Towards Automated RISC-V Microarchitecture Design with Reinforcement Learning

Chen Bai[1], **Jianwang Zhai**[2✉], **Yuzhe Ma**[3], **Bei Yu**[1✉], **Martin D.F. Wong**[4]

[1]The Chinese University of Hong Kong
[2]Beijing University of Posts and Telecommunications
[3]The Hong Kong University of Science and Technology (Guangzhou)
[4]Hong Kong Baptist University

{cbai, byu}@cse.cuhk.edu.hk, zhaijw@bupt.edu.cn, yuzhema@hkust-gz.edu.cn, mdfwong@hkbu.edu.hk

Abstract

Microarchitecture determines the implementation of a microprocessor. Designing a microarchitecture to achieve better performance, power, and area (PPA) trade-off has been increasingly difficult. Previous data-driven methodologies hold inappropriate assumptions and lack more tightly coupling with expert knowledge. This paper proposes a novel reinforcement learning-based (RL) solution that addresses these limitations. With the integration of microarchitecture scaling graph, PPA preference space embedding, and proposed lightweight environment in RL, experiments using commercial electronic design automation (EDA) tools show that our method achieves an average PPA trade-off improvement of 16.03% than previous state-of-the-art approaches with $4.07\times$ higher efficiency. The solution qualities outperform human implementations by at most $2.03\times$ in the PPA trade-off.

Introduction

The instruction set architecture (ISA) is the interface between software and hardware. RISC-V, an open standard ISA, has garnered significant attention from both academia and industry nowadays (Mis 2023e). The microarchitecture, also known as computer organization, determines how a particular microprocessor is implemented given an ISA. It sets the cornerstone for a microprocessor's overarching design points: performance, power, and area (PPA).

Nevertheless, it is challenging to design a microarchitecture efficiently to achieve pre-determined PPA design goals for target workloads (computation-bound or memory-bound programs) with manual efforts. Computer architects often rely on design space exploration (DSE) to find appropriate solutions. Those solutions can maximize performance and minimize power and area for target workloads. DSE is an iterative, automated, and trial-and-error procedure, and non-trivial due to two factors. First, the design space is enormous and complicated. It comes from the high complexity of a microarchitecture, as shown in Figure 1, which includes different *components* responsible for implementing specific functions (Chen et al. 2020; Grayson et al. 2020). Second,

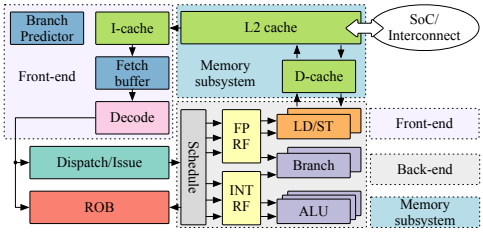

Figure 1: An overview of the example microarchitecture. The instructions fetched from I-cache are sent to functional units (e.g., ALU, LD/ST, etc.) for execution. Register files (RF) save the operands and results. Reorder buffer (ROB) achieves the precise interrupts (Smith and Pleszkun 1988). Components are highlighted with diverse colors and the same color denotes a similar function.

evaluating the PPA values of a single microarchitecture design requires an extremely high runtime. For example, cycle-accurate simulators or EDA tools are often leveraged in the evaluation, resulting in several days, even weeks to get PPA values for one design. Thus, it is a fond dream for architects to iterate the design space and retrieve optimal solutions.

Previous methodologies have been proposed. In industry, architects' expert knowledge is a heuristic to guide the DSE. However, it is a concern that personal bias can lead to sub-optimal solutions. In academia, both analytical and data-driven methods have been proposed. The analytical methods conduct interpretable equations to describe relations between microarchitectures and PPA values for various workloads. Karkhanis and Smith (2007) adopted interval analysis to construct such equations. However, the analytical model requires much expert knowledge, and is unscalable for newly-emerged microprocessors. Data-driven methods are utilized accordingly when we lack accesses to experts. The microarchitecture is viewed as a black box. Chen et al. (2014) employed a ranking model. Li et al. (2016) applied statistical sampling and AdaBoost learning. Bai et al. (2021; 2023b) proposed a Bayesian optimization-based framework. Such data-driven methods generally outperform analytical methods owing to many advanced machine-learning techniques (Yu et al. 2023). However, they are not free of criticism. Blindly exploring microarchitectures (purely driven by the algorithm rather than tightly coupled with expertise) can be naive since architects already know the characteris-

✉Corresponding Authors

tics of most designs (Bai et al. 2023a).

In this paper, we follow the approach of previous data-driven methods but with key distinctions: our method removes prior unrealistic assumptions, and our solution is deeply integrated with expert knowledge. Previous data-driven methods often assume a positive correlation between the PPA difference and feature embeddings of microarchitectures. On the contrary, the assumption does not hold in general. Our RL solution is free of such assumptions. Moreover, using the *microarchitecture scaling graph*, we tightly embed the expert knowledge to formulate the Markov decision process (MDP). The scaling graph encodes the sequential decision precedences of the microarchitecture components. Accordingly, we propose a multi-objective RL framework based on the MDP. The framework enables the automated RISC-V microarchitecture design with a single agent for different PPA design preferences. It is worth noting that our solution focuses on RISC-V due to the forecast that RISC-V could motivate competitive commercial products over x86, ARM, etc., for many applications in the future (Mis 2022). Our main contributions are as follows:

1) We propose an MDP model with the microarchitecture scaling graph, embracing architects' expertise and providing strong prior knowledge for our agent.

2) We embed the PPA design preferences into RL and re-formulate the multi-objective optimization to a unified dynamic-weighted reward signal. It is helpful since this feature allows agents to explore microarchitectures for different PPA design preferences online.

3) We propose a lightweight environment to accelerate the learning process. With calibrated PPA models, we accelerate the learning process by over $100\times$ times compared to using EDA tools only (Zhai et al. 2021).

4) Our experiments use representative RISC-V microprocessors and evaluate with commercial EDA tools at 7-nm technology. Results show that our method can achieve an average of 16.03% PPA trade-off improvement over prior state-of-the-art approaches with $4.07\times$ higher efficiency. And the solution qualities outperform human implementations by at most $2.03\times$ in the PPA trade-off.

