Trilogy of Microprocessor Microarchitecture Design Space Exploration: Delving into the Depths

BAI, Chen

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

in

Computer Science and Engineering

The Chinese University of Hong Kong

July 2024

Thesis Assessment Committee

Professor XU Hong (Chair)

Professor YU Bei (Thesis Supervisor)

Professor WONG Martin Ding Fat (Thesis Co-Supervisor)

Professor LO Chi Lik Eric (Committee Member)

Professor XIE Yuan (External Examiner)

Abstract

Microprocessor, skilled at general computing, shapes human being's digitalized world. Traditional microprocessor design flow requires high cost and human labor workforce input, including architecture design, microarchitecture design, logic design, circuit design, and physical design stages. Microarchitecture design, at the very early stage of the design flow, can affect the design quality with a small change. Hence, a good microarchitecture generated at the early stage can yield twice the result with half the effort, and benefit the entire microprocessor design flow.

Microarchitecture, also termed computer organization, determines the implementation of a particular microprocessor based on an instruction set architecture (ISA). It sets the foundation for the microprocessor's overarching design aspects: performance, power, and area (PPA). Microarchitecture exploration is a design space exploration (DSE) problem in the microarchitecture design stage, aiming to find microprocessor parameters that achieve Pareto-optimal PPA trade-offs. Computer architects often face parameter combinations ranging in the billions or trillions. Moreover, obtaining PPA results for each parameter combination through detailed simulations is highly time-consuming. To address these challenges, a practical DSE algorithm is required to explore promising solutions within a limited time. Hence, architects can accomplish

aggressive PPA design goals, reduce non-recurring engineering costs throughout the microprocessor design cycle, and meet product delivery deadlines to satisfy customer demands.

The problem is not new. The industry and academia have proposed numerous solutions. In this thesis, standing on the shoulders of giants, we present a series of works termed the trilogy of microprocessor microarchitecture design space exploration to delve deeper into the problem. We provide new perspectives and advance solution quality through our years of recent research outcomes.

Specifically, the trilogy of works includes a customized Bayesian optimization-based DSE algorithm, a reinforcement learning pathway, and an interpretable DSE process via bottleneck analysis. Besides, this thesis introduces a microarchitecture DSE open benchmarking platform, used in the Computer-Aided-Design (CAD) contest of the International Conference on Computer-Aided Design (ICCAD) in 2022, enabling fair comparisons of various black-box methodologies proposed by researchers worldwide. In this thesis, our proposed methodologies cover the breakthroughs from the black-box perspective to the white-box perspective, delving into the problem stepwise.

Furthermore, the trilogy of works is open-sourced to the research community, helping researchers worldwide to reproduce published experimental results and innovate based on our existing achievements.

摘要

勝任通用計算的微理器塑造了人類的數字化世界。 傳統的微處理器設計流程需要高成本和人力投入，包括架構設計，微架構設計，邏輯設計，電路設計和物理設計階段。 微架構設計，在微處理器設計流程的早期階段，只需微小的變化就可以影響設計質量。 因此，在早期生成的一個好的微架構可以事半功倍，並有利於整個微處理器設計流程。

微體系結構，也稱為計算機組織，基於指令集架构（ISA）來確定特定微處理器的實現。 它為微處理器的總體設計目即性能、功耗和面積（PPA）奠定了基礎。微體系結構探索是一個設計空間探索（DSE）的問題，旨在找到實現PPA權衡的帕累托最優的微處理器參數配置。 計算機架構師經常面臨數十億或數萬億的參數組合。 此外，通過詳細的仿真獲得每種參數組合的PPA結果是非常耗時的。 為了應對這些挑戰，需要一種實用的設計空間探索算法用來在有限的時間內探索有前景的解決方案。 因此，計算機架構師可以實現激進的性能、功耗和面積設計目標，降低整個微處理器設計週期中的非經常性工程成本，並在產品交付的截止日期前來完成微處理器設計以滿足客戶需求。

這個問題並不新鮮。 工業界和學術界提出了許多解決方案。 在這篇論文中，站在巨人的肩膀上，我們呈現了一系列工作，被稱呼為微處理器微體系結構設計空間探索三部曲， 以深入探討這個問題。 我們通過多年的最新研究成果為該問題提供

了新的視角並提高了解決方案的質量。

具體而言，這三項工作包括定制的基於貝葉斯優化的設計空間探索算法、基於化學習路徑和通過瓶頸分析的可解釋性設計空間探索算法。 此外，本文還介紹了一個微架構設計空間探索開放基準測試平台。 該平台用於2022年國際計算機輔助設計大會（ICCAD）的計算機輔助設計（CAD）競賽，能夠公平地比較世界各地研究人員提出的各種黑盒方法。 在這篇論文中，我們提出的方法涵蓋了從黑盒方法論發展到白盒方法論的突破，逐步深入研究了這個問題。

