RISC-V MICROARCHITECTURE EXPLORATION VIA REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Microarchitecture determines a processor's detailed structure, affecting the processor's performance, power, and area (PPA). Deciding on a microarchitecture to achieve a good balance between the PPA values is a non-trivial problem. Previous arts mainly require expert knowledge. The solution becomes inefficient as nowadays processors become increasingly complicated. Machine learning has solved problems automatically with high-quality results via reduced access to domain knowledge. In this paper, we formulate the problem as a Markov decision process and propose an end-to-end solution framework via reinforcement learning. Firstly, a dynamically-weighted reward design is proposed to accommodate the optimization of multiple negatively-correlated objectives. Secondly, local heuristic search is adopted in the action design with prior knowledge of microarchitectures. Thirdly, lightweight calibrated PPA models are incorporated to accelerate the learning process. Experimenting with electronic design automation (EDA) tools on famous RISC-V processors demonstrate that our methodology can learn from experience and outperform human implementations and previous arts' solutions in PPA and overall running time.

1 INTRODUCTION

The processor design cycle consists of numerous complicated steps, requiring high cost and workforce input to meet the strict time-to-market product delivery deadline. Engineers participate in the design loop to optimize the processor's performance, power, and area (PPA) iteratively, as shown in Figure 1. Microarchitecture is a processor's detailed structure, and exploring a microarchitecture is one step in the design cycle. A microarchitecture consists of different ***components***, *e.g.*, branch predictor, caches, *etc.* The target is finding a good microarchitecture, achieving higher performance, lower power, and smaller areas.

However, manually selecting such microarchitecture based on expert knowledge from the design space is non-trivial. On the one hand, the design space is extremely large, *i.e.*, its size can be more than ten or twenty orders of magnitude in industrial design. Chen et al. (2020); Grayson et al. (2020). On the other hand, evaluating a candidate's performance, power, and area with electronic design automation (EDA) tools is time-consuming, *e.g.*, several hours on 80-core Xeon high-performance computers with 1 TB memory for an academic in-order processor design Asanović et al. (2016).

Previous solutions require domain knowledge to study the trade-offs of each component carefully. Nevertheless, the aspiration to automate the solution process with less access to experts has never been stopped.

In this paper, we formulate the problem as a Markov decision process (MDP) and present a novel end-to-end solution framework based on reinforcement learning (RL). Specifically, we focus on the processors implementing RISC-V [1], due to its significant attention from academia and industry these years. We hold an insight: components contribute differently to the PPA values, and learning from historical experience, the agent can determine the microarchitecture via local heuristic search. It is worth noting that PPA is multiple negatively-correlated objectives. Higher performance microarchitecture is often accompanied by more power dissipation and a larger area. The reason lies that

[1]RISC-V: an instruction set architecture maintained by RISC-V foundation: `https://riscv.org/`

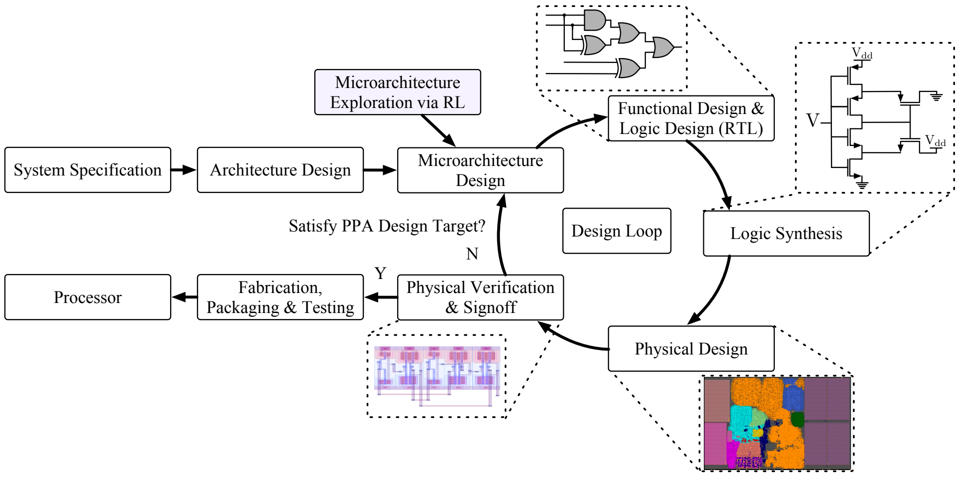

Microarchitecture Exploration via RL
System Specification
Architecture Design
Microarchitecture Design
Functional Design & Logic Design (RTL)
Logic Synthesis
Physical Design
Physical Verification & Signoff
Fabrication, Packaging & Testing
Processor
Design Loop
Satisfy PPA Design Target?
N
Y
V
Vdd

Figure 1: The processor design cycle. Engineers spare much effort to optimize a given microarchitecture with complicated steps. Determine a better microarchitecture can reduce the non-recurring engineering cost Magarshack & Paulin (2003). We solve microarchitecture exploration via RL.

more hardware resources are allocated to components, improving the processor performance while sacrificing power dissipation and area. Hence, we customize the RL to solve the problem.

The main contributions can be summarized as follows:

1) A unified reward design transfers the multiple objectives to a dynamic-weighted signal. The idea is to handle exploring microarchitectures with different PPA design preferences in the de-facto processor design cycle.

2) To accelerate the learning process, we design a lightweight environment, *i.e.*, composed of calibrated PPA models instead of heavyweight EDA tools. The calibrated PPA models accelerate the learning process by more than $110\times$ compared to EDA tools.

3) A local heuristic search is incorporated in the episode based on prior knowledge of microarchitectures. It improves the agent to explore promising microarchitectures efficiently and effectively.

4) The solution targets processors implementing RISC-V. Experiments show that our explored microarchitectures of the famous RISC-V in-order processor Rocket Asanović et al. (2016) and different scales of RISC-V out-of-order processors SonicBOOM Asanovic et al. (2015); Celio (2017); Zhao et al. (2020) outperform previous methodologies and human implementations.

2 Related Work

RISC-V & RISC-V Implementations. Unlike commercial and expensive instruction set architecture (ISA), *e.g.*, ARM, x86, *etc.*, RISC-V, pursues for free, open-source. Academia and industry have done many pioneering works for RISC-V, *i.e.*, providing open-source RISC-V processor implementations. Rocket Asanović et al. (2016), is a six-stage pipeline in-order processor. SonicBOOM Zhao et al. (2020), based on BOOM Asanovic et al. (2015); Celio (2017) is a ten-stage pipeline out-of-order design. The industry has also released several designs, *e.g.*, Xuantie-910 Chen et al. (2020) aims at high-performance application scenarios.

Microarchitecture Design Space Exploration. Microarchitecture can be parameterized as combinations of discrete integers. Each integer represents the allocated hardware resources for corresponding components, *e.g.*, number of queue entries, stack size, buffer size, *etc.*

In industry, processor designers determine the combination from expert knowledge [2]. However, personal bias may lead to sub-optimal results is a concern. The academia proposes white-box or black-box solutions to automate the solution.