Preliminary

RISC-V Microprocessors. Unlike other ISAs (ARM, x86, etc.), RISC-V is free for commercial usage. The free license drives the appearance of many RISC-V microprocessors, some of which are representatives. Rocket (Asanović et al. 2016) is a six-stage pipeline in-order microprocessor. SonicBOOM (Zhao et al. 2020) is a ten-stage pipeline out-of-order design. XiangShan (Xu et al. 2022) features advanced microarchitecture optimizations. Xuantie-910 (Chen et al. 2020) is an open-source implementation from the industry.

Microarchitecture PPA Modeling. Computer architects use various tools to evaluate PPA values of microarchitecture designs. When the register-transfer-level design (RTL) [1] is available, EDA tools are necessary to report PPA values. The PPA evaluation flow with EDA tools (Mis 2023b) includes steps like logic synthesis, placement and routing, netlist simulation, power analysis, etc. When the RTL implementation is unavailable, pre-RTL simulation infrastructures like microprocessor performance simulators are used to report first-hand PPA values. Compared to EDA tools, pre-RTL simulation infrastructures are less accurate.

[1]Register-transfer level design (RTL) is a description of hardware implementations using programming languages such as Verilog and VHDL (Mis 2023c).

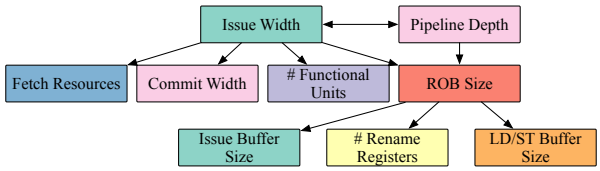

Figure 2: An overview of the microarchitecture scaling graph. Colors are matched with Figure 1.

In this paper, we propose a lightweight RL environment to couple the pre-RTL simulation infrastructures with EDA tools, i.e., improve the modeling accuracy of pre-RTL simulation infrastructures without sacrificing efficiency. Specifically, we leverage GEM5 (Binkert et al. 2011), a performance simulator, and McPAT (Li et al. 2009) as our fundamental PPA modeling tools in the RL environment.

Microarchitecture Scaling Graph. Since the microarchitecture scaling graph first appeared (Eyerman et al. 2009), architects relied on it to investigate the mechanistic microexecutions, including simulator implementation (Carlson, Heirman, and Eeckhout 2011), analytical performance model (Carlson et al. 2014), etc.

The microarchitecture scaling graph is directed, elucidating the scaling precedence constraints between components, as shown in Figure 2. Nodes are components, and directed edges are scaling precedences. According to Figure 2, an interplay exists between the pipeline width and issue width. The pipeline width and issue width determine the ROB size, while the ROB size decides the issue buffer size, load/store (LD/ST) buffer size, etc. The scaling graph is derived from extensive simulations and architects' discussions. The relations unfolded by the scaling graph are general for mainstream microarchitectures due to a widely applied typical von Neumann architecture.

Problem Formulation. Given the microarchitecture design space, the problem is to find the solution to Equation (1) within a limited time budget.

$$\underset{\boldsymbol{s}\in\mathbb{D}^n}{max}[\text{Perf}(\boldsymbol{s}), -\text{Power}(\boldsymbol{s}), -\text{Area}(\boldsymbol{s})], \tag{1}$$

where $\mathbb{D}$ is an n-dimensional microarchitecture design space, $\boldsymbol{s}$ is a vector to parameterize a design (feature embedding of a microarchitecture), Perf, Power, and Area are PPA values, respectively.

Methodology

Overview

We propose an RL solution framework with customized MDP (S, A, P, R) formulation, as shown in Figure 3. The state space S is the design space. The action space A is the candidate set of components' types or corresponding hardware resources listed in Table 1. The components' types

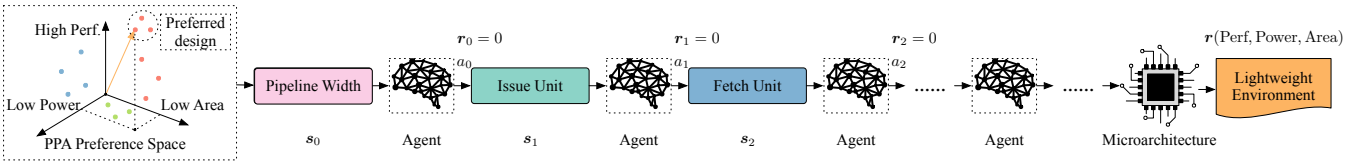

Figure 3: Overview of our RL framework. $\boldsymbol{s}$ denotes a state, a is an action, and $\boldsymbol{r}$ represents an immediate vectorized reward.

refer to a type of branch predictor, cache replacement policy, etc., and hardware resources specify the queue, buffer, or stack sizes. P is the state transition. The reward space R involves all the vectorized PPA values. In the training, the agent learns to generate appropriate *partial* components stepwise given sampled PPA preference vectors to formulate the complete microarchitecture. In the DSE, the trained agent produces solutions w.r.t. a fixed PPA preference specified by architects.

The PPA preference space is incorporated into our framework since the microarchitecture design is faced with diverse workloads. High-performance computing scenarios emphasize performance more, while embedded applications push pressure on high power efficiency and area efficiency. Different PPA preference vectors denote architects' design goals, and our single agent benefits from the PPA preference-aware DSE with the proposed framework.

Combine RL w. Microarchitecture Scaling Graph

We combine RL with the microarchitecture scaling graph via a practical episode design. In each step of an episode, the agent produces partial components. The episode ends until a complete microarchitecture is formulated.

The state of a microarchitecture is encoded as a vector with each element denoting a selection of a particular component parameter. Elements are masked for undefined components and masks are removed progressively as more components are determined by advancing the step. The action space is correlated with the step since different components relate to distinct action candidates. Each step determines one component. Once the complete microarchitecture is generated, we adopt the lightweight environment to evaluate the reward $\boldsymbol{r}(\text{Perf}, \text{Power}, \text{Area})$. Otherwise, the reward is zero. The precedence of decision-making among components follows the scaling graph, as the graph unveils cause-and-effect relationships between different components. Another noteworthy point is that the action space is changed in each step, leading to the output misalignment for a single agent. We apply a typical engineering trick: the normalization of action probability to deal with it (Schrittwieser et al. 2020).

