or smaller EIPV but closer to the real Pareto frontier. It is worth noting that we partition the design space w.r.t. IntPhyRegister, as discussed in Section 4.5. Hence, the proposed method has a better effect when we target to optimize for integer-intensive applications. Additionally, previous state-of-the-art approaches (NIPS'19 and NIPS'22) receive a relatively good Pareto

Table 4. Comparison with Two-wide BOOM

Methodology	Microarchitecture Embedding ⁺	Average Clock Cycles	Average Power (watts)
Two-wide BOOM [12–14]	[4, 16, 32, 12, 4, 8, 2, 2, 64, 80, 64, 1, 2, 1, 16, 16, 4, 2, 8]	74915.2963	6.0700×10^{-2}
BOOM-Explorer [48]	[4, 16, 16, 8, 2, 8, 2, 2, 32, 64, 64, 1, 3, 1, 24, 24, 8, 4, 8]	73333.7407	5.8600×10^{-2}
Diversity-guided BOOM-Explorer	[4, 16, 16, 8, 4, 8, 2, 2, 32, 64, 64, 1, 2, 1, 24, 24, 8, 4, 8]	73279.2820	5.8200×10^{-2}

⁺ The parameters are in the same order as Table 1.

hypervolume and smaller ADRS more efficiently than other baselines. Like random forest and XGBoost, the NIPS’11 baseline [44] adopts tree-based structures as the surrogate model but uses Bayesian optimization. The NIPS’19 [46] and NIPS’22 baselines [47] leverage information-theoretic acquisition function designs. Although, these baselines perform well in general DSE problems, they achieve mediocre results compared to diversity-guided BOOM-Explorer largely due to the missing customization in the algorithm design such as tightly coupled with expert knowledge.

5.4 Comparison of Pareto-optimal BOOM Microarchitectures

We compare Pareto-optimal microarchitectures explored by proposed algorithms and human implementations [12–14] on more benchmarks to study how each BOOM microarchitecture balance the performance and power. The Pareto-optimal microarchitectures have similar parameter settings, as listed in Table 4.

Pareto-optimal designs found by BOOM-Explorer and diversity-guided BOOM-Explorer have the same decode width as the two-wide BOOM. However, the Pareto-optimal design reduces hardware components on the branch predictor (i.e., RasEntry, BranchCount, etc.), entries of the reorder buffer, and so on, but enlarges instructions issue **width**, **load queue (LDQ)**, store queue (STQ), and so on. Moreover, it has different cache organizations, e.g., different associate sets. Because LSU introduced in Section 2.1 tends to become a bottleneck of the microarchitecture, the Pareto-optimal design increases hardware resources for LDQ and STQ, increasing associate sets and **miss status handling register (MSHR)** entries for D-cache to overcome more data conflicts. Furthermore, the Pareto-optimal design found by diversity-guided BOOM-Explorer reduces the resources of **return address stack (RAS)** and **branch target buffer (BTB)**, since they affect the tradeoff less for the same branch prediction algorithm [30]. It also reduces the instruction issue slot but increases the I-cache associative sets. Both Pareto-optimal designs achieve a better tradeoff on power and performance by reducing redundant hardware resources while increasing necessary components on critical paths [52].

We evaluate these microarchitectures with more benchmarks. Table 4 shows the average clock cycles and power values for all these benchmarks. These benchmarks are chosen from different application scenarios, e.g., add-int, add-fp, and so on, are from ISA basic instructions, iir, firdim, and so on, are from DSP-oriented algorithms [53], compress, duff, and so on, are from real-time computing applications [54], and so on. Figure 10 shows the comparison of performance and power values, respectively. For all of these benchmarks, BOOM-Explorer’s design runs approximately 2.11% faster and, at the same time, dissipates 3.45% less power than the two-wide BOOM. The solution from diversity-guided BOOM-Explorer achieves 2.18% faster and 3.99% better on power dissipation than human implementations.