[2]Intel Skylake microarchitecture determination requires many experts' efforts: `https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server)`

The white-box methods try to build analytical models to evaluate the relations between components' resources and PPA values. The exploration is conducted by sweeping the design space with analytical models Kathail et al. (2002); Brooks et al. (2003); Karkhanis & Smith (2004; 2007). Given the difficulty of constructing analytical models, black-box methods emerge. Several learning techniques are applied to solve the problem, *e.g.*, via artificial neural networks İpek et al. (2006), regression methods Lee & Brooks (2007) for POWER CPU Moudgill et al. (1999), RankBoost learning Chen et al. (2014), AdaBoost learning Li et al. (2016), Bayesian optimization Bai et al. (2021), *etc.* Regarding solution quality, black-box methods usually achieve more promising results than white-box methods. However, current black-box methods are still limited to improving the final results further.

Microarchitecture Simulation & Modeling. Accurate microarchitectures' PPA values are obtained from EDA tools, but the runtime cost is exceptionally high, restricting an efficient exploration.

The difficulty advances the development of many software processor simulation infrastructures Austin et al. (2002); Yourst (2007); Brooks et al. (2003); Patel et al. (2011); Carlson et al. (2011); Binkert et al. (2011); Sanchez & Kozyrakis (2013), and some power or area modeling tools Brooks et al. (2000); Muralimanohar et al. (2009); Li et al. (2009); Li, Sheng and Chen, Ke and Ahn, Jung Ho and Brockman, Jay B and Jouppi, Norman P (2011); Lee & Jha (2011). They can give rough PPA estimations without EDA tools, *i.e.*, achieve a trade-off between accuracy and runtime cost.

GEM5 Binkert et al. (2011) is the state-of-the-art simulator in academia, generating performance values by mimicking the executing behaviors of processors, *i.e.*, using *events statistics* (buffer reads and writers, cache accesses, *etc.*) to characterize. McPAT Li et al. (2009) is the state-of-the-art power and area modeling tool, which can hierarchically calculate power and area values based on microarchitectures and events statistics.

3 Methodology

3.1 Problem Formulation.

We use embedding to denote a microarchitecture for a specified processor, raising an abstraction to characterize it.

Definition 1 (Microarchitecture Embedding) *Microarchitecture embedding is a combination of candidate values of each component. It is denoted as a feature vector $\boldsymbol{s}$.*

We formulate the problem as a design space exploration with Definition 1.

Problem 1 (Microarchitecture Design Space Exploration) *Microarchitecture design space exploration is to solve a multi-objective optimization problem*

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

where $\mathbb{D}$ is an n-dimensional microarchitecture design space, Perf, Power, and Area denote performance, power, and area of a microarchitecture embedding $\boldsymbol{s}$.

The solution to the problem is an iterative trial and optimization process.

We pose the problem settings as a Markov decision process (MDP), *i.e.*, the exploration is analogous to searching for a strategy in go. A microarchitecture embedding is a checkerboard, with a modification of components as a decision and better results according to Equation (1) as the winning criterion.

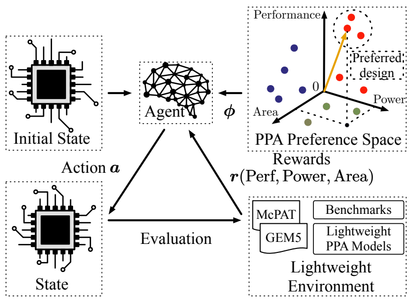


Figure 2: Overview of our RL framework

Figure 2 shows an overview of the RL framework and key elements of the MDP. The state space S comprises microarchitecture embeddings, as an example listed in Table 2. The action space A consists of determinations of hardware resources for a component. The state transition is deterministic. The reward R is the vector of PPA values. The framework constitutes two stages, *i.e.*, the learning and exploration stages. In the learning stage, the initial state and PPA preference vector are the input to the agent. The agent adjusts hardware resources until no PPA improvements are achieved within a pre-specified early-stopping criterion in each episode. Through learning from episodes, the agent explores microarchitectures corresponding to the PPA preference vector. The initial state is carefully chosen, *e.g.*, a human implementation, *etc.*, and the preference vector is resampled at the beginning of each episode. In the exploration stage, the preference vector is specified by users and set to be fixed in the entire stage, and the agent tries to explore the target design.

3.2 Dynamic-weighted Reward.

Since a single agent cannot handle multiple objectives, a weighted-sum formula is often applied in the reward design He et al. (2020); Mirhoseini et al. (2021). Such a technique is limited since an agent needs to be retrained if the coefficients controlling the trade-off between different objectives are changed. In the de-facto processor design cycle, multiple PPA preferences exist, *e.g.*, a higher performance candidate or a design emphasizing power or area efficiency more. Consequently, our reward unifies different PPA preferences in dynamic-weighted singal design.

$$R = (\alpha, \beta, \gamma)^\top \cdot \text{CONCAT}(r_{\text{perf}}, r_{\text{power}}, r_{\text{area}}) \tag{2}$$

where α, β, and γ are weights controlling the PPA trade-off, and r_{perf}, r_{power}, and r_{area} are PPA values, respectively. The single agent should handle the changing coefficients α, β, and γ without learning from scratch. It motivates us to embed the PPA preference space into the framework.

3.3 Embed Preference Space to RL.

The PPA preference space Φ contains coefficients α, β, and γ combinations, and we term them preference vectors. Each preference vector $\boldsymbol{\phi}$ adheres to the simplex constraints, *i.e.*, $\forall i,\ \phi_i \geq 0,\ \sum_i \phi_i = 1$. Since the PPA are negatively correlated, the *Pareto coverage set* (PCS) exists, *i.e.*, a group of microarchitectures belonging to this set cannot improve PPA simultaneously. Through linearization, as shown in Equation (2), the optimal solution to the problem becomes the *convex coverage set* (CCS) Roijers et al. (2015); Roijers & Whiteson (2017) The convex coverage set is the subset of the PCS that achieves maximal reward with a given $\boldsymbol{\phi}$, as shown in Equation (3).

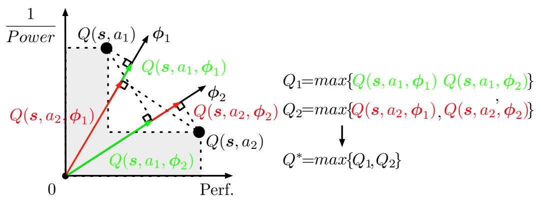

$\frac{1}{Power}$
$Q(\boldsymbol{s}, a_1)$
$\boldsymbol{\phi}_1$
$Q(\boldsymbol{s}, a_1, \boldsymbol{\phi}_1)$
$\boldsymbol{\phi}_2$
$Q(\boldsymbol{s}, a_2, \boldsymbol{\phi}_1)$
$Q(\boldsymbol{s}, a_2, \boldsymbol{\phi}_2)$
$Q(\boldsymbol{s}, a_2)$
$Q(\boldsymbol{s}, a_1, \boldsymbol{\phi}_2)$
0
Perf.
$Q_1 = max\{Q(\boldsymbol{s}, a_1, \boldsymbol{\phi}_1)\ Q(\boldsymbol{s}, a_1, \boldsymbol{\phi}_2)\}$
$Q_2 = max\{Q(\boldsymbol{s}, a_2, \boldsymbol{\phi}_1), Q(\boldsymbol{s}, a_2, \boldsymbol{\phi}_2)\}$
$Q^* = max\{Q_1, Q_2\}$

Figure 3: An example details the optimization procedure. Green and red colors are leveraged to distinguish different state-action vectors.

$$\text{CCS}(\Pi) = \{\pi^* \mid \boldsymbol{\phi}^\top V^{\pi^*}(\boldsymbol{s}_0) \geq \boldsymbol{\phi}^\top V^{\pi'}(\boldsymbol{s}_0), \forall \boldsymbol{\phi} \in \Phi, \exists \pi^*, \forall \pi' \in \Pi\}, \tag{3}$$

where $V^\pi(\boldsymbol{s}_0) = \mathbb{E}_\pi[\boldsymbol{r} \mid \boldsymbol{s} = \boldsymbol{s}_0]$ is the expectation of vector rewards received from the initial state $\boldsymbol{s}_0$, and Π is the policy set. Equation (3) indicates that the optimal policy $\pi^* \in \Pi$ can recover the CCS given $\boldsymbol{\phi}$.