The rationale for the episode design is that we place strong prior knowledge for the agent. The prior knowledge is derived from the scaling graph, revealing the components' decision priority. The priority unfolds the significance of a component's influence on PPA values. For example, the pipeline width determines the maximal instructions fetched simultaneously. And the structure of the issue unit is then decided based on the width. Because the issue unit adjusts instruction issue rates based on how many instructions are fetched by the front-end and buffers those instructions for the back-end (see Figure 1). Hence, our episode design explicitly provides such domain knowledge for the agent.

As shown in Figure 3, once a PPA preference and the pipeline width are specified, the agent first determines the appropriate issue queue sizes, with the following determinations involving fetch queue size, type of a branch predictor, etc., sequentially. Although relations between some components are not uncovered from the scaling graph (e.g., the issue buffer size and the number of physical registers indicated in Figure 2), we determine their structures in a fixed order within an episode.

Dynamic-weighted Reward

Since a single agent cannot handle multiple objectives simultaneously, a weighted summation is applied in the reward computation, as listed in Equation (2).

$$r = \boldsymbol{r}(\text{Perf}, \text{Power}, \text{Area}) \cdot (\alpha, \beta, \gamma)^\top \tag{2}$$

where α, β, and γ are weights controlling the PPA trade-off. We align the reward optimization with our objectives via normalizing Perf, Power, and Area, i.e., maximizing the reward equals maximizing the performance, and minimizing the power and area.

However, weights can be changed as architects' PPA design goals vary. A transparent limitation is that the agent needs to be retrained once the weights are changed. Accordingly, an online adaptation for changed weights is necessary. That is, the single agent can handle the changing coefficients α, β, and γ without learning from scratch. It motivates us to embed the PPA preference space into the framework.

Embed Preference Space into RL

The PPA preference space Φ is the set of preference vectors $\boldsymbol{\phi} = (\alpha, \beta, \gamma)$, which balance the PPA values in various degrees and satisfy the simplex constraints, i.e., $\forall i,\ \boldsymbol{\phi}_i \geq 0,\ \sum_i \boldsymbol{\phi}_i = 1$. We embed Φ into RL, making the agent learn the *convex coverage set* (CCS) w.r.t. Equation (2) (Roijers, Whiteson, and Oliehoek 2015; Roijers and Whiteson 2017). Hence, a single agent can maximize r without retraining or fine-tuning in the DSE when $\boldsymbol{\phi}$ is changed.

PPA values are negatively correlated. The respective optimal performance, power, and area cannot be achieved simultaneously. Therefore, the reward space R has the *Pareto frontier*. Pareto microarchitectures cannot improve PPA further concurrently. CCS is the convex subset of the Pareto frontier, as formulated in Equation (3).

$$\begin{aligned} \text{CCS} = \{\boldsymbol{r} \in \mathcal{PF}(R) \mid \\ \exists \boldsymbol{\phi} \in \Phi, \boldsymbol{r}\boldsymbol{\phi}^\top \geq \boldsymbol{r}'\boldsymbol{\phi}^\top, \forall \boldsymbol{r}' \in \mathcal{PF}(R)\}, \end{aligned} \tag{3}$$

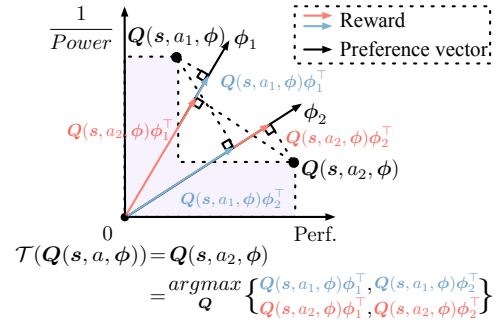

Figure 4: Optimization procedure with Equation (4).

where $\mathcal{PF}(R)$ is the Pareto frontier of R, and $\mathcal{PF}(R) = \{\boldsymbol{r} \mid \nexists \boldsymbol{r}' \succeq \boldsymbol{r}, \forall \boldsymbol{r}, \forall \boldsymbol{r}' \in R\}$ [2]. Equation (3) indicates that solutions having $\boldsymbol{r}$ in the CCS are optimal by attaining the maximal r (see Equation (2)) when a specific $\boldsymbol{\phi}$ is given.

To facilitate the agent's learning of the CCS during training, a generalized Bellman optimality equality is applied (Yang, Sun, and Narasimhan 2019). The generalized equality is an extension from the single objective Bellman optimality. The main idea is to optimize the policy towards maximal r given a particular $\boldsymbol{\phi}$, as shown in Equation (4).

$$\begin{aligned} \boldsymbol{Q}(\boldsymbol{s},a,\boldsymbol{\phi}) &= \boldsymbol{r}(\boldsymbol{s},a) + \zeta \mathbb{E}_{\boldsymbol{s}' \sim \mathcal{P}(\cdot|\boldsymbol{s},a)} \mathcal{T}(\boldsymbol{Q}(\boldsymbol{s}',a,\boldsymbol{\phi})), \\ \mathcal{T}(\boldsymbol{Q}(\boldsymbol{s}',a,\boldsymbol{\phi})) &= \underset{\boldsymbol{Q}}{arg} \underset{a' \in A, \boldsymbol{\phi}' \in \Phi}{max} \boldsymbol{Q}(\boldsymbol{s}',a',\boldsymbol{\phi}')\boldsymbol{\phi}^\top, \end{aligned} \tag{4}$$

where ζ is a discount factor. $\boldsymbol{Q}(\boldsymbol{s},a,\boldsymbol{\phi})$ is the state-action vector when $\boldsymbol{s}$ is the state, a is the action, and $\boldsymbol{\phi}$ is the preference vector. $\mathcal{T}(\boldsymbol{Q}(\boldsymbol{s},a,\boldsymbol{\phi}))$ is $\boldsymbol{Q}$, which attains the maximal r via traversing the action space A, and sampled $\boldsymbol{\phi}'$ [3].

Figure 4 details an example optimization with Equation (4) in the performance-power space, given $\boldsymbol{\phi}$. Before applying Equation (4), we sample multiple different $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$, holding an insight that the agent can learn to generate other policies according to varied preferences. At state $\boldsymbol{s}$, the agent is faced with actions a_1 and a_2. Under $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$, four rewards are highlighted with blue and red colors. Ultimately, the policy is optimized with $\boldsymbol{Q}(\boldsymbol{s},a_2,\boldsymbol{\phi})$ since it achieves the maximal reward among all rewards.