此外，這系列三部曲工作向學術界開源，幫助世界各地的研究人員復現已發表的實驗結果，並在我們現有成果的基礎上進行創新。

Acknowledgments

I would like to express my gratitude to my supervisor, Prof Bei YU, for his support and guidance throughout my Ph.D. study. Prof. YU is energetic, intelligent, and amiable. He has inspired me to brainstorm potential research directions, suggested beautiful methodologies, and supported me in pursuing what problems I would like to study. Without Prof. YU, I could never accomplish my thesis based on these years of research outcomes. I learned a lot from Prof. YU, not only in philosophical thinking but also in how to become a better person. I would like to thank my co-supervisor, Prof. Martin Ding Fat WONG, for giving me valuable comments and suggestions. Besides, I would like to thank Prof. Yuan XIE, who has shown me the pathway towards the elite in my career. His ABCDE principle serves as a golden guide, shedding light on the necessities for success.

I would like to thank my thesis committee members for constructive comments. Prof. Herry Xu HONG and Prof. Eric Chi Lik LO recommend how to improve my thesis further.

It is my great honor to work with many talented collaborators during my Ph.D. study. Thanks to Prof. Qi SUN, Prof. Jianwang ZHAI, Prof. Yuzhe MA, Prof. Jiayi HUANG, Dr. Xuechao WEI, Prof. Youwei ZHUO, Dr. Sicheng LI, Dr. Yi CAI, for our intimate collaborations. These collaborations achieve the series of works focusing on research topics from microprocessor microarchitecture design space exploration via black-box or analytical methodologies to DNN orchestration with dataflow architecture accelerators. I also thank my mentors and superiors during my internship, who have provided me valuable industrial data and working experience, and also helped me a lot. Dr. Xuanqi CHEN, Dr. Winnie Wing Yee LO, and Dr. Kevin Kai Yuan

CHAO from Huawei HiSilicon (HK) Turing & Key Technologies Development Department. Dr. Xuechao WEI, Dr. Hongzhong ZHENG, and Dr. Yen Kuang CHEN from computing technology laboratory of Alibaba DAMO Academy. Besides, I would like to thank my labmates for thought-provoking discussions and cooperations. Thanks to Dr. Haoyu YANG, Prof. Tinghuan CHEN, Prof. Hao GENG, Dr. Leon Zhuolun HE, Dr. Ran CHEN, Dr. Wenqian ZHAO, Dr. Binwu ZHU, Mr. Ziyi WANG, Mr. Ziyang YU, Mr. Yuxuan ZHAO, Mr. Xinyun ZHANG, Mr. Shuo YIN, Mr. Peng XU, Mr. Su ZHENG, Mr. Shixin CHEN, Mr. Yuan PU, Mr. Lancheng ZOU, Ms. Yuntao LU and all other CUDA members.

Furthermore, I owe my deepest gratitude to my parents. Their everlasting encouragement and support allow me to pursue somebody I want to be in my lifetime.

This work is dedicated to my loving parents.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The size of present-day applications demands higher performance and energy-efficient computing systems [145]. The microprocessor currently serves as the only commercially available computer architecture that can speed up general computing, including irregular programs [108]. It consists of the arithmetic, logic, and control circuitry to perform the functions of a computer's central processing unit (CPU).

However, the microprocessor design cycle consists of numerous complicated steps, requiring high cost and workforce input, as shown in Figure 1.1. In the design cycle, computer architects formulate customer demands as system specifications, and appropriate computer architecture is conceived to meet those requirements. Microarchitecture is a detailed architecture implementation meticulously designed based on architectural descriptions. Massive formal documents are produced, and reliability, availability, and serviceability (RAS) are heavily discussed [86, 64]. Hardware engineers work on functional and logic implementation based on formal documents to generate the register-transfer-level (RTL) design. Logic synthesis is introduced to