Inspired by the idea of MORL Yang et al. (2019), our RL applies the extended Bellman equality, as shown in Equation (4),

$$Q^*(\boldsymbol{s}, a, \boldsymbol{\phi}) = \boldsymbol{r}(\boldsymbol{s}, a) + \gamma \mathbb{E}_{\boldsymbol{s}' \sim P(\cdot|\boldsymbol{s}, a)} \underset{Q}{arg} \underset{a' \in A,\ \exists \boldsymbol{\phi}' \in \Phi}{max} \boldsymbol{\phi}^\top Q(\boldsymbol{s}', a', \boldsymbol{\phi}'), \tag{4}$$

where γ is a discount factor. $Q^*(\boldsymbol{s}', a', \boldsymbol{\phi}') = \text{CONCAT}(r_{\text{perf}}, r_{\text{power}}, r_{\text{area}})$ is a state-action vector conditioned on a given state $\boldsymbol{s}'$, an action a', and a preference vector $\boldsymbol{\phi}'$. The transition probability $\boldsymbol{s}' \sim P(\cdot \mid \boldsymbol{s}, a)$ is deterministic. Since the preference space Φ is an uncountable set, we sample multiple $\boldsymbol{\phi}'$ and obtain the maximal $Q^*(\boldsymbol{s}', a', \boldsymbol{\phi}')$ from them.

Figure 3 details the optimization procedure. Each time we sample different $\boldsymbol{\phi}$, holding an insight that the agent can learn to generate other policies according to variable preferences. As shown in Figure 3, assume the agent explores the performance-power space. The better solution set should be far from the origin coordinate. At state $\boldsymbol{s}$, the agent is faced with two actions, *i.e.*, a_1 and a_2. Under

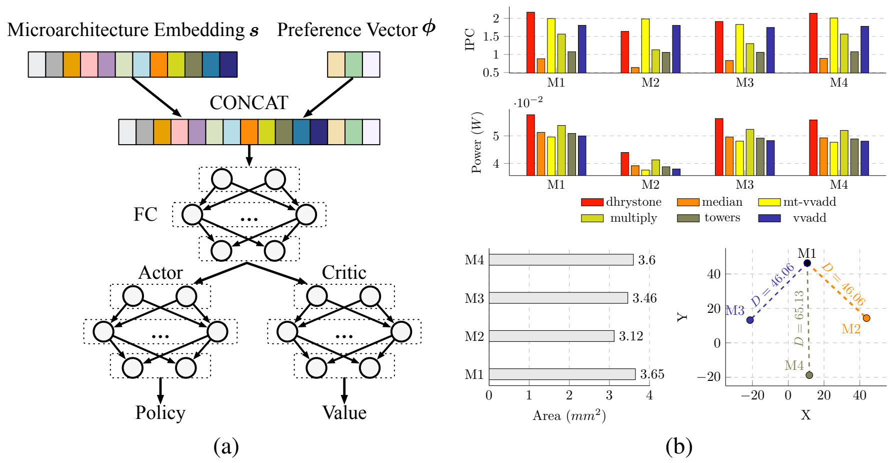


Figure 4: (a) Overview of our actor and critic network. (b) PPA values of six widely-used benchmarks on four SonicBOOM microarchitectures, and the visualization of embeddings distances.

two different preference vectors, ϕ_1 and ϕ_2, four state-action vectors exist along the direction of ϕ_1 and ϕ_2. Only the state-action vector having the greatest *utility* is selected according to Equation (4).

We use the advantage actor-critic (A2C) Mnih et al. (2016) to implement the optimization strategy. The gradients of the actor θ_a is listed in Equation (5),

$$\triangledown \boldsymbol{\theta}_a = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \triangledown_{\boldsymbol{\theta}_a} \log \pi_{\boldsymbol{\theta}_a}(a_t \mid \boldsymbol{s}_t)\boldsymbol{\phi}^\top A(\boldsymbol{s}_t, a_t, \boldsymbol{\phi}')] + \alpha \triangledown_{\boldsymbol{\theta}_a} H(\pi(\boldsymbol{s}_t;\boldsymbol{\theta}_a)), \tag{5}$$

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 incurring relatively low possible variance, τ 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 α controls the strength of entropy regularization. For the critic, the loss function is shown in Equation (6), where L2 normalization between two state-action vectors is applied.

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

In Equation (3), β is a coefficient to balance these two terms, $\boldsymbol{\theta}_c$ denotes parameters of the critic, and 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 & Williams (1994) is leveraged. However, the policy gradients *w.r.t.* Equation (5) require many transition samples to give a relatively accurate approximation for a steady and stable improvement. The difficulty is handled by using the generalized advantage estimator (GAE) Schulman et al. (2016):

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

In Equation (7), λ is a coefficient controlling the strength of the exponential-weighted average.

3.4 Actor-Critic Architecture & Local Heuristic Search

Both actor and critic networks should capture the given preference vector, so we construct their architectures following the philosophy in Abels et al. (2019). The intuition is that by creating inputs of preference vectors for the actor and critic, the actor can generate appropriate actions. The critic can evaluate new state with a known preference vector. It promotes us to concatenate the microarchitecture embedding and the preference vector, as shown in Figure 4a.

In the state transition, we find that a small but critical change can bring a big difference to microarchitectures. Figure 4b illustrates the observation. We use an example microarchitecture of SonicBOOM Zhao et al. (2020), denoted as M1, and its variants with a slight modification. M2 changes the branch predictor Seznec & Michaud (2006), M3 reduces the decode width, and M4 decreases branch speculation tags. t-SNE Van der Maaten & Hinton (2008) is utilized to visualize the embedding distances to M1, also shown in Figure 4b. Notwithstanding that M2 and M3 have the same distance to M1, they incur obviously different PPA values gaps to M1. M3 has 8.54% lower performance (IPC), 3.00% less power, and 5.09% smaller area than M1. While M2 has 13.09%, 23.75%, and 14.48% lower PPA values than M1, respectively, demonstrating a more substantial difference from M1. It is also observed that the distance of microarchitecture embedding 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%, dissipating 3.67% more power and 1.39% larger area than M4. Therefore, we argue that different components contribute differently to the PPA values, and improvements can be achieved by a small change correctly, ignoring insignificant decisions. For instance, reducing branch tags on M1 only can bring more profits for better power efficiency while maintaining overall performance (M4). Applying Gshare McFarling (1993) for M1 and neglecting others can promote more power and area efficiency (M2). Thus, we adopt a local heuristic search, *i.e.*, in each decision process, the agent only changes hardware resources for a single component, leaving others unchanged.