Our RL framework adopts the asynchronous advantage actor-critic (A3C) (Mnih, Badia et al. 2016) in favor of high training efficiency over PPO (Schulman et al. 2017) or SAC (Haarnoja et al. 2018). The actor is a policy network used to generate an action. The critic evaluates the complete state to determine whether the optimization becomes better or worse than expected. The gradients of the actor $\boldsymbol{\theta}_a$ is listed in Equation (5).

$$\begin{aligned} \bigtriangledown \boldsymbol{\theta}_a = & \kappa \bigtriangledown_{\boldsymbol{\theta}_a} H(\pi(\boldsymbol{s}_t;\boldsymbol{\theta}_a)) + \\ & \mathbb{E}_{\xi \sim \pi}[\sum_{t=0}^{\infty} \bigtriangledown_{\boldsymbol{\theta}_a} \log \pi_{\boldsymbol{\theta}_a}(a_t \mid \boldsymbol{s}_t) A(\boldsymbol{s}_t,a_t,\boldsymbol{\phi}')\boldsymbol{\phi}^\top], \end{aligned} \tag{5}$$

[2] $\boldsymbol{r}' \succeq \boldsymbol{r}$ denotes that each element of $\boldsymbol{r}'$ is better than $\boldsymbol{r}$.

[3] The size of A at each step is small, allowing us to traverse efficiently. However, since Φ is an uncountable set, we sample multiple $\boldsymbol{\phi}$ in the training and compute $\mathcal{T}(\boldsymbol{Q}(\boldsymbol{s}',a,\boldsymbol{\phi}))$ based on these samples otherwise.

where $A(\boldsymbol{s}_t,a_t,\boldsymbol{\phi}') = Q(\boldsymbol{s}_t,a_t,\boldsymbol{\phi}') - V(\boldsymbol{s}_t,\boldsymbol{\phi}')$ is the advantage function featuring relatively low variance, ξ is a trajectory following the policy π, and $\boldsymbol{\theta}_a$ denotes parameters of the actor. The entropy of the policy π is incorporated in optimizing the actor ($H(\pi(\boldsymbol{s}_t;\boldsymbol{\theta}_a))$. It can prevent the agent from always selecting the currently found best action. A coefficient κ controls the strength of entropy regularization. For the critic, Equation (6) gives the loss function with an L2 normalization applied between two state-action vectors.

$$\begin{aligned} L_c = & \rho \|(\boldsymbol{Q}^* - \boldsymbol{Q}(\boldsymbol{s},a,\boldsymbol{\phi}';\boldsymbol{\theta}_c))\boldsymbol{\phi}^\top\|_2^2 + \\ & (1-\rho)\|\boldsymbol{Q}^* - \boldsymbol{Q}(\boldsymbol{s},a,\boldsymbol{\phi}';\boldsymbol{\theta}_c)\|_2^2, \end{aligned} \tag{6}$$

where ρ is a coefficient to balance these two terms, $\boldsymbol{\theta}_c$ denotes parameters of the critic, and $\boldsymbol{Q}^*$ is obtained from Equation (4). The first term in Equation (6) enforces optimizing the critic network w.r.t. the maximal reward shown in Equation (2). The *n-step* TD errors (Peng and Williams 1994) is leveraged. However, Equation (5) requires many transition samples to give a relatively accurate gradients approximation for a steady and stable improvement. We employ the generalized advantage estimator (GAE) to handle it (Schulman et al. 2016), as listed in Equation (7).

$$\boldsymbol{r}_t = \sum_{n=0}^{N} (\lambda\zeta)^{N-n} (\boldsymbol{r}_{t+k} + \zeta V_{t+1+k}(\boldsymbol{s}_t,\boldsymbol{\phi}') - V_{t+k}(\boldsymbol{s}_t,\boldsymbol{\phi}')), \tag{7}$$

where λ is a coefficient controlling the strength of the exponential-weighted average.

Conditioned Actor-Critic Network

The input of our actor and critic networks is the concatenation of state and corresponding preference vectors. The preference vectors serve as conditional inputs to the actor and critic networks. Both networks are multilayer perceptrons with leaky ReLU as the activation function. The intuition of the concatenation is to support the online adaptation of changed preferences for agents. Hence, many policies are optimized on-the-fly (Abels et al. 2019).

Accelerate Learning via Lightweight Environment

Training the agent with pre-RTL simulation infrastructures as the RL environment is inaccurate while using EDA tools in the loop is inefficient. So, we propose a "lightweight" environment to combine the merits of both modeling flows, which the "lightweight" refers to that our environment can achieve a speed-up of $100\times \sim 110\times$ in PPA estimation compared to using EDA tools in the training loop.

The lightweight environment is based on the calibration, which is set up before the RL training (Zhai et al. 2021). In the calibration, we leverage the EDA flow as a golden PPA (ground truths) generation flow. And we adopt the pre-RTL simulation infrastructures as feature extraction flow. The extracted features encode knowledge to microarchitectures and workloads, e.g., queue, buffer, or stacks' number of reads and writes, number of load and store instructions, etc. Supervised learning is then applied to train PPA black-box models such as XGBoost (Chen and Guestrin 2016) separately, with

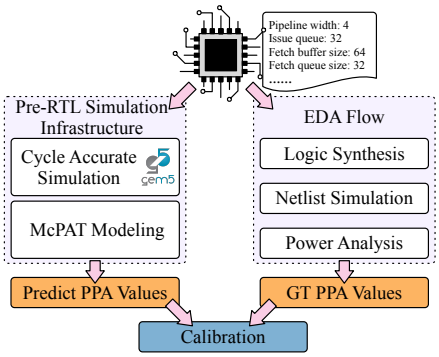

Figure 5: Overview of the PPA calibration.

the loss function defined in Equation (8).

$$L = |f(\boldsymbol{s}, \boldsymbol{e}, \boldsymbol{p}) - y_{\text{gt}}|_2^2, \tag{8}$$

where f is a black-box model. $\boldsymbol{s}$, $\boldsymbol{e}$, and $\boldsymbol{p}$ are inputs of the model, denoting state, cycle accurate simulation statistics, and other PPA-related features (e.g., leakage and subthreshold power, etc.), respectively. y_{gt} is the ground truth.

Figure 5 provides an overview of the calibration flow. In the RL training and DSE, a microarchitecture is initially evaluated using pre-RTL simulation infrastructures and is calibrated with trained PPA black-box models. Furthermore, we duplicate the environment into multiple instances, permitting higher training parallelism.