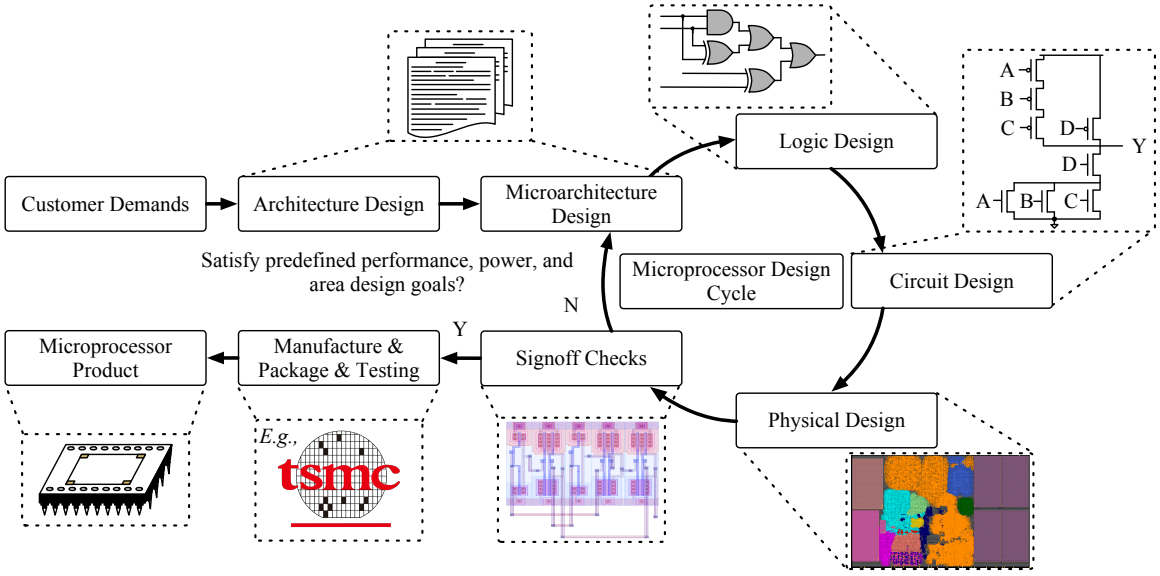


Figure 1.1: The overview of microprocessor design cycle. Engineers spare much effort to optimize a given microarchitecture in the loop. Determine a better microarchitecture can reduce the non-recurring engineering cost [126].

optimize and transfer the RTL design into a circuit implementation, *i.e.*, a netlist composed of primitive standard cells associated with a specific technology [163]. The physical design places the standard cells, routes wires, synthesizes the power delivery network, implements a clock tree, *etc.* [162]. Various signoff checks, including functional verification, timing analysis, IR drop analysis, signal integrity analysis, design rule checking (DRC), layout versus schematic (LVS), *etc*, intensely investigate the validity of the circuit [102]. After back-end signoff, the microprocessor can be taped out and fabricated as a commercial product.

Unfortunately, the ever-increasing competitive pressure from business rivals and the long microprocessor design cycle push higher requirements for engineers to satisfy the strict time-to-market product delivery deadline. The efforts are spared in the design loop until the predetermined overarching metrics: performance, power, and area

(PPA) design targets are attained. In this design cycle, changes made at an earlier design phase have a greater impact on PPA. Meanwhile, accurate estimation of PPA at an earlier design phase is also more difficult. For example, compared to functional engineering change order (ECO), the architecture or microarchitecture design phase sets the cornerstone for PPA [11]. Therefore, a correct decision at the earlier design phase will yield twice the result with half the effort for the entire microprocessor design cycle. Microarchitecture design space exploration (DSE), a problem root in the microarchitecture design phase, aiming to decide optimal microarchitecture configurations, is a pivot stage that affects the entire flow. The problem is not new and has been discussed in industry and academia for many years. Standing on the shoulders of giants, this thesis aims to tackle the problem from new perspectives. Furthermore, this thesis presents a trilogy of works that delve deeper into the problem, providing new insights and advancing solution quality through the years of recent research.

Section 1.1 presents a general problem formulation. Section 1.2 manifests the major challenges of the problem. Section 1.3 lists the thesis structure and contributions.

1.1 A General Problem Formulation

We leverage a general problem formulation as a background. The formulations of related microarchitecture exploration problems stem from the general problem formulation.

Problem 1 (Microarchitecture Design Space Exploration)**.** *Given the microprocessor microarchitecture design space and test cases, how do we efficiently search for the optimal microprocessor microarchitecture design configurations that meet specific*

performance, power, and area (PPA) design goals?

Problem 1 lists the general problem formulation. The solution process expects to develop a search algorithm, termed the DSE algorithm. As shown below, two basic requirements are necessary for the practical "silver bullet" solution.

- Achieving high solution quality: the output results of the DSE algorithm should enable the microarchitecture of a microprocessor to achieve the predetermined PPA design targes.
- Low running time: the DSE algorithm should achieve the goal efficiently.

It is worth noting that more requirements appear for particular problem formulations, *e.g.*, the accuracy of PPA prediction models, the size of used history data, *etc.* In later chapters, this thesis presents more problem formulations based on the general problem formulation, and corresponding solutions are also proposed.