3.5 Accelerate Learning via Lightweight PPA Models

EDA tools consume much runtime cost to generate a label for one transition, so we rely on GEM5 and McPAT software simulation and modeling tools. Although GEM5 Binkert et al. (2011) and McPAT Li et al. (2009) can give us first-hand estimations of PPA values, they are not accurate. To mitigate the limitation, we calibrate GEM5 and McPAT with machine learning techniques via supervised learning, as shown in Equation (8)

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

where $\boldsymbol{s}$, $\boldsymbol{e}$, and $\boldsymbol{p}$ are microarchitecture embedding, simulation event statistics, and other PPA-related features (*e.g.*, leakage and sub-threshold power, *etc.*). f is a nonlinear function, and y_{gt} is the golden label obtained from EDA tools. The event statistics are acquired from software simulations. We hold the intuition that by calibrating the simulation and modeling tools against golden values generated by EDA tools, we can leverage the calibrated models to predict PPA values for unseen microarchitectures with acceptable accuracy. Hence, we replace the heavyweight EDA tools with lightweight PPA models and software simulation and modeling tools for the RL environment.

Compared to the high runtime cost with EDA tools, our lightweight PPA models can achieve $100\times \sim 110\times$ speed up in microarchitecture evaluations.

4 Experiments

We evaluate the proposed solution framework with representative in-order and out-of-order RISC-V designs, *i.e.*, Rocket Asanović et al. (2016) and different scales of SonicBOOM Zhao et al. (2020). We illustrate that one agent can explore microarchitectures with different PPA preferences of SonicBOOM (small, medium, large, mega, and giga size). The scales are different due to the pipeline width chosen for various applications (power saving or high-performance scenarios). All experiments are conducted on Quad Intel(R) Xeon(R) CPU E7-4820 V3 with a 1 TB main memory. The explored microarchitectures are evaluated with EDA tools to report the final PPA metric values. Specifically, the performance, power, and area values are obtained from Cadence Genus 18.12-e012_1, Synopsys VCS M-2017.03, and Synopsys PrimeTime PX R-2020.09-SP1, under an advanced 7-nm technology Clark et al. (2016). The evaluation procedure with EDA tools is detailed in Figure 8.

4.1 Experiments Settings

The components indexes of the microarchitecture design space for Rocket and SonicBOOM has been listed in Table 1 and Table 2, respectively. More details are shown in Table 5, Table 6 and Table 7. We compare our method with previous state-of-the-arts, *i.e.*, Bayesian optimization-based method Bai et al. (2021), and prior arts, including Adaboost RT-based learning methodology Li et al. (2016), ranking-boost-based learning strategy Chen et al. (2014), and human implementations Asanović et al. (2016); Zhao et al. (2020). The actor and critic stack multi-perceptrons and leaky

Table 1: Components Index of Rocket

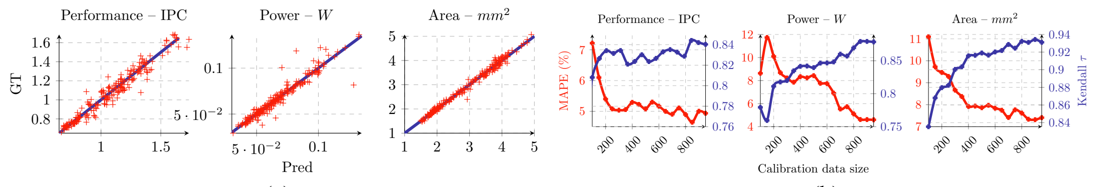
Design	Component					
	BTB [*]	I-Cache	FPU	MUL/DIV	VM	D-Cache
Rocket	1, 2, 3, 4, 5	1, 2, 3, 4, 5, 6	1, 2	1, 2, 3	1, 2	1, 2, 3, 4, 5, 6, 7, 8, 9

[*] The values are indexes but not hardware resources; please refer to Table 5 to check the design space of detailed hardware resources.

Table 2: Components Index of SonicBOOM

Design	Component							
	BP [*]	IFU	maxBrCount	ROB	PRF	ISU	LSU	D-Cache
Small SonicBOOM [+]	1, 2, 3	1, 2, 9, 10	1, 2, 3	1, 2, 3	1, 6, 8, 8, 9, 10, 13	1, 6, 13, 14	1, 2, 7	1, 3, 9
Medium SonicBOOM	1, 2, 3	2, 3, 10	3, 4, 5, 6	4, 5, 6, 7	2, 5, 10, 11	2, 7, 8, 15	2, 3, 4, 6	1, 3, 9
Large SonicBOOM	1, 2, 3	2, 3, 4, 6, 11	5, 6, 8	7, 9, 11	3, 5, 11, 12	3, 9, 10, 12, 15, 17	3, 5, 6, 8	2, 4, 6, 10
Mega SonicBOOM	1, 2, 3	4, 6, 12, 14	7, 8, 9	10, 8, 14, 15	3, 5, 4, 12	4, 11, 18	4, 5, 8, 9	4, 6, 8
Giga SonicBOOM	1, 2, 3	4, 5, 7, 13	7, 8, 9	10, 14, 15, 17	3, 5, 4, 12	5, 12, 19	4, 5, 8, 9, 10	4, 6, 8

[*] The values are indexes but not hardware resources; please refer to Table 6 and Table 7 to check the design space of detailed hardware resources.
[+] Different scales of SonicBOOM choose unique decode width and fetch buffer size.



(a) (b)

Figure 5: (a) The accuracy of lightweight PPA Models on SonicBOOM. The blue line is $GT = Pred$ that helps visualize the error. (b) The Kendall τ and MAPE curves vs. calibration data size on SonicBOOM.

Table 3: Accuracy of Lightweight PPA Models

Design	Performance		Power		Area	
	MAPE	Kendall τ	MAPE	Kendall τ	MAPE	Kendall τ
Rocket	0.4376%	0.8639	3.9017%	0.9155	1.5268%	0.9483
SonicBOOM	4.319%	0.8534	3.932%	0.9015	3.730%	0.9214

ReLU Maas et al. (2013). The coefficient α in Equation (5) is set as 1 for most designs, β in Equation (6) is set as 0.5, λ in Equation (7) is set as 0.95 and the discount factor γ in Equation (7) is set as 0.99. Adam optimizer is used, and the initial learning rate is 0.001. Although most baselines are not targeted to RISC-V implementations, their methods are proven transferable. We implement these baselines manually according to the papers. The median, vvadd, multiply, and tower benchmarks from official RISC-V benchmark suites [3] are selected in the experiments.

4.2 Accuracy of Lightweight PPA Models

The accurate performance, power, and area values for different microarchitectures are required to calibrate lightweight PPA models. We built up more than 1000 microarchitectures of Rocket and different scales of SonicBOOM. We leverage GEM5 V21.1.0.1 Lowe-Power et al. (2020) and McPAT V1.3 Li et al. (2009). The simulation is conducted with "system call" emulation.