5.5 Effectiveness of MicroAL

To assess the effectiveness of MicroAL, we conduct an ablation study on the effectiveness of MicroAL. We investigate the integration of MicroAL to some baselines, which allows us to modify the initialization algorithm that is not tightly coupled with the exploration procedure. Specifically,

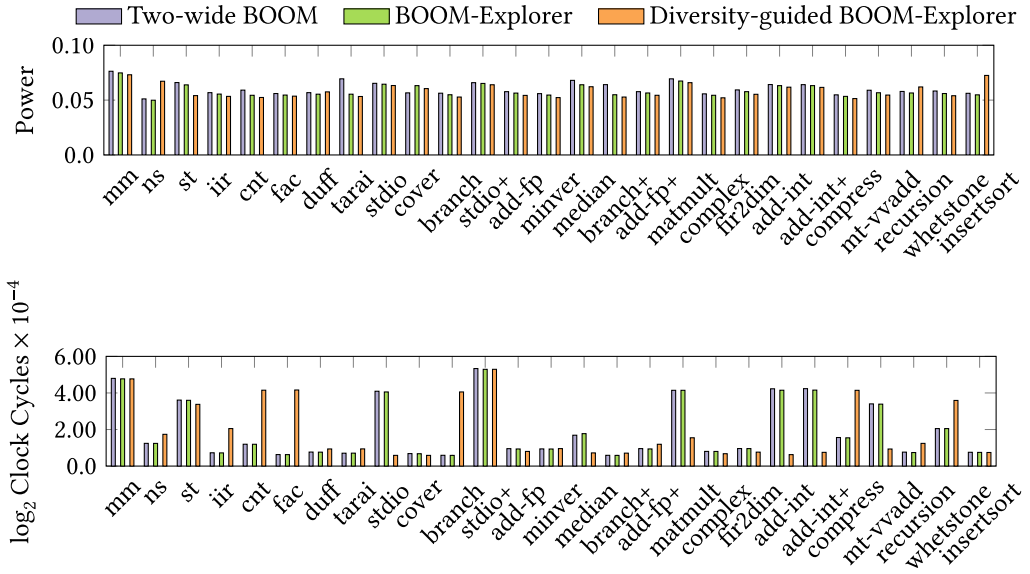


Fig. 10. Comparisons of explored BOOM microarchitectures with more benchmarks.

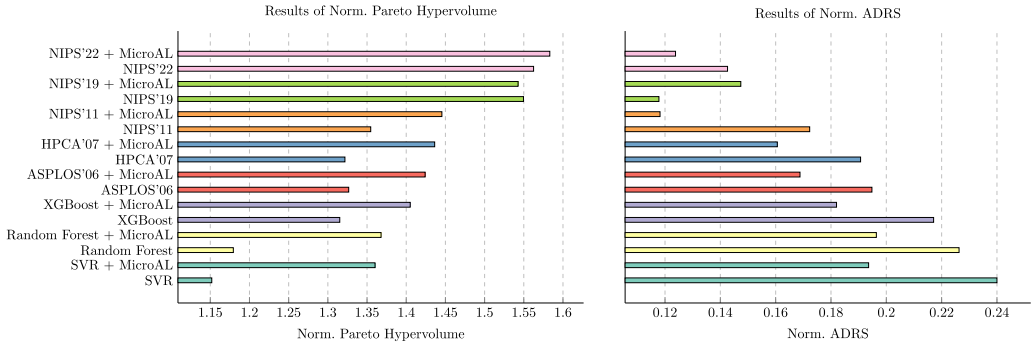


Fig. 11. We integrate MicroAL into chosen baselines to investigate the MicroAL's effectiveness on the normalized Pareto hypervolume and ADRS.

we conduct MicroAL with SVR, random forest, XGBoost, ASPLOS'06 [18], HPCA'07 [27], NIPS'11 [44], NIPS19 [46], and NIPS'22 [47].

Figure 11 lists the comparative results on the Pareto hypervolume and ADRS when we integrate MicroAL to baselines. Two summaries are drawn from Figure 11. First, integrating MicroAL into baselines can improve an average of 8.06% Pareto hypervolume, and 12.15% ADRS. For example, when we integrate MicroAL into SVR, the Pareto hypervolume can be increased by approximately 18.08% while the ADRS is reduced by a large margin. Second, MicroAL can have negative effects on some benchmarks like NIPS'19 [46]. Although integrating MicroAL to NIPS'19 [46] results in a similar Pareto hypervolume, it degrades the ADRS by 25.16%. We argue that the reason behind the phenomenon is that we leverage single-task GP models to individually model the performance and power. In contrast, the samples generated by MicroAL are highly different. Therefore, the built surrogate models are rather “weak.” However, objective-related features can be utilized to capture

the distinctions between MicroAL's samples. So, a good co-design between the initialization and the surrogate model is significant.