1.2 Major Challenges of Microarchitecture Design Space Exploration

The major challenges of the Problem 1 are summarized in two factors.

- Extremely large design space: the microarchitecture consists of complicated components such as the overall instruction pipeline, cache hierarchy, branch predictors, execution units, *etc.* The formulated microarchitecture design space size can be more than 1×10^{30} [210].

- High runtime in PPA evaluations: it requires high runtime to evaluate the PPA values of a microarchitecture for workloads with an acceptable fidelity using tools like cycle-accurate simulators.

The challenges mentioned above restrict traversing the entire microarchitecture design space and retrieving the optimal solutions. Previous researchers and engineers from industry or academia have proposed many methodologies. The industry solutions heavily rely on the expertise of computer architects. Academic solutions have conducted many experiments to analyze the relationship between a microarchitecture design space and PPA values. As the applications are changing, microarchitectures of modern microprocessors are increasingly complicated, and advanced DSE algorithms are evolving, new solutions are presented in this thesis to overcome the major challenges.

1.3 Thesis Structure and Contributions

Starting from Problem 1, this thesis investigates the methodologies discussed by researchers from industry and academia, aiming to tackle the problem by combining contemporary technologies, improving the solution quality and efficiency, and providing new understandings and insights into the problem.

The rest of this thesis is organized as follows. In Chapter 2, we provide preliminaries of the problem. In particular, we illustrate the meanings of some terminologies, categorize the prior methodologies, summarize the limitations of the state-of-the-arts, and provide rationales for the directions of improvement.

In Chapter 3, we present the first microarchitecture design space exploration

framework, BOOM-Explorer, which focuses on RISC-V Berkeley Out-of-Order Machine (BOOM) with Bayesian optimization. We view the BOOM microarchitecture as a black box and propose several algorithms from a statistics perspective to combine expert knowledge. Three dedicated designed algorithms are highlighted in BOOM-Explorer. First, a microarchitecture-aware active learning algorithm, MicroAL, is introduced to initialize samples from the entire design space with domain knowledge. Second, a deep kernel learning-based Gaussian process model, DKL-GP, is proposed to effectively characterize the microarchitecture design space. Third, a negatively correlated multi-objective exploration flow is provided to probe the potential Pareto-optimal solutions as much as possible. The flow is further enhanced by a diversity-guided strategy to embrace the benefits of parallelism efficiency.

In Chapter 4, we analyze the mathematical limitations of the Gaussian process and its contradiction to the characteristics of the microarchitecture design space. Following the previous data-driven approach, we propose a new methodology but with key distinctions. First, by leveraging the extraordinarily high black-box modeling power of deep learning (DL), we formulate the design space exploration as a Markov decision process (MDP) and propose a reinforcement-learning (RL) pathway toward automated RISC-V microarchitecture design. Second, we tightly couple the expertise and RL utilizing the microarchitecture scaling graph. A single RL agent is trained and applied for the exploration *w.r.t.* different PPA design preferences, *e.g.*, emphasizing higher performance or power and area efficiency.

In Chapter 5, we build up a microarchitecture DSE open benchmarking platform for all researchers worldwide to fairly compare different black-box methodologies. The microarchitecture DSE open benchmarking platform is used in the 2022 CAD

Contest at ICCAD. We serve as the topic chair in the contest. The problem provides challenges of enhancing microprocessor microarchitecture design ability by efficiently trading off PPA metrics in a short time. The contest attracts 166 registered teams from 8 countries/regions. We provide contest benchmarking tools, a post-logic-synthesized dataset including more than ten thousand microarchitectures, and an online team ranking platform. We expect the release and organization of the CAD Contest problem to ignite the research passion for the topic and researchers worldwide to provide more practical, efficient, and accurate solutions to improve baseline algorithms further.

In Chapter 6, we open the black box of the microarchitecture, and tackle the problem with a proposed explanable DSE approach, ArchExplorer. The massive analysis of microexecutions lead to two design principles for ArchExplorer. Besides, we point out the limitations of previous wide-used critical path analysis methodology, and present a new graph representation of microexecution. The new formal graph-based microexecution analysis provides more design insights into the microarchitecture for computer architects. We firmly believe that the research value of the graph representation proposed in ArchExplorer goes beyond the microarchitecture DSE problem. The related formal microexecution analytical methodology will guide us in studying other problems, including microprocessor performance modeling, instruction issuing, a combination of machine-learning techniques for architecture modeling, *etc.*

In Chapter 7, we provide future work and conclude this thesis.