Why RL?

Previous data-driven methods apply statistical analysis (Li et al. 2016), Gaussian process (Bai et al. 2021), etc. However, a limitation can be observed. *Most previous methods attribute the degree of PPA difference to the distance between feature embeddings of microarchitectures.* For example, the Gaussian process assumes the existence of such relations (Bai et al. 2021; Williams and Rasmussen 2006). On the contrary, we find the relation does not hold generally, and demonstrate it with an anti-example shown in Figure 6.

Figure 6 provides an example with SonicBOOM (Zhao et al. 2020). M1 is the baseline microarchitecture. M2 changes the branch predictor (Seznec and Michaud 2006), M3 reduces the decode width, and M4 decreases branch speculation tags, respectively. t-SNE (Van der Maaten and Hinton 2008) is utilized to visualize the embedding distances to M1. Notwithstanding that M2 and M3 have the same distance to M1, they incur different PPA value gaps to M1. M3 has 8.54%, 3.00%, and 5.09% smaller PPA values than M1. M2 demonstrates a more substantial difference, i.e., 13.09%, 23.75%, and 14.48% lower PPA values than M1. The embedding distance between M1 and M2 is closer than that between M1 and M4. However, compared with M2, M4's PPA values are even closer to M1, i.e., M1 outperforms IPC by 0.36% [4], dissipating 3.67% more power and 1.39% larger area than M4.

Our RL solution can remove unrealistic assumptions. Thus, it alleviates the limitations of prior data-driven methods. However, RL might be one of many remedies while our MDP formulation can capture the structure of the problem.

[4]Instruction per cycle (IPC) is a performance metric.

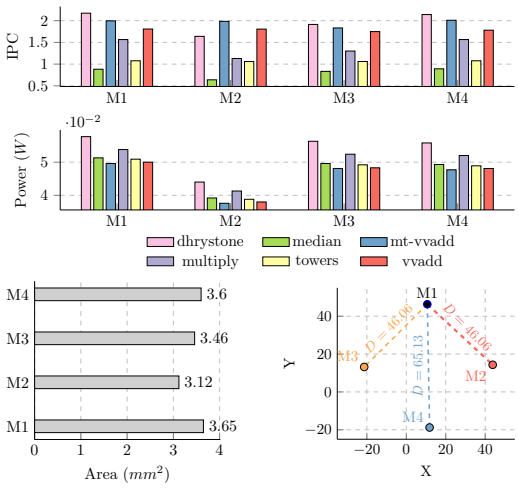

Figure 6: Four SonicBOOM microarchitectures' PPA values of six widely-used benchmarks reported from EDA tools, and the visualization of embeddings distances.

Experiments

RISC-V Microarchitecture Design Space

We evaluate the proposed RL framework with representative in-order and out-of-order RISC-V microprocessors, Rocket (Asanović et al. 2016) and different scales of SonicBOOM (Zhao et al. 2020) (categorized by "pipelineWidth"), as shown in Table 1. We include cache structures, branch predictors, functional units, load/store units, issue units, etc., in the design space. The Rocket and SonicBOOM design space size is 5.18×10^6 and 1.02×10^{16}, respectively.

Experimental Settings & Baselines

All experiments are conducted on 80 Quad Intel(R) Xeon(R) CPU E7-4820 V3 cores with a 1 TB main memory. The PPA values reported in the main results are from commercial EDA tools. Specifically, the performance, power, and area values are obtained from Synopsys VCS M-2017.03 (Mis 2023g), Synopsys PrimeTime PX R-2020.09-SP1 (Mis 2023f), and Cadence Genus 18.12-e012_1 (Mis 2023a) with 7-nm technology (Clark et al. 2016). Due to the page limit, we mainly show the results of SonicBOOM for some experiments, specifically for the Large SonicBOOM.

The coefficient κ in Equation (5) is set as 1, ρ in Equation (6) is 0.5, λ in Equation (7) is 0.95 and the discount factor ζ in Equation (7) is 0.99. Adam optimizer is used, and the initial learning rate is 0.001.

We compare our method with current state-of-the-arts, i.e., Bayesian optimization-based (Bai et al. 2021) (ICCAD'21), Adaboost-based (Li et al. 2016) (DAC'16), ranking-based (Chen et al. 2014) (ISCA'14), and human efforts (Asanović et al. 2016; Zhao et al. 2020). The baselines are implemented according to the original papers. We use `towers`, `vvadd`, `spmv` from official RISC-V tests (Mis 2023d) as workloads in the DSE. Results on more workloads

Table 1: RISC-V Microarchitecture Design Space

Design	Component	Parameters	Candidate
Rocket	Branch predictor	RAS	$0:12:3$ *
		BTB.nEntries	$0:56:14$
		BHT.nEntries	$0:1024:256$
	I-cache	nWays	$1, 2, 4$
		nTLBWays	$4:32:4$
	Functional unit	FPU	$1, 2$
		mulDiv	$1, 2, 3$
		VM	$1, 2$
	D-cache	nSets	$32, 64$
		nWays	$1, 2, 4$
		nTLBWays	$4:32:4$
		nMSHRs	$1, 2, 3$
Small/Medium Large/Mega Giga SonicBOOM	Branch predictor	Type	$1, 2, 3$
		maxBrCount	$4:22:2$
	IFU	numFetchBufferEntries	$6:46:2$
		fetchWidth	$4, 8$
		ftq.nEntries	$12:64:4$
	pipelineWidth		$1:5:1$
	ROB		$24:160:4$
	PRF	numIntPhysRegisters	$40:176:8$
		numFpPRF	$34:132:6$
	ISU	numFpPhysRegisters	$1:5:1$
		numEntries	$6:52:2$
		dispatchWidth	$1:5:1$
	LSU	LDQ	$6:32:2$
		STQ	$6:36:2$
	I-cache	nWays	$4, 8$
		nSets	$32, 64$
	D-cache	nWays	$4, 8$
		nSets	$64, 128$
		nMSHRs	$2:10:2$

* The values are start number:end number:stride, e.g., $0:12:3$ denotes the entries of RAS can be 0, 3, 6, etc., until 12.

are also elucidated. To compare the efficiency of algorithms fairly, all baselines adopt the lightweight environment, but searched solutions are re-evaluated with EDA tools.