6 CONCLUSIONS

In this article, BOOM-Explorer is proposed to explore the Pareto optimality within the BOOM microarchitecture design space efficiently. Several techniques, including MicroAL, DKL-GP, EIPV, and diversity-guided parallel exploration, are proposed in diversity-guided BOOM-Explorer, following two lessons learned from prior arts. Experimental results with RISC-V Berkeley-Out-of-Order Machine under 7-nm technology show that our proposed methodology achieves an average of 18.75% higher Pareto hypervolume, 35.47% less average distance to reference set, and 65.38% less overall running time compared to previous approaches. Adapting BOOM-Explorer for other microprocessors requires mild changes, including additional expert knowledge and the VLSI flow support. We hope to see more research emerging in our community to improve microarchitecture design space explorations of RISC-V microprocessors.

REFERENCES

- [1] RISC-V. 2023. Wikipedia, Wikimedia Foundation. <https://en.wikipedia.org/wiki/RISC-V>
- [2] Philippe Magarshack and Pierre G. Paulin. 2003. System-on-chip beyond the nanometer wall. In *ACM/IEEE Design Automation Conference (DAC'03)*. IEEE, 419–424.
- [3] Yunsup Lee, Andrew Waterman, Henry Cook, Brian Zimmer, Ben Keller, Alberto Puggelli, Jaehwa Kwak, Ruzica Jevtic, Stevo Bailey, Milovan Blagojevic, Pi-Feng Chiu, Rimas Avizienis, Brian Richards, Jonathan Bachrach, David Patterson, Elad Alon, Borivoje Nikolic, and Krste Asanovic. 2016. An agile approach to building RISC-V microprocessors. *IEEE Micro* 36, 2 (2016), 8–20.
- [4] James E. Smith. 1998. A study of branch prediction strategies. In *IEEE/ACM International Symposium on Computer Architecture (ISCA'98)*. 202–215.
- [5] George Z. Chrysos and Joel S. Emer. 1998. Memory dependence prediction using store sets. In *IEEE/ACM International Symposium on Computer Architecture (ISCA'98)*. 142–153.
- [6] Sparsh Mittal. 2016. A survey of recent prefetching techniques for processor caches. *Comput. Surv.* 49, 2 (2016), 1–35.
- [7] Todd Austin, Eric Larson, and Dan Ernst. 2002. SimpleScalar: An infrastructure for computer system modeling. *IEEE Trans. Comput.* 35, 2 (2002), 59–67.
- [8] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The GEM5 simulator. *ACM SIGARCH Comput. Archit. News* 2 (2011), 1–7.
- [9] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *IEEE International Conference on High Performance Computing (HiPC'11)*. 1–12.
- [10] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. *ACM SIGARCH Comput. Archit. News* 41, 3 (2013), 475–486.
- [11] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, Gabe Black, Gedare Bloom, Bobby R. Bruce, Daniel Rodrigues Carvalho, Jeronimo Castrillon, Lizhong Chen, Nicolas Derumigny, Stephan Diestelhorst, Wendy Elsasser, Carlos Escuin, Marjan Fariborz, Amin Farmahini-Farahani, Pouya Fotouhi, Ryan Gambord, Jayneel Gandhi, Dibakar Gope, Thomas Grass, Anthony Gutierrez, Bagus Hanindhito, Andreas Hansson, Swapnil Haria, Austin Harris, Timothy Hayes, Adrian Herrera, Matthew Horsnell, Syed Ali Raza Jafri, Radhika Jagtap, Hanhwi Jang, Reiley Jeyapaul, Timothy M. Jones, Matthias Jung, Subash Kannoth, Hamidreza Khaleghzadeh, Yuetsu Kodama, Tushar Krishna, Tommaso Marinelli, Christian Menard, Andrea Mondelli, Miquel Moreto, Tiago Mück, Omar Naji, Krishnendra Nathella, Hoa Nguyen, Nikos Nikoleris, Lena E. Olson, Marc Orr, Binh Pham, Pablo Prieto, Trivikram Reddy, Alec Roelke, Mahyar Samani, Andreas Sandberg, Javier Setoain, Boris Shingarov, Matthew D. Sinclair, Tuan Ta, Rahul Thakur, Giacomo Travaglini, Michael Upton, Nilay Vaish, Ilias Vougioukas, William Wang, Zhengrong Wang, Norbert Wehn, Christian Weis, David A. Wood, Hongil Yoon, and Éder F. Zulian. 2020. The GEM5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152* (2020).
- [12] Krste Asanovic, David A. Patterson, and Christopher Celio. 2015. *The Berkeley Out-of-order Machine (BOOM): An Industry-competitive, Synthesizable, Parameterized RISC-V Processor*. Technical Report. University of California at Berkeley.