The contributions of this thesis are briefly summarized as follows. First, during years of recent research into this problem, we propose methodologies from the black-box perspective to the white-box perspective. From data-driven statistical analysis

to interpretable formal analysis, we delve into the problem. The related experimental results, discussions, and conclusions provide abundant design insights into microarchitecture and microexecutions. Second, our latest research outcomes receive promising experimental results. Compared to previous analytical methodologies, we reduce demands for expert access. Compared to previous black-box methodologies, our approaches ask for fewer computing demands for training data. Third, we provide a new graph representation methodology of formal analysis for microexecutions. Fourth, we open source all the code to help our community reproduce published experimental results, research into the problem deeper, and continue to innovate based on our existing achievements.

Chapter 2

Preliminaries

2.1 Microprocessor Microarchitecture

Microarchitecture determines how a microprocessor is implemented given an instruction set architecture (ISA). It encompasses the intricate details of how a microprocessor carries out instructions, manages data flow, and executes computations.

The microprocessor microarchitecture is usually represented as block diagrams that describe the memory hierarchy, interconnections of the various components, including the data path and control path, and timing information of the instruction pipeline, as shown in Figure 2.1. We use MIPS R10000, a dynamic, super-scalar microprocessor that implements the 64-bit MIPS ISA, as an example to illustrate the execution of a microarchitecture [206]. R10000 is designed for high-performance applications even with poor memory locality. It fetches and decodes four instructions per cycle and dynamically issues them to five fully pipelined, low-latency execution units. The five execution pipelines begin to execute when they receive instructions is-

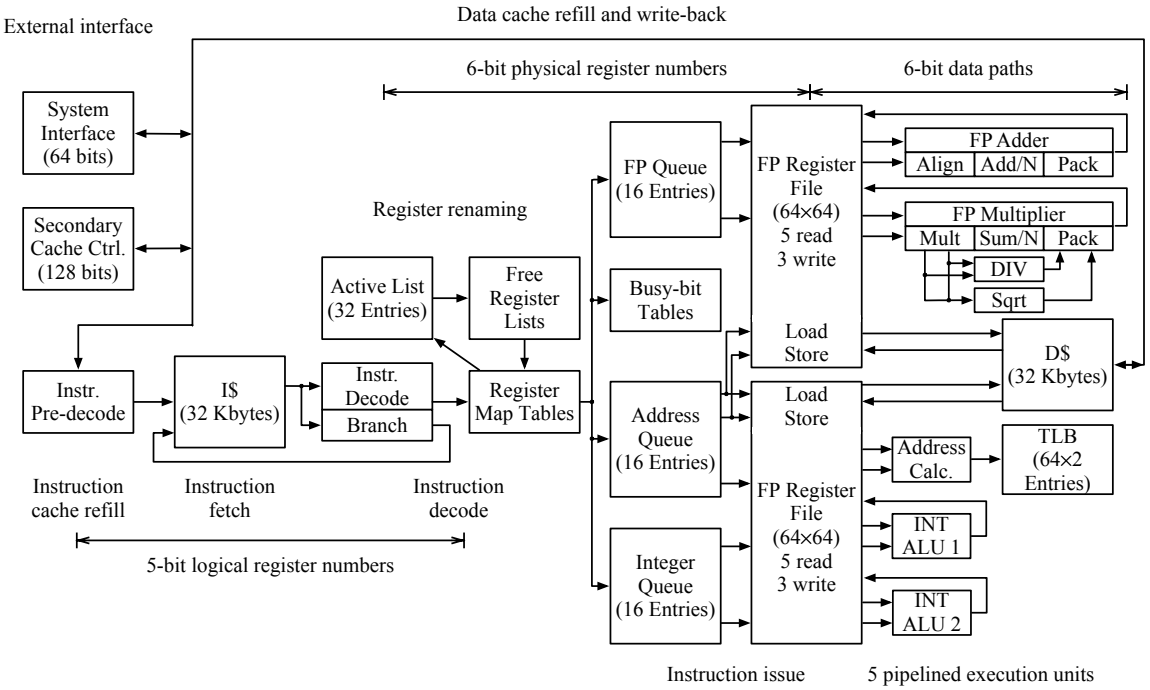


Figure 2.1: Pipeline timing diagram of MIPS R10000. Figure is adapted from Yeager *et al.* [206]. BOOM is a RISC-V implementation based on R10000 [22, 44, 214].

sued from a queue, and operands are read out from register files. R10000 aggressively looks ahead to find instructions from a two-way associative I-cache to issue out-of-order. Misaligned instructions are handled, and the core front end implements a 2-bit branch prediction algorithm. To support the out-of-order execution and remove false dependencies (*viz.*, write after write and write after read), R10000 dynamically maps the logical register numbers into physical register numbers. The mechanism allows for the transparent tracking of dependencies on register operands and memory addresses, *i.e.*, it is invisible to programmers. To achieve register renaming, R10000 implements register map tables, free lists, active lists, and busy-bit tables. Decoded instructions are put into three instruction queues *w.r.t.* the instruction type. R10000 schedules instructions based on instructions' locations in the queue instead of instruction age.