Figure 5a illustrates the correlation between the ground truths and predictions of lightweight PPA models for different SonicBOOM microarchitectures. Three models have good correlations with actual PPA values. A question arises of how many samples are enough to be utilized to calibrate and guarantee the accuracy of lightweight PPA models for unseen microarchitectures. We obtain the final lightweight PPA models until we find that *Kendall* τ and *mean absolute percentage error* (MAPE) tested under unseen microarchitecture embeddings cannot improve further. Figure 5b lists Kendall τ and MAPE curves *w.r.t.* different calibration data size for SonicBOOM. We observe that by utilizing approximately 800 microarchitectures of SonicBOOM, lightweight PPA models become harder to be

[3] Official RISC-V Benchmark Suites: `https://github.com/riscv-software-src/riscv-tests`

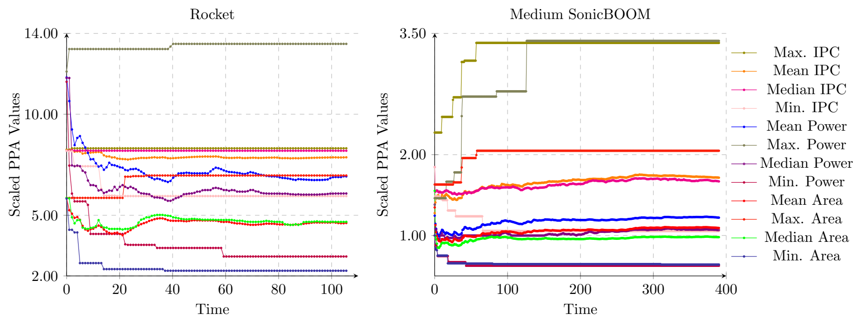


Figure 6: Scaled PPA values curve in the RL training stage.

Table 4: Comparison with Human Efforts and Prior Arts

Design	Method	Performance IPC	Power W	Area mm^2	Perf / Power		Perf / Area		(Perf × Perf) / (Power × Area)		RT (s)
					Val.	Ratio	Val.	Ratio	Val.	Ratio	
Rocket	Human Efforts Asanović et al. (2016)	0.8011	0.0026	0.9082	304.0123	–	0.8821	–	268.1664	–	–
	ArchRanker Chen et al. (2014)	0.6290	0.0021	0.5745	303.1361	↓ 0.29%	1.0948	↑ 24.11%	331.8743	↑ 23.76%	19371
	AdaBoost Learning Li et al. (2016)	0.8213	0.0024	0.7408	346.9028	↑ 14.11%	1.1086	↑ 25.68%	384.5878	↑ 43.41%	31855
	BOOM-Explorer Bai et al. (2021)	0.8215	0.0026	0.5978	310.8760	↑ 2.26%	**1.3742**	**↑ 55.79%**	427.1965	↑ 59.30%	8010
	Our Method	0.8241	0.0020	0.6397	**407.4559**	**↑ 34.03%**	1.2882	↑ 46.04%	**524.8692**	**↑ 95.73%**	12105
Small SonicBOOM	Human Efforts Zhao et al. (2020)	0.7661	0.0212	1.5048	36.1382	–	0.5091	–	18.3992	–	–
	ArchRanker Chen et al. (2014)	0.8060	0.0229	1.6672	35.1593	↓ 2.71%	0.4835	↓ 5.04%	16.9978	↓ 7.62%	11690
	AdaBoost Learning Li et al. (2016)	0.6404	0.0216	1.6634	29.7179	↓ 17.77%	0.3850	↓ 24.38%	11.4414	↓ 37.82%	42751
	BOOM-Explorer Bai et al. (2021)	0.7714	0.0214	1.5147	36.0875	↓ 0.14%	0.5092	↑ 0.02%	18.3773	↓ 0.12%	10756
	Our Method	0.7687	0.0208	1.4865	**36.9571**	**↑ 2.27%**	**0.5171**	**↑ 1.57%**	**19.1114**	**↑ 3.87%**	13616
Medium SonicBOOM	Human Efforts Zhao et al. (2020)	1.1003	0.0267	1.9332	41.2103	–	0.5692	–	23.4554	–	–
	ArchRanker Chen et al. (2014)	1.075	0.0248	1.8534	43.4176	↑ 5.36%	0.5798	↑ 1.87%	25.1731	↑ 7.32%	11305
	AdaBoost Learning Li et al. (2016)	1.0947	0.0276	2.0470	39.6647	↑ 3.75%	0.5348	↓ 6.04%	21.2129	↓ 9.56%	42929
	BOOM-Explorer Bai et al. (2021)	1.0693	0.0244	1.8251	43.8223	↑ 6.34%	0.5859	↑ 2.93%	25.6734	↑ 9.46%	10756
	Our Method	1.0873	0.0244	1.8032	**44.5615**	**↑ 8.13%**	**0.6030**	**↑ 5.94%**	**26.8693**	**↑ 14.56%**	10955
Large SonicBOOM	Human Efforts Zhao et al. (2020)	1.3128	0.0457	3.2055	28.7263	–	0.4095	–	11.7648	–	–
	ArchRanker Chen et al. (2014)	1.2325	0.0413	2.9608	29.8600	↑ 3.95%	0.4163	↑ 1.64%	12.4295	↑ 5.65%	11305
	AdaBoost Learning Li et al. (2016)	1.2800	0.0420	2.9928	30.4235	↑ 5.91%	0.4269	↑ 4.25%	12.9892	↑ 10.41%	40782
	BOOM-Explorer Bai et al. (2021)	1.3144	0.0425	3.0603	30.9266	↑ 7.66%	0.4295	↑ 4.87%	13.2830	↑ 12.90%	11426
	Our Method	1.3198	0.0412	3.0334	**32.0724**	**↑ 11.65%**	**0.4351**	**↑ 6.24%**	**13.9542**	**↑ 18.61%**	18729
Mega SonicBOOM	Human Efforts Zhao et al. (2020)	1.6345	0.0592	4.8059	27.6090	–	0.3401	–	9.3896	–	–
	ArchRanker Chen et al. (2014)	1.6109	0.0583	4.7580	27.6436	↑ 0.13%	0.3386	↓ 0.45%	9.3594	↓ 0.32%	20590
	AdaBoost Learning Li et al. (2016)	1.6464	0.0575	4.8015	28.6208	↑ 3.66%	0.3429	↑ 0.82%	9.8139	↑ 4.52%	42929
	BOOM-Explorer Bai et al. (2021)	1.6533	0.0564	4.7291	29.3139	↑ 6.18%	0.3496	↑ 2.80%	10.2481	↑ 9.14%	11426
	Our Method	1.6408	0.0537	4.3759	**30.5684**	**↑ 10.72%**	**0.3750**	**↑ 10.25%**	**11.4617**	**↑ 22.07%**	10556
Giga SonicBOOM	Human Efforts Zhao et al. (2020)	1.6446	0.0715	5.0691	23.0016	–	0.3244	–	7.4626	–	–
	ArchRanker Chen et al. (2014)	1.6140	0.0700	4.9570	23.0742	↑ 0.32%	0.3256	↑ 0.36%	7.5132	↑ 0.68%	11690
	AdaBoost Learning Li et al. (2016)	1.5892	0.0666	4.7926	23.8801	↑ 3.82%	0.3316	↑ 2.21%	7.9186	↑ 6.11%	43624
	BOOM-Explorer Bai et al. (2021)	1.6623	0.0753	5.0859	22.0758	↓ 4.03%	0.3268	↑ 0.74%	7.2154	↓ 3.31%	11426
	Our Method	1.6673	0.0697	4.8883	**23.9207**	**↑ 4.00%**	**0.3411**	**↑ 5.13%**	**8.1587**	**↑ 9.33%**	12667

improved. It suggests that lightweight PPA models can predict PPA values with acceptable accuracy after calibrating with limited sample size. Table 3 demonstrates the accuracy of the lightweight PPA models used by RL. Specifically, we use XGBoost Chen & Guestrin (2016), and more details are demonstrated in Table 8.