Accuracy of Lightweight PPA Models

We use the Kendall τ and the mean absolute percentage error (MAPE) to measure the accuracy of lightweight PPA models. The higher the Kendall τ and the lower the MAPE, the more accurate the lightweight PPA models are.

Figure 7 lists the accuracy of lightweight PPA XGBoost models and the ratio of microarchitectures in the training data set leveraged in the calibration flow. In the first row, the blue line "GT = Pred" visualizes the error when PPA models are trained using the entire training data set. The Kendall τ are 0.93, 0.95, and 0.94 for PPA models, respectively, indicating acceptable accuracy when using these models in the RL framework. However, a question arises of how much data set is needed in the calibration flow. We answer the question by testing PPA models trained on different scales of training data set for unseen designs until the Kendall τ and MAPE cannot be improved further. Results are shown in the second row of Figure 7. By leveraging around $800 \sim 900$ SonicBOOM microarchitecture designs, the Kendall τ for PPA modeling results can achieve higher than 0.92. The agent could learn the bias in the RL training if lightweight PPA models are fixed. The model's online update strategy remains to be discussed in future work.

RL Training

Figure 8 displays the RL training metric curves for Large SonicBOOM, which include PPA values, and specific values of PPA preference vectors. Different PPA preference vectors are sampled throughout the training, resulting in per-

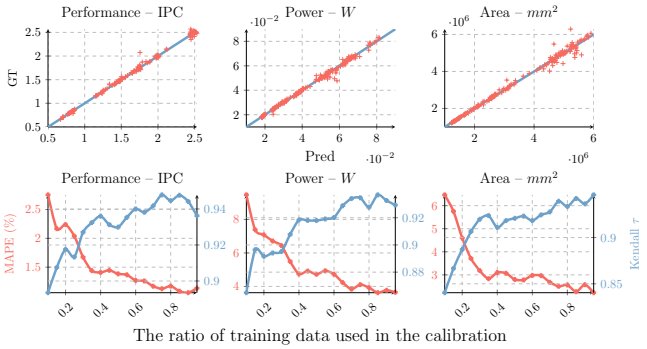

Figure 7: The accuracy of lightweight PPA models, and MAPE and Kendall τ curves w.r.t. the calibration data size.

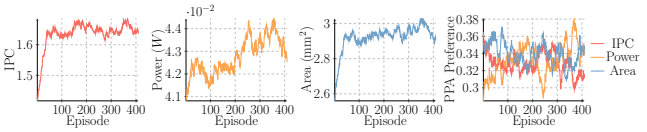

Figure 8: RL training status of Large SonicBOOM.

turbed PPA values received in each episode. IPC curves are increased gradually and flattened eventually as trained with more episodes. Power and area values are changing in response to divergent PPA preference vectors.

Comparison w. Human Efforts & Prior Arts

Three metrics: Perf/Power, Perf/Area, and (Perf $\times$ Perf)/(Power $\times$ Area) are used in the experiments. These metrics measure how much performance per watt, performance per area, and PPA trade-off a microprocessor can attain. The higher the values, the better the power/area efficiency of the microprocessor. In the DSE, we use predetermined PPA preference vectors for different scales of SonicBOOM. The preference vectors are $(1/12, 1/12, 10/12)$, $(1/7, 1/7, 5/7)$, $(1/3, 1/3, 1/3)$, $(5/7, 1/7, 1/7)$, and $(10/12, 1/12, 1/12)$ for Small, Medium, Large, Mega, and Giga SonicBOOM, respectively. We use $(1/3, 1/3, 1/3)$ for Rocket. These preference vectors are used in the RL to identify optimal designs and to calculate scalar rewards for solutions obtained through the baseline algorithms. We facilitate fair comparisons between different methodologies by comparing the solutions that yield the maximal reward among our method and the baseline algorithms in the abovementioned three metrics.

The rationale behind setting such preference vectors is to emphasize specific design priorities based on the scale of the microprocessors. We prioritize higher power and area efficiency for small microprocessors, as reflected in the preference vectors. On the other hand, for larger microprocessors, we emphasize higher performance. For the middle scale of SonicBOOM, i.e., Large SonicBOOM, we aim for a higher degree of balance among the PPA values.

Table 2 lists the results. The relative runtime for exploration is also reported. Explored Rocket and nearly all scales of SonicBOOM by our method are better than prior works