- [13] Christopher Patrick Celio. 2017. *A Highly Productive Implementation of an Out-of-Order Processor Generator*. eScholarship, University of California.
- [14] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. 2020. SonicBOOM: The 3rd generation Berkeley Out-of-Order Machine. In *Workshop on Computer Architecture Research with RISC-V (CARRV'20)*.
- [15] RISC-V Unit Tests Benchmark Suites. 2023. GitHub. <https://github.com/riscv-software-src/riscv-tests>
- [16] Vinod Kathail, Shail Aditya, Robert Schreiber, B. Ramakrishna Rau, Darren C. Cronquist, and Mukund Sivaraman. 2002. PICO: Automatically designing custom computers. *IEEE Trans. Comput.* 35, 9 (2002), 39–47.
- [17] David Brooks, Pradip Bose, Viji Srinivasan, Michael K. Gschwind, Philip G. Emma, and Michael G. Rosenfield. 2003. New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors. *IBM J. Res. Devel.* 47, 5.6 (2003), 653–670.
- [18] Engin İpek, Sally A. McKee, Rich Caruana, Bronis R. de Supinski, and Martin Schulz. 2006. Efficiently exploring architectural design spaces via predictive modeling. *ACM Int. Conf. Archit. Supp. Program. Lang. Oper. Syst.* 40, 5 (2006), 195–206.
- [19] Tejas S. Karkhanis and James E. Smith. 2007. Automated design of application specific superscalar processors: An analytical approach. In *IEEE/ACM International Symposium on Computer Architecture (ISCA'07)*. 402–411.
- [20] Christophe Dubach, Timothy Jones, and Michael O'Boyle. 2007. Microarchitectural design space exploration using an architecture-centric approach. In *IEEE/ACM International Symposium on Microarchitecture (MICRO'07)*. IEEE, 262–271.
- [21] Dandan Li, Shuzhen Yao, Yu-Hang Liu, Senzhang Wang, and Xian-He Sun. 2016. Efficient design space exploration via statistical sampling and AdaBoost learning. In *ACM/IEEE Design Automation Conference (DAC'16)*. 1–6.
- [22] Hossein Golestani, Rathijit Sen, Vinson Young, and Gagan Gupta. 2022. Calipers: A criticality-aware framework for modeling processor performance. In *ACM International Conference on Supercomputing (ICS'22)*.
- [23] Brian A. Fields, Rastislav Bodik, Mark D. Hill, and Chris J. Newburn. 2003. Using interaction costs for microarchitectural bottleneck analysis. In *IEEE/ACM International Symposium on Microarchitecture (MICRO'03)*. IEEE, 228–239.
- [24] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E. Smith. 2009. A mechanistic performance model for superscalar out-of-order processors. *IEEE Trans. Comput.* 27, 2 (2009), 1–37.
- [25] Chen Bai, Jiayi Huang, Xuechao Wei, Yuzhe Ma, Sicheng Li, Hongzhong Zheng, Bei Yu, and Yuan Xie. 2023. ArchExplorer: Microarchitecture exploration via bottleneck analysis. In *IEEE/ACM International Symposium on Microarchitecture (MICRO'23)*. IEEE/ACM.
- [26] Richard E. Kessler. 1999. The Alpha 21264 microprocessor. *IEEE Micro* 19, 2 (1999), 24–36.
- [27] Benjamin C. Lee and David M. Brooks. 2007. Illustrative design space studies with microarchitectural regression models. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'07)*. 340–351.
- [28] Durga L. Shrestha and Dimitri P. Solomatine. 2006. Experiments with AdaBoost.RT, an improved boosting scheme for regression. *Neural Comput.* 18, 7 (2006), 1678–1710.
- [29] Vinay Vashishtha, Manoj Vangala, and Lawrence T. Clark. 2017. ASAP7 predictive design kit development and cell design technology co-optimization. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'17)*. 992–998.
- [30] André Seznec and Pierre Michaud. 2006. A case for (partially) TAgged GEometric history length branch prediction. *J. Instruct.-level Parallel.* 8 (2006), 23.
- [31] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avizienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: Constructing hardware in a Scala embedded language. In *ACM/IEEE Design Automation Conference (DAC'12)*. 1212–1221.
- [32] Shuhan Zhang, Fan Yang, Dian Zhou, and Xuan Zeng. 2020. An efficient asynchronous batch Bayesian optimization approach for analog circuit synthesis. In *ACM/IEEE Design Automation Conference (DAC'20)*. 1–6.
- [33] Qi Sun, Tinghuan Chen, Siting Liu, Jin Miao, Jianli Chen, Hao Yu, and Bei Yu. 2021. Correlated multi-objective multi-fidelity optimization for HLS directives design. In *IEEE/ACM Proceedings Design, Automation and Test in Europe (DATE'21)*.
- [34] Carl Edward Rasmussen. 2003. Gaussian processes in machine learning. In *Summer School on Machine Learning*. Springer, 63–71.
- [35] Kai Yu, Jinbo Bi, and Volker Tresp. 2006. Active learning via transductive experimental design. In *International Conference on Machine Learning (ICML'06)*. 1081–1088.
- [36] Yuzhe Ma, Subhendu Roy, Jin Miao, Jiamin Chen, and Bei Yu. 2018. Cross-layer optimization for high speed adders: A pareto driven machine learning approach. *IEEE Trans. Comput.-aid. De. Integ. Circ. Syst.* 38, 12 (2018), 2298–2311.
- [37] Yuzhe Ma, Ziyang Yu, and Bei Yu. 2019. CAD tool design space exploration via Bayesian optimization. In *ACM/IEEE Workshop on Machine Learning CAD (MLCAD'19)*. 1–6.
- [38] Chris Williams, Edwin V. Bonilla, and Kian M. Chai. 2007. Multi-task Gaussian process prediction. In *Annual Conference on Neural Information Processing Systems (NIPS'07)*. 153–160.