4.3 RL TRAINING

Figure 6 shows the scaled PPA values curves in the RL training stage. The average performance (IPC) increase as the training continues, and the average power and area rewards decrease, consistent with Equation (1). Agents for Medium SonicBOOM are trained with more episodes due to the larger design space than Rocket. The curves flatten gradually begin at the 80-th and the 160-th round (A round includes variable episodes controlled by the early-stopping criterion) on two designs, respectively, denoting the convergence of the training.

4.4 COMPARISON W. HUMAN EFFORTS & PRIOR ARTS

We compare the RL solution to various baselines with three metrics, *i.e.*, Perf/Power, Perf/Area, and (Perf × Perf)/(Power × Area) for fairness. Different pre-defined preference vectors are used in the exploration stage of our framework. $[0.125, 0.5, 0.375]$ is utilized for small scales of SonicBOOM (Small and Medium) to pursue high power and area efficiency while maintaining the performance baseline, *i.e.*, 0.76 in IPC for Small SonicBOOM. $[0.6, 0.2, 0.2]$ is applied to large scales of SonicBOOM (Mega and Giga) to emphasize higher performance. $[1/3, 1/3, 1/3]$ is employed to focus on PPA balance for Large SonicBOOM. The reason is that modern multi-core systems integrate dif-

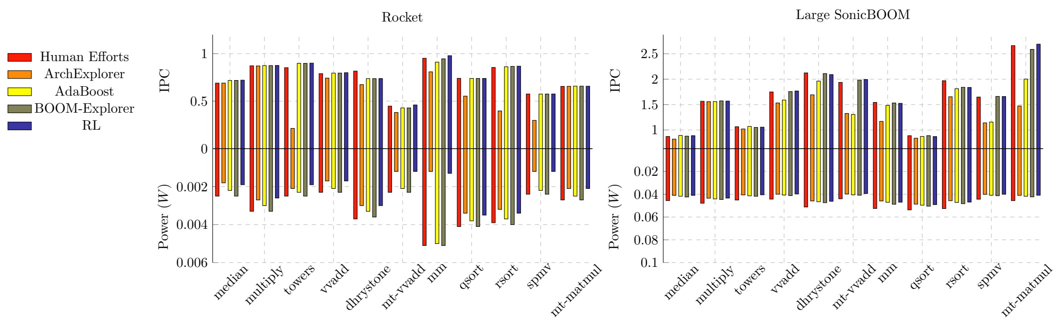


Figure 7: Analysis w. more benchmarks on Rocket and Large SonicBOOM.

ferent scales of processors, *i.e.*, small cores purse high power and area efficiency. In contrast, large cores require high performance and have soft constraints on power and area overhead.

Table 4 shows results compared with human efforts and baselines. The running time (RT) for exploration *w.r.t.* each design, including the RL solution, is also reported in Table 4. The constrained optimization follows the settings in Chen et al. (2014). Random sampling is chosen for Bai et al. (2021) in Rocket. According to Table 4, the explored Rocket and different scales of SonicBOOM by RL are better than prior works, and human efforts, except that the Rocket searched by ICCAD'21 Bai et al. (2021) possess higher area efficiency. In most cases, higher performance with a better balance on power and area is achieved, *e.g.*, Giga SonicBOOM found by RL outperforms human implementations by 1.38% while reducing power dissipation and area overhead by 2.52% and 3.57%. For small SonicBOOM, the solution given by RL meets the expectation, *i.e.*, maintains minimum performance requirements while increasing power and area efficiency as much as possible.

4.5 Analysis w. More Benchmarks

We analyze explored microarchitectures with more benchmarks to study how RL solutions outperform other methods and human implementations. Rocket and Large SonicBOOM are chosen in this section for the analysis.

Figure 7 shows performance (IPC) and power values of Rocket, and Large SonicBOOM with more official RISC-V benchmark suites. Cross all benchmarks, the Rocket found by RL increases 0.83% performance, and reduces 31.92% power dissipation than other solutions. Although Rocket given by ISCA'14 Chen et al. (2014) reduces 32.12% power dissipation compared with human efforts, it sacrifices a 23.79% drop in performance. It is worth noting that the RL solution utilizes a two-way set-associative I-cache, four-way set-associative D-cache, and an unroll pipelined multiplier, *etc.* Prior arts and human efforts use a larger return address stack, larger I-cache size, *etc.*, which bring limited performance improvement while increasing power and area considerably. The Large SonicBOOM given by RL achieves comparable performance while reducing 9.98% more power than human efforts. The solution assigns $1.5\times$ more fetch buffers, $1.3\times$ more fetch target queues, LDQ, STQ, *etc.*, and decreases 50% branch tags, 33% FP registers, and INT issue queue entries, *etc.* to maintain low power dissipation. The mt-vvadd and mt-matmul are multi-threaded benchmarks containing intensive calculations. RL solutioni assign more front-end resources with appropriate back-end resources to help fetch more instructions in parallel. Hence, the RL solution gains 1.07% and 2.93% performance improvement while reducing 10.45% and 10.31% power dissipation on these two benchmarks. We can observe that the RL solutions balance PPA values and retain the performance requirement across all benchmarks by smartly assigning hardware resources.

5 Conclusion

An RL solution is proposed to explore better RISC-V microarchitectures in this work. With an insight that a tiny change can make a big difference step by step for the task, the RL solution outperforms previous arts and human efforts. By incorporating the PPA preference space into RL, the framework can be more practical for industry application. Experiments have verified the method on RISC-V in-order core and different scales of an out-of-order core.

REFERENCES

Axel Abels, Diederik Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. Dynamic weights in multi-objective deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 11–20. PMLR, 2019.

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 Asanović, and Borivoje Nikolić. Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs. *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 40(4):10–21, 2020.

Krste Asanovic, David A Patterson, and Christopher Celio. The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor. Technical report, University of California at Berkeley Berkeley United States, 2015.

Krste Asanović, Rimas Avizienis, Jonathan Bachrach, et al. The Rocket Chip Generator. Technical report, University of California, Berkeley, 2016.

T. Austin, E. Larson, and D. Ernst. SimpleScalar: an Infrastructure for Computer System Modeling. *Computer*, 35(2):59–67, 2002.

Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. Chisel: constructing hardware in a scala embedded language. In *DAC Design Automation Conference 2012*, pp. 1212–1221. IEEE, 2012.

Chen Bai, Qi Sun, Jianwang Zhai, Yuzhe Ma, Bei Yu, and MD Wong. BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–9, 2021.

Nathan Binkert, Bradford Beckmann, Gabriel Black, et al. The Gem5 Simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, 2011.

David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. *ACM SIGARCH Computer Architecture News*, 28(2):83–94, 2000.

David Brooks, Pradip Bose, Viji Srinivasan, Michael K Gschwind, Philip G Emma, and Michael G Rosenfield. New Methodology for Early-stage, Microarchitecture-level Power-performance Analysis of Microprocessors. *IBM Journal of Research and Development*, 47(5.6):653–670, 2003.