Table 2: Comparison with Human Efforts and Prior Arts

Design	Method	Performance IPC	Power W	Area mm^2	Perf / Power		Perf / Area		(Perf × Perf) / (Power × Area)		Runtime
					Val.	Ratio	Val.	Ratio	Val.	Ratio	
Rocket	Human Efforts	0.7338	0.0027	0.9082	267.4708	– [1]	0.8080	–	216.1090	–	–
	ISCA'14	0.8157	0.0023	0.7943	**359.3222**	**1.3434**×	1.0270	1.2710×	369.0075	1.7075×	8.6111×
	DAC'16	0.5485	0.0018	0.5337	305.3090	1.1415×	1.0278	1.2721×	313.8042	1.4527×	5.8961×
	ICCAD'21	0.7278	0.0021	0.7448	352.7177	1.3187×	0.9771	1.2093×	344.6327	1.5947×	1.5011×
	Ours	0.7278	0.0023	0.5762	313.6958	1.1728×	**1.2631**	**1.5633**×	**396.2335**	**1.8335**×	**1.0000**
Small SonicBOOM	Human Efforts	0.7837	0.0203	1.5048	38.6057	–	0.5209	–	20.1062	–	–
	ISCA'14	0.8197	0.0150	1.2838	54.7692	1.4187×	0.6385	1.2260×	34.9710	1.7393×	5.8033×
	DAC'16	0.8076	0.0147	1.2512	54.8119	1.4198×	0.6454	1.2393×	35.3765	1.7594×	4.7918×
	ICCAD'21	0.8469	0.0200	1.5026	42.3436	1.0968×	0.5636	1.0821×	23.8645	1.1869×	1.3053×
	Ours	0.8403	0.0152	1.2538	**55.2813**	**1.4320**×	**0.6702**	**1.2868**×	**37.0491**	**1.8427**×	**1.0000**
Medium SonicBOOM	Human Efforts	1.1938	0.0256	1.9332	46.6952	–	0.6175	–	28.8363	–	–
	ISCA'14	1.2362	0.0196	1.6242	**62.9622**	**1.3484**×	**0.7611**	**1.2324**×	**47.9192**	**1.6618**×	5.6879×
	DAC'16	1.3757	0.0254	1.9247	54.0894	1.1584×	0.7148	1.1574×	38.6609	1.3407×	4.6966×
	ICCAD'21	1.4454	0.0271	2.1583	53.3342	1.1422×	0.6697	1.0844×	35.7170	1.2386×	1.2793×
	Ours	1.2872	0.0206	1.7351	62.5886	1.3404×	0.7419	1.2014×	46.4339	1.6103×	**1.0000**
Large SonicBOOM	Human Efforts	1.4871	0.0446	3.2055	33.3430	–	0.4639	–	15.4686	–	–
	ISCA'14	1.4900	0.0309	2.5420	48.2184	1.4461×	0.5861	1.2634×	28.2626	1.8271×	5.8920×
	DAC'16	1.4919	0.0324	2.6744	45.9976	1.3795×	0.5578	1.2024×	25.6592	1.6588×	4.8651×
	ICCAD'21	1.9162	0.0409	3.6715	46.8507	1.4051×	0.5219	1.1250×	24.4520	1.5808×	1.3252×
	Ours	1.5882	0.0314	2.5643	**50.6324**	**1.5185**×	**0.6193**	**1.3350**×	**31.3580**	**2.0272**×	**1.0000**
Mega SonicBOOM	Human Efforts	1.9500	0.0578	4.8059	33.7571	–	0.4058	–	13.6972	–	–
	ISCA'14	2.4957	0.0566	5.3676	44.0942	1.3062×	0.4650	1.1459×	20.5020	1.4968×	5.5443×
	DAC'16	2.4995	0.0562	5.3797	44.4483	1.3167×	0.4646	1.1451×	20.6513	1.5077×	4.5780×
	ICCAD'21	2.4823	0.0607	4.7008	40.9170	1.2121×	**0.5281**	**1.3014**×	21.6066	1.5774×	1.2470×
	Ours	2.5232	0.0557	5.2512	**45.3005**	**1.3420**×	0.4805	1.1842×	**21.7674**	**1.5892**×	**1.0000**
Giga SonicBOOM	Human Efforts	1.8717	0.0716	5.0691	26.1538	–	0.3692	–	9.6572	–	–
	ISCA'14	2.2528	0.0622	6.0010	36.2192	1.3849×	0.3754	1.0167×	13.5970	1.4080×	5.6321×
	DAC'16	2.2522	0.0773	5.5995	29.1480	1.1145×	**0.4022**	**1.0893**×	11.7236	1.2140×	4.6505×
	ICCAD'21	2.2650	0.0745	5.8652	30.4162	1.1630×	0.3862	1.0459×	11.7460	1.2163×	1.2668×
	Ours	2.2692	0.0595	5.7459	**38.1587**	**1.4590**×	0.3949	1.0695×	**15.0696**	**1.5605**×	**1.0000**

[1] "–" denotes not applicable.

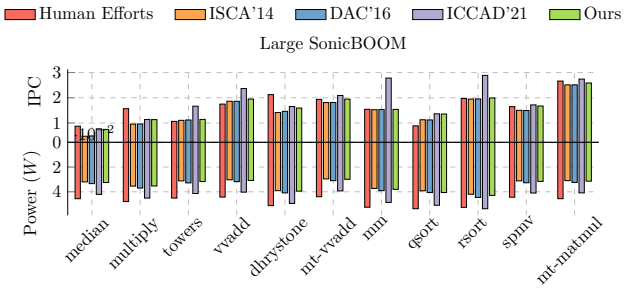

Figure 9: Analysis w. more workloads.

and human efforts. In summary, our solutions achieve an average of 24.64%, 17.13%, and 6.33% than ICCAD'21, DAC'16, and ISCA'14 in PPA trade-off, respectively. Moreover, the RL solutions outperform human efforts up to 2.03× better in $(\text{Perf} \times \text{Perf})/(\text{Power} \times \text{Area})$. And our method can find those solutions using an average of 4130.89 seconds, i.e., 4.07× higher efficiency than baselines. For Medium SonicBOOM, power/area efficiency is comparable since our solution trades 4.13% more performance than ISCA'14.

Analysis w. More Workloads

We analyze explored microarchitectures with more workloads to study how RL solutions outperform other methods and human implementations. Figure 9 lists related results for Large SonicBOOM. The areas for human implementations, ISCA'14, DAC'16, ICCAD'21 and ours are 3.21 mm^2, 2.54 mm^2, 2.67 mm^2, 3.67 mm^2, and 2.56 mm^2, respectively. While human implementations and the ICCAD'21 solution achieve higher performance on most workloads, they require more area and dissipate more power, resulting in a lower performance per watt or PPA trade-off. Compared to human implementations, our solution demonstrates significant improvements in three metrics by factors of 1.35×, 1.23×, and 1.66×, respectively. Additionally, it outperforms the baselines with average improvements of 7.74%, 12.47%, and 21.15% in the corresponding metric values.

Our solution adopts a Gshare branch predictor, 16KB I-cache, and 32KB D-cache. The PPA trade-off is further enhanced by increasing integer issue queue sizes and removing redundant resources. Our solution achieves a superior PPA trade-off by selecting a more suitable branch predictor, cache structures, and a balanced allocation of resources for queues, stacks, and buffers.

Conclusion

We propose an RL solution to deal with automated RISC-V microarchitecture design. Our solution removes unrealistic assumptions and is tightly coupled with expert knowledge. Experiments show that our method on RISC-V Rocket and SonicBOOM achieves an average of 16.03% PPA trade-off improvement over prior state-of-the-art approaches with 4.07× higher efficiency.

Acknowledgements

This work is supported in part by National Key R&D Program of China (2022YFB2901100) and The Research Grants Council of Hong Kong SAR (No. CUHK14210723).

References

2022. Europe steps up as RISC-V ships 10bn cores. https://www.eenewseurope.com/en/europe-steps-up-as-risc-v-ships-10bn-cores/.

2023a. Cadence Genus Synthesis Solution. https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html.

2023b. EDA Design Flow. https://en.wikipedia.org/wiki/Design_flow_(EDA).

2023c. Hardware Description Language. https://en.wikipedia.org/wiki/Hardware_description_language.

2023d. Official RISC-V Tests. https://github.com/riscv-software-src/riscv-tests.