Besides, the round-robin request circuit can raise the priority for old instructions. Two integer algorithm logical units (ALU), a floating-point adder, a floating-point multiplier, a floating-point divide and square root serve as the pool of functional units. The load/store unit decouples the functional unit and D-cache. R10000 archives in-order instruction commits for precise interruptions [181] and maintains cache coherence using snoopy or directory-based protocols.

Different components include various design options. For example, the number of rows, set associations, and line size together determines a cache design's compulsory miss, capacity miss, and conflict miss [84]. The instruction fetch width and decode width decide the pipeline width of a microarchitecture. The number of re-order buffer entries can constrain the number of in-flight instructions in the pipeline, *i.e.*, the instruction window that microarchitecture used to track and extract instruction-level parallelism. In addition, how many physical registers, number of specific functional units, and depth of issue queues are required in the microarchitecture? Different applications require diverse trade-offs between PPA values. Hence, the optimal configurations for these components remain to be answered.

2.2 Representative RISC-V Microprocessor Implementations

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 [23] is a six-stage pipeline in-order microprocessor. SonicBOOM (often shorted as BOOM) [214] is a ten-stage pipeline out-of-order design.

XiangShan [200] features advanced microarchitecture optimizations. Xuantie-910 [46] is an open-source implementation from the industry.

BOOM is an open-source superscalar out-of-order RISC-V microprocessor in academia, and it has proved to be industry-competitive in low-power, embedded application scenarios [22, 44, 214].