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

Christopher Patrick Celio. *A Highly Productive Implementation of an Out-of-Order Processor Generator*. eScholarship, University of California, 2017.

Chen Chen, Xiaoyan Xiang, et al. 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)*, pp. 52–64, 2020.

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

Tianshi Chen, Qi Guo, Ke Tang, Olivier Temam, Zhiwei Xu, Zhi-Hua Zhou, and Yunji Chen. Archranker: A ranking approach to design space exploration. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 1–12, 2014.

Lawrence T Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. ASAP7: A 7-nm FinFET Predictive Process Design Kit. *Microelectronics Journal*, 53:105–115, 2016.

Brian Grayson, Jeff Rupley, Gerald Zuraski Zuraski, Eric Quinnell, Daniel A Jiménez, Tarun Nakra, Paul Kitchin, Ryan Hensley, Edward Brekelbaum, Vikas Sinha, et al. Evolution of the Samsung Exynos CPU Microarchitecture. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 40–51. IEEE, 2020.

Zhuolun He, Yuzhe Ma, Lu Zhang, Peiyu Liao, Ngai Wong, Bei Yu, and Martin DF Wong. Learn to floorplan through acquisition of effective local search heuristics. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pp. 324–331. IEEE, 2020.

Engin Ïpek, Sally A. McKee, Rich Caruana, Bronis R. de Supinski, and Martin Schulz. Efficiently Exploring Architectural Design Spaces via Predictive Modeling. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 195–206, 2006.

Tejas S Karkhanis and James E Smith. A First-order Superscalar Processor Model. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, pp. 338–349. IEEE, 2004.

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

Vinod Kathail, Shail Aditya, Robert Schreiber, B Ramakrishna Rau, Darren C Cronquist, and Mukund Sivaraman. PICO: Automatically Designing Custom Computers. *IEEE Transactions on Computers*, 35(9):39–47, 2002.

Benjamin C Lee and David M Brooks. Illustrative Design Space Studies with Microarchitectural Regression Models. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 340–351. IEEE, 2007.

Chun-Yi Lee and Niraj K Jha. CACTI-FinFET: An Integrated Delay and Power Modeling Framework for FinFET-based Caches Under Process Variations. In *ACM/IEEE Design Automation Conference (DAC)*, pp. 866–871. IEEE, 2011.

Dandan Li, Shuzhen Yao, Yu-Hang Liu, Senzhang Wang, and Xian-He Sun. Efficient design space exploration via statistical sampling and AdaBoost learning. In *ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2016.

Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 469–480, 2009.

Li, Sheng and Chen, Ke and Ahn, Jung Ho and Brockman, Jay B and Jouppi, Norman P. CACTI-P: Architecture-level Modeling for SRAM-based Structures with Advanced Leakage Reduction Techniques. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 694–701. IEEE, 2011.

Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, et al. The GEM5 Simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152*, 2020.

Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *International Conference on Machine Learning (ICML)*, volume 30, pp. 3. Citeseer, 2013.

Philippe Magarshack and Pierre G Paulin. System-on-chip Beyond the Nanometer Wall. In *ACM/IEEE Design Automation Conference (DAC)*, pp. 419–424. IEEE, 2003.

Scott McFarling. Combining Branch Predictors. Technical report, The Western Research Laboratory, 1993.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A Graph Placement Methodology for Fast Chip Design. *Nature*, 594(7862):207–212, 2021.

Volodymyr Mnih, Adria Puigdomenech Badia, et al. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, volume 48, pp. 1928–1937, 2016.

M. Moudgill, P. Bose, and J.H. Moreno. Validation of Turandot, a Fast Processor Model for Microarchitecture Exploration. In *International Performance Computing and Communications Conference (IPCCC)*, pp. 451–457, 1999.

Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. CACTI 6.0: A tool to model large caches. *HP laboratories*, 27:28, 2009.

Avadh Patel, Furat Afram, and Kanad Ghose. Marss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors. In *1st International Qemu Users' Forum*, pp. 29–30. Citeseer, 2011.

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

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

Diederik Marijn Roijers, Shimon Whiteson, and Frans A Oliehoek. Computing Convex Coverage Sets for Faster Multi-objective Coordination. *Journal of Artificial Intelligence Research*, 52:399–443, 2015.

Daniel Sanchez and Christos Kozyrakis. ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems. *ACM SIGARCH Computer architecture news*, 41(3):475–486, 2013.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In Yoshua Bengio and Yann LeCun (eds.), *International Conference on Learning Representations (ICLR)*, 2016.

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

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

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

Matt T Yourst. PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator. pp. 23–34. IEEE, 2007.

Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. SonicBOOM: The 3rd Generation Berkeley Out-of-order Machine. In *Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2020.

Table 5: Components of Rocket

Index	VM	FPU [1]		MUL/DIV [2]			BTB [3]			I-Cache		D-Cache			
		minFLen	fLen	mulUnroll	mulEarlyOut	divEarlyOut	nRAS	nEntries	BHT	nWays	nTLBEntries	nSets	nWays	nTLBEntries	nMSHRs
1	false	0	0	8	true	true	0	0	0	4	32	64	4	32	1
2	true	32	32	1	false	false	6	28	512	1	4	64	1	4	0
3		16	32	8	false	false	3	14	256	2	4	32	1	4	0
4		32	64	4	true	true	12	56	1024	2	16	64	2	4	0
5		16	64	4	false	false	8	32	512	2	32	64	2	16	0
6							3	14	512	1	16	64	2	32	1
7							12	56	2048	4	16	64	1	16	0
8							20	80	2048	2	8	64	4	32	2
9										1	8	64	1	4	2
10										4	8	32	4	32	2
11										4	32	64	2	16	2
12												64	2	16	2
13												64	2	16	4
14												64	4	16	4
15												64	4	16	8

[1] Floating-point process unit.
[2] Multipliers and dividers.
[3] Branch target buffer stores predicted target branch addresses.

Table 6: Components of SonicBOOM I

Index	Fetch Width	Branch Predictor	Decode Width	maxBrCount	LSU [1]		D-Cache			RF [2]	
					LDQ	STQ	nWays	nMSHRs	nTLBWays	PhyINT	PhyFP
1	4	TAGEL	1	6	8	8	4	2	8	52	48
2	8	Gshare	2	8	16	16	8	2	8	80	64
3		BiModal	3	10	24	24	4	4	16	100	96
4			4	12	32	32	8	4	16	128	128
5			5	14	40	40	4	8	32	96	64
6				16	20	20	8	8	32	40	40
7				18	6	6	4	16	48	64	60
8				20	30	30	8	16	48	50	46
9				22	34	34	4	4	8	48	44
10					42	42	8	4	8	72	68
11										90	86
12										132	132
13										46	42

[1] Load store unit.
[2] PhyINT and PhyFP are shorted for the number of integer and floating-point physical registers, respectively.