2023e. RISC-V Instruction Set Architecture. https://en.wikipedia.org/wiki/RISC-V.

2023f. Synopsys PrimeTime PX Power Analysis. https://news.synopsys.com/index.php?s=20295&item=123041.

2023g. Synopsys VCS. https://www.synopsys.com/verification/simulation/vcs.html.

Abels, A.; Roijers, D.; Lenaerts, T.; Nowé, A.; and Steckelmacher, D. 2019. Dynamic Weights In Multi-objective Deep Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, 11–20. PMLR.

Asanović, K.; Avizienis, R.; Bachrach, J.; et al. 2016. The Rocket Chip Generator. Technical report, University of California, Berkeley.

Bai, C.; Huang, J.; Wei, X.; Ma, Y.; Li, S.; Zheng, H.; Yu, B.; and Xie, Y. 2023a. ArchExplorer: Microarchitecture Exploration Via Bottleneck Analysis. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 268–282.

Bai, C.; Sun, Q.; Zhai, J.; Ma, Y.; Yu, B.; and Wong, M. 2021. BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1–9.

Bai, C.; Sun, Q.; Zhai, J.; Ma, Y.; Yu, B.; and Wong, M. D. 2023b. BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*.

Binkert, N.; Beckmann, B.; Black, G.; et al. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News*, 39(2): 1–7.

Carlson, T. E.; Heirman, W.; and Eeckhout, L. 2011. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *ACM/IEEE Supercomputing Conference (SC)*, 1–12.

Carlson, T. E.; Heirman, W.; Eyerman, S.; Hur, I.; and Eeckhout, L. 2014. An Evaluation of High-level Mechanistic Core Models. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(3): 1–25.

Chen, C.; Xiang, X.; Liu, C.; Shang, Y.; Guo, R.; Liu, D.; Lu, Y.; Hao, Z.; Luo, J.; Chen, Z.; et al. 2020. Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance risc-v processor with vector extension: Industrial product. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 52–64.

Chen, T.; and Guestrin, C. 2016. XGBoost: A Scalable Tree Boosting System. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 785–794.

Chen, T.; Guo, Q.; Tang, K.; Temam, O.; Xu, Z.; Zhou, Z.-H.; and Chen, Y. 2014. Archranker: A ranking approach to design space exploration. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 1–12.

Clark, L. T.; Vashishtha, V.; Shifren, L.; Gujja, A.; Sinha, S.; Cline, B.; Ramamurthy, C.; and Yeric, G. 2016. ASAP7: A 7-nm FinFET Predictive Process Design Kit. *Microelectronics Journal*, 53: 105–115.

Eyerman, S.; Eeckhout, L.; Karkhanis, T.; and Smith, J. E. 2009. A Mechanistic Performance Model for Superscalar Out-of-order Processors. *Transactions on Computer Systems*, 27(2): 1–37.

Grayson, B.; Rupley, J.; Zuraski, G. Z.; Quinnell, E.; Jiménez, D. A.; Nakra, T.; Kitchin, P.; et al. 2020. Evolution of the Samsung Exynos CPU Microarchitecture. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 40–51. IEEE.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning (ICML)*, 1861–1870. PMLR.

Karkhanis, T. S.; and Smith, J. E. 2007. Automated design of application specific superscalar processors: an analytical approach. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 402–411.

Li, D.; Yao, S.; Liu, Y.-H.; Wang, S.; and Sun, X.-H. 2016. Efficient design space exploration via statistical sampling and AdaBoost learning. In *ACM/IEEE Design Automation Conference (DAC)*, 1–6.

Li, S.; Ahn, J. H.; Strong, R. D.; Brockman, J. B.; Tullsen, D. M.; and Jouppi, N. P. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 469–480.

Mnih, V.; Badia, A. P.; et al. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, volume 48, 1928–1937.

Peng, J.; and Williams, R. J. 1994. Incremental Multi-step Q-learning. In *Machine Learning Proceedings 1994*, 226–232. Elsevier.

Roijers, D. M.; and Whiteson, S. 2017. Multi-objective Decision Making. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 11(1): 1–129.

Roijers, D. M.; Whiteson, S.; and Oliehoek, F. A. 2015. Computing Convex Coverage Sets for Faster Multi-objective Coordination. *Journal of Artificial Intelligence Research*, 52: 399–443.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2020. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature*, 588(7839): 604–609.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In Bengio, Y.; and LeCun, Y., eds., *International Conference on Learning Representations (ICLR)*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*.

Seznec, A.; and Michaud, P. 2006. A case for (partially) TAgged GEometric history length branch prediction. *The Journal of Instruction-Level Parallelism*, 8: 23.

Smith, J. E.; and Pleszkun, A. R. 1988. Implementing Precise Interrupts in Pipelined Processors. *IEEE Transactions on Computers*, 37(5): 562–573.

Van der Maaten, L.; and Hinton, G. 2008. Visualizing Data Using t-SNE. *Journal of Machine Learning Research (JMLR)*, 9(11).

Williams, C. K.; and Rasmussen, C. E. 2006. *Gaussian Processes for Machine Learning*, volume 2. MIT press Cambridge, MA.

Xu, Y.; Yu, Z.; Tang, D.; Chen, G.; Chen, L.; Gou, L.; Jin, Y.; Li, Q.; Li, X.; Li, Z.; et al. 2022. Towards Developing High Performance RISC-V Processors Using Agile Methodology. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 1178–1199. IEEE.

Yang, R.; Sun, X.; and Narasimhan, K. 2019. A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation. In *Annual Conference on Neural Information Processing Systems (NIPS)*.

Yu, Z.; Bai, C.; Hu, S.; Chen, R.; He, T.; Yuan, M.; Yu, B.; and Wong, M. 2023. IT-DSE: Invariance Risk Minimized Transfer Microarchitecture Design Space Exploration. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1–9. IEEE.

Zhai, J.; Bai, C.; Zhu, B.; Cai, Y.; Zhou, Q.; and Yu, B. 2021. McPAT-Calib: A Microarchitecture Power Modeling Framework for Modern CPUs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1–9. IEEE.

Zhao, J.; Korpan, B.; Gonzalez, A.; and Asanovic, K. 2020. SonicBOOM: The 3rd Generation Berkeley Out-of-order Machine. In *Workshop on Computer Architecture Research with RISC-V (CARRV)*.