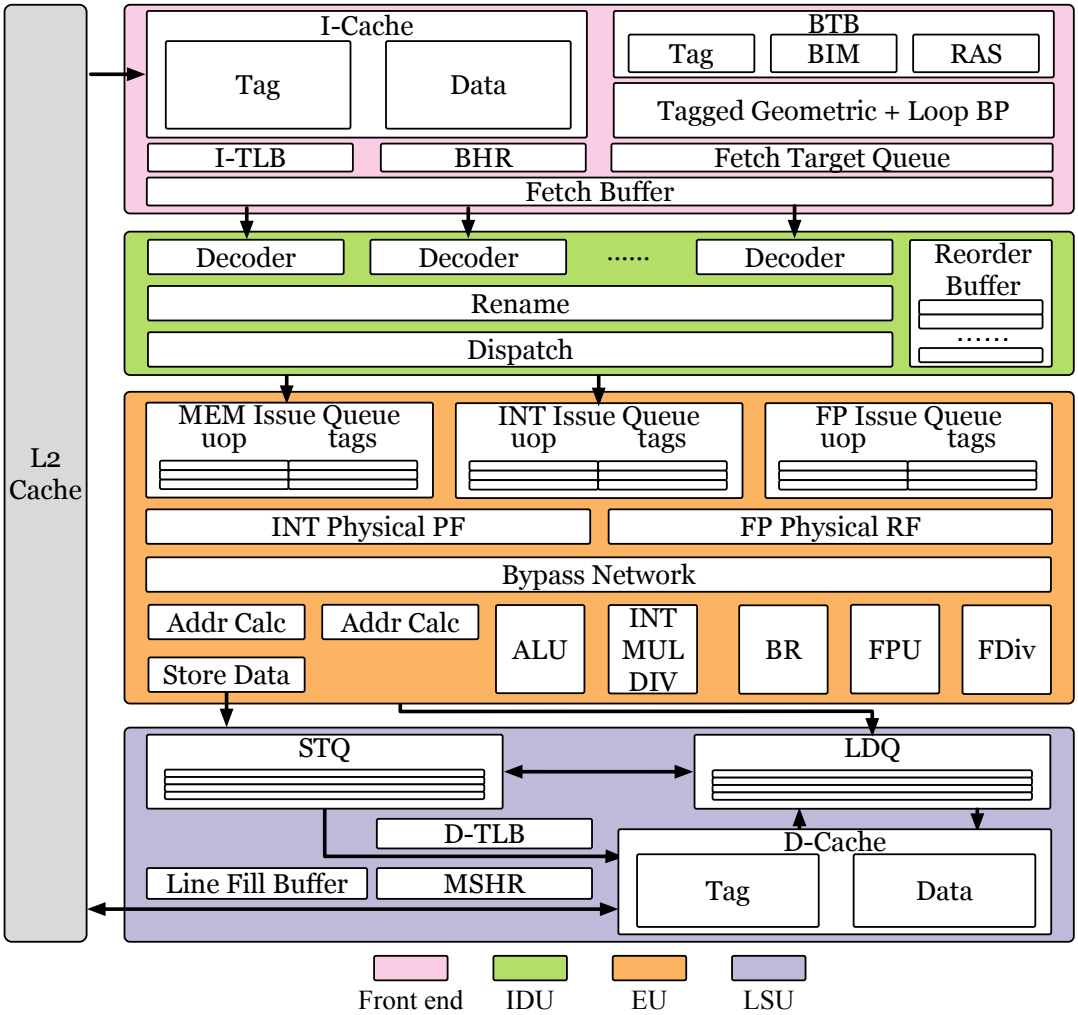


Figure 2.2: BOOM implements a ten-stage pipeline that includes *Fetch*, *Decode*, *Register Rename*, *Dispatch*, *Issue*, *Register Read*, *Execute*, *Memory*, *Writeback*, and *Commit*.

Figure 2.2 demonstrates the organization of BOOM. BOOM is mainly composed

of a front end (FrontEnd), an instruction decoding unit (IDU), an execution unit (EU), and a load-store unit (LSU). FrontEnd fetches instructions from I-cache, predicting branch target addresses, handling return addresses, and packing consecutive instructions as a fetch packet to the fetch buffer. IDU decodes instructions retrieved from the fetch buffer as micro-ops, and dispatch, schedule, and issue them according to instruction types. EU integrates various functional units, including dividers, multipliers, accelerator interfaces, *etc.* LSU interacts with EU and D-cache, deciding when to fire memory instructions to D-cache. BOOM implements explicit renaming logic, short-forward branch optimizations, an advanced branch predictor [172], a loop branch predictor [171], *etc.*, to improve its overall performance. With high-level hardware description language like Chisel [26], many components and their connections can be parameterized to support various BOOM microarchitectures. For example, the parameterization of the size of the branch target buffer [152], return address stack [180], branch prediction history tables [45], *etc.*, broadens BOOM's potential to balance the performance and power dissipation in FrontEnd. We can achieve divergent trade-offs by adjusting these parameters to meet design requirements involving low-power and embedded computation applications.

2.3 Literature Review

Regarding the problem, researchers and engineers have discussed and proposed many methodologies.

The industrial solutions primarily rely on the expertise of computer architects. Simulation environments and PPA models are built for the design space exploration

[133, 39, 38]. Computer architects decide the optimal microarchitecture configurations based on the experience of historical commercial products and massive analysis reports from those internal simulation results. IBM T.J. Watson Research Center developed Turandot, a trace-driven parameterized microprocessor model, and Aria, an execution-driven trace generator [132, 133]. The simulation environments support early exploration for microarchitecture issue width, branch prediction algorithm, cache configurations, *etc.* The PowerTimer toolset has also been developed for use in the early-stage microarchitecture power-performance analysis.[39]. The merits of PowerTimer are the relatively accurate power models derived from analytical equations with circuit and technology parameters. Hewlett-Packard Laboratories launched the PICO (program in, chip out) project to automate the design of computer systems [103]. PICO integrates a parameterized architectural template, a constructor to generate design from a component library, and an evaluator to measure the quality of the design [14, 167].

In academia, the methodologies can be categorized as white-box and black-box approaches. The white-box methodologies investigate the relations between the microarchitecture design space and PPA values and construct interpretable equations or rely on graph-based models to characterize [111, 129, 101, 116, 93, 75]. The interpretable equations are built from a first-order microprocessor performance model or throughput model [141, 100, 62, 185, 63, 43], analytical power model [197, 40, 41, 134, 128, 119, 98], and area model [135, 178]. For example, Karkhanis *et al.* proposed a design framework that takes as inputs the design targets, design alternatives, and one or more application programs [101]. The framework output is the set of out-of-order superscalar microprocessors that are Pareto-optimal *w.r.t.* PPA values. On the other

hand, the graph-based models are from critical path analysis, which is leveraged to unveil the bottlenecks of a microarchitecture during microexecutions [68, 67, 69, 137**?** , 156]. The insights into the microarchitecture derived from the graph-based models can be utilized as exploration guidance [116, 143, 144, 186, 75, 27]. Lee *et al.* identify the microarchitecture's key performance bottlenecks and estimate the exact impacts of latency adjustments without launching an extra step of simulations [116]. The related methodologies are extended to multi-core microprocessors [93].