- [39] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. 2016. Deep kernel learning. In *Artificial Intelligence and Statistics*. PMLR, 370–378.
- [40] Amar Shah and Zoubin Ghahramani. 2016. Pareto frontier learning with expensive correlated objectives. In *International Conference on Machine Learning (ICML'16)*. 1919–1927.
- [41] Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning*. Vol. 135. MIT Press, Cambridge, MA.
- [42] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Yakun Sophia Shao, Krste Asanovic, and Borivoje Nikolic. 2020. Chipyard: Integrated design, simulation, and implementation framework for custom SoCs. *IEEE Micro* 40, 4 (2020), 10–21.
- [43] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- [44] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Annual Conference on Neural Information Processing Systems (NIPS'11)*.
- [45] Shuangnan Liu, Francis C. M. Lau, and Benjamin Carrion Schafer. 2019. Accelerating FPGA prototyping through predictive model-based HLS design space exploration. In *ACM/IEEE Design Automation Conference (DAC'19)*. 1–6.
- [46] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. 2019. Max-value entropy search for multi-objective Bayesian optimization. In *Annual Conference on Neural Information Processing Systems (NIPS'19)*.
- [47] Ben Tu, Axel Gandy, Nikolas Kantas, and Behrang Shafei. 2022. Joint entropy search for multi-objective Bayesian optimization. In *Annual Conference on Neural Information Processing Systems (NIPS'22)*. 9922–9938.
- [48] Chen Bai, Qi Sun, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin D. F. Wong. 2021. BOOM-Explorer: RISC-V BOOM microarchitecture design space exploration framework. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'21)*. IEEE, 1–9.
- [49] Kai-Tai Fang and Yuan Wang. 1993. *Number-theoretic Methods in Statistics*. Vol. 51. CRC Press.
- [50] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD'16)*. 785–794.
- [51] Peter J. M. Van Laarhoven and Emile H. L. Aarts. 1987. Simulated annealing. In *Simulated Annealing: Theory and Applications*. Springer, 7–15.
- [52] Brian Fields, Shai Rubin, and Rastislav Bodik. 2001. Focusing processor policies via critical-path prediction. In *IEEE/ACM International Symposium on Computer Architecture (ISCA'01)*. IEEE, 74–85.
- [53] V. Zivojnovic, J. Martinez, C. Schläger, and Heinrich Meyr. 1994. DSPstone: A DSP-Oriented benchmarking methodology. In *International Conference on Signal Processing, Applications & Technology*.
- [54] Metin Kuzhan and Veysel Harun Şahin. 2020. MBBench: A WCET benchmark suite. *Sakarya Univ. J. Comput. Inf. Sci.* 3, 1 (2020), 40–50.

Received 31 May 2023; revised 15 September 2023; accepted 8 October 2023