Table 7: Components of SonicBOOM II

Index	Instruction Fetch Unit [1]		ROB [2]	Instruction Issue Unit [3]								
	Fetch Buffer	FTQ		MEM.IW	MEM.QE	MEM.DW	INT.IW	INT.QE	INT.DW	FP.IW	FP.QE	FP.DW
1	8	16	30	1	8	1	1	8	1	1	8	1
2	16	32	32	1	12	2	2	20	2	1	16	2
3	24	32	36	1	16	3	3	32	3	1	24	3
4	32	40	52	2	24	4	4	40	4	2	32	4
5	40	32	64	2	24	5	5	40	5	2	32	5
6	30	30	80	1	10	1	1	14	1	1	12	1
7	45	40	84	1	10	2	2	14	2	1	12	2
8	4	8	92	1	14	2	2	26	2	1	20	2
9	6	14	96	1	14	3	3	26	3	1	20	3
10	14	30	100	1	20	3	3	36	3	1	28	3
11	30	38	105	2	20	4	4	36	4	2	28	4
12	36	42	110	2	20	5	5	36	5	2	28	5
13	35	40	118	1	4	1	1	4	1	1	4	1
14	32	36	120	1	6	1	1	6	1	1	6	1
15			128	1	12	2	2	28	2	1	20	2
16			130	1	12	3	3	28	3	1	20	3
17			135	1	14	3	3	30	3	1	22	3
18				2	26	4	4	42	4	2	34	4
19				2	26	5	5	42	5	2	34	5

[1] Fetch Buffer and FTQ represents fetch buffer entries and fetch target queue entries, respectively.
[2] Reorder buffer entries.
[3] DW, IW, and QE are shorted for dispatch width, issue width, and queue entries.

APPENDIX

MICROARCHITECTURE DESIGN SPACE

Table 1 and Table 2 only list the indexes to corresponding structures of components. We append the detailed structures of each component in this section.

As shown in Table 5, Table 6 and Table 7, the detailed hardware resources for one component are indexed by an integer listed in Table 1 and Table 2. We include different components into the design space. For example, different branch predictors are deserved to be explored for various PPA design preferences in SonicBOOM. TAGE Seznec & Michaud (2006) prefers high-performance computing scenarios while Gshare McFarling (1993) is more power-efficient and BiModal is more area-efficient. If the BP of SonicBOOM is 1, then the microarchitecture selects TAGE as the branch predictor. The number of physical registers and resources of the instruction issue unit is important for a processor, so various candidates are included in the design space. We manually group the components and prune the design space from invalid or inappropriate combinations, *e.g.*, a small fetch buffer is unlikely to accompany with a large FTQ, a large associative D-Cache does not come with a few translation-lookaside buffers (TLB) entries, *etc.* Some combinations that failed to generate a circuit that can work well are also removed. According to the design space, more than 1×10^5 microarchitectures exist for Rocket Asanović et al. (2016). Since an out-of-order processor is more complicated than an in-order design, SonicBOOM Zhao et al. (2020) has more than 1×10^9 microarchitectures in the design space.

The Very-Large-Scale-Integration (VLSI) Flow

The very-large-scale integration (VLSI) flow is leveraged to evaluate accurate PPA values of each microarchitecture with various EDA tools, as shown in Figure 1. We use Chipyard Amid et al. (2020) to generate different Rocket and SonicBOOM microarchitectures. The generated design are compiled from Chisel Bachrach et al. (2012) to register-transfer-level (RTL) circuit descriptions.

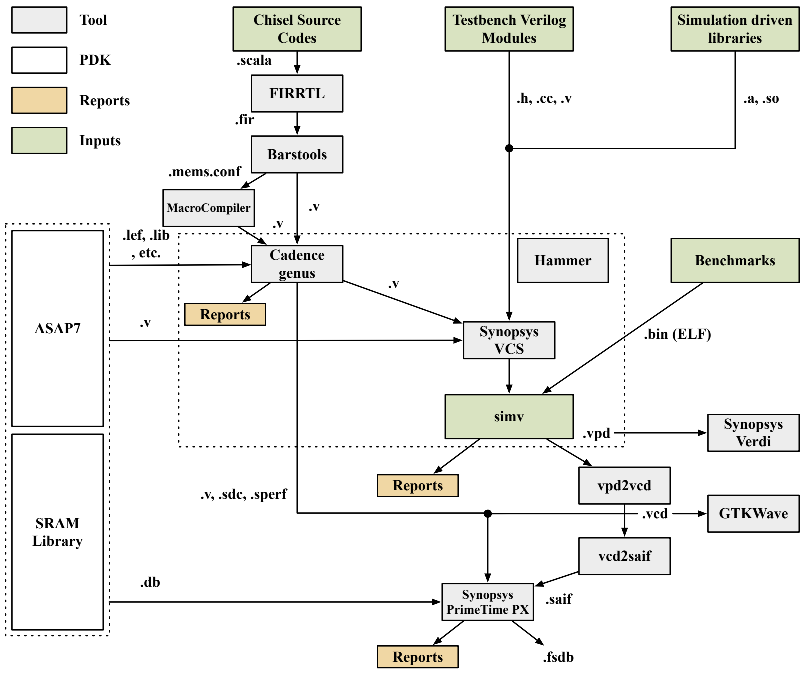


Figure 8: Overview of the VLSI flow.

Figure 8 illustrates the VLSI flow. Firstly, the RTL designs are synthesized with Cadence Genus 18.12-e012_1. We get the area report after mapping the processor circuit to ASAP7 Clark et al. (2016) standard cells. Secondly, we utilize Synopsys VCS M-2017.03 to generate cycle-accurate simulators ("simv" in Figure 8)for the circuit. Different benchmarks are used to evaluate the performance of the simulator. The performance is obtained in this step. Thirdly, Synopsys PrimeTime PX R-2020.09-SP1 is leveraged to get the power report after we finish the simulation in the second step. It costs more than 12 hours to get the PPA values for a single Giga SonicBOOM via the VLSI flow with our high-performance computers. The PPA values are estimated with 1 GHz.

PPA Calibration

In the calibration stage, the accurate PPA values are obtained following the procedure described in Figure 8. In SonicBOOM, with different scales, we use more than a thousand microarchitectures. To get lightweight PPA models, we try various machine learning methods to calibrate against accurate PPA values according to Equation (8). Table 8 lists the calibration results with different methods, and the method is evaluated with ten-fold cross-validation. XGBoost Regression Chen & Guestrin

Table 8: Calibration Results w. Different Methods

Method	Performance		Power		Area	
	MAPE	Kendall τ	MAPE	Kendall τ	MAPE	Kendall τ
Support Vector Regression	10.3413%	0.6835	30.4165%	0.3614	30.1698%	0.7638
LASSO	8.7754%	0.7117	20.6828%	0.4233	4.6288%	0.9088
ElasticNet	8.7156%	0.7121	20.2292%	0.4408	6.0250%	0.8681
KNN Regression	7.4181%	0.7573	16.3281%	0.5531	8.2859%	0.7834
Gaussian Process Regression	6.9500%	0.8046	9.1173%	0.8042	7.6623%	0.8218
Ridge Regression	6.9011%	0.8051	9.1407%	0.8029	4.5526%	0.9094
Linear Regression	6.8322%	0.8053	8.9513%	0.8070	4.5643%	0.9093
Decision Tree Regression	5.4856%	0.8071	7.6297%	0.8052	6.2462%	0.8552
AdaBoost Regression	7.6717%	0.8306	10.5262%	0.8110	7.1186%	0.8527
Random Forest	4.4335%	0.8507	4.8803%	0.8738	4.5679%	0.9006
XGBoost Regression	**4.3197%**	**0.8534**	**3.9327%**	**0.9015**	**3.7300%**	**0.9214**

(2016) achieves the best calibration results in both MAPE and Kendall τ among all methods. Therefore, we adopt XGBoost as the lightweight PPA models.