The black-box methodologies reduce the demand for microexecution analysis and view the microarchitecture as a black box. With machine learning-based (ML) methods, black-box methodologies construct PPA prediction models and conduct design space exploration for microarchitectures [91, 94, 115, 59, 92, 48, 118, 28, 120, 29, 210, 31]. İpek *et al.* proposed an exploration algorithm with artificial neural networks [130]. The models produce highly accurate performance estimates for unknown microarchitectures in the design space, *i.e.*, achieving $4 \sim 5\%$ average error using 1% samples from the complete design space [91]. Lee *et al.* provided a non-linear regression solution to find Pareto-optimal solutions in the microarchitecture design space [114, 115]. Different non-linear operators are proposed, including square-root transformation, log transformation, cubic splines, *etc.* ArchRanker argues that the information computer architect mostly needs during the DSE process is whether a given configuration will perform better than another rather than precisely estimating the performance of that configuration [48]. Therefore, ArchRanker proposed a DSE algorithm based on RankBoost [70]. RankBoost, an ML model combining ensemble learning and boosting algorithms, is employed to rank predicted PPA values of different microarchitectures from the design space [215, 166]. Li *et al.* leverage the

orthogonal array (OA) sampling [65] and ActBoost algorithm [179] to conduct the DSE for the GEM5 [36] microarchitecture design space [118].

Chapter 3

BOOM-Explorer

3.1 Introduction

Recently, RISC-V, an open-source instruction set architecture (ISA), gained much attention and received strong support from academia and industry. Berkeley Out-of-Order Machine (BOOM) [22, 44, 214], a RISC-V design fully compliant with RV64GC instructions [4, 5], is competitive in power and performance against low-power, embedded out-of-order microprocessors in academia. By adopting Chisel hardware construction language [26], BOOM can be parametric, providing great opportunities to explore a series of microarchitecture designs that better balance power and performance for different purposes of use.

Microarchitecture defines the implementation of an ISA in a processor. Due to different organizations and combinations of components inside a microprocessor, microarchitecture designs under a specific technology process can affect a microprocessor's performance, power dissipation, chip area, etc. [159, 78]. Finding a good

microarchitecture that can accommodate a good balance between power and performance is a notorious problem because of two restrictions. On the one hand, the design space is extremely large. Its size can be exponential with more components to consider, *e.g.*, special queues [155], buffers [181], branch predictors [207], vector execution units [109], external co-processors [23], *etc.* Thus, we cannot traverse and evaluate each microarchitecture to retrieve the best one. On the other hand, acquiring metrics, *e.g.*, power, performance, *etc.*, costs a lot of time when we verify one microarchitecture with diverse benchmarks.

In industry, the traditional solution is based on prior engineering experience from computer architects [133]. However, it lacks scalability for newly emerged processors. In academia, to overcome these two obstacles, researchers proposed various arts, which can be categorized as two kinds of methodologies. First, in view of the difficulty in constructing an analytical model, researchers can otherwise characterize a microarchitecture design space with fewer samples as much as possible by leveraging statistical sampling and predictive black-box models. Li *et al.* [118] proposed AdaBoost Learning with novel sampling algorithms to explore the design space. Second, to search for more designs within a limited time budget, researchers often rely on coarse-grained simulation infrastructure rather than a very-large-scale-integration (VLSI) flow for register-transfer level designs (RTL) to accelerate the process [132, 24, 39, 36]. Moreover, the simulation can be further sped up by decreasing redundant overhead [151, 106, 33, 66].

Unfortunately, both of these academic solutions contain several limitations. In the first place, despite the fact that statistical analysis performs well when highly reliable prediction models can be constructed, it fails to embed prior knowledge of microar-

chitectures to improve design space exploration further [91, 59, 115]. For another, coarse-grained simulation infrastructure is used widely to accelerate the simulation. Nevertheless, most of them lose sufficient accuracy, especially for distinct processors [80, 18]. The low quality of results is often generated due to the misalignment between simulation and real running behaviors of processors. More importantly, because it is difficult to model the power dissipation of modern processors at the architecture level [119], some infrastructure cannot provide power value [24, 36]. Because of the aforementioned limitations, academia lacks sufficient discussions on methodologies that can explore microarchitecture designs, achieving a good trade-off between power and performance.

In this chapter, following the first strategy, we propose a BOOM-Explorer framework to address these issues. In BOOM-Explorer, without sacrificing the accuracy of a predictive model, we embed prior knowledge of BOOM to form a microarchitecture-aware active learning (MicroAL) algorithm based on transductive experimental design by utilizing BOOM RTL samples among the entire design space as few as possible [209]. Secondly, a novel Gaussian process model with deep kernel learning functions (DKL-GP) initialized through MicroAL is proposed to characterize the features of different microarchitectures. The design space is then explored via correlated multi-objective Bayesian optimization flow based on DKL-GP [183]. Our framework can not only take advantage of fewer microarchitecture designs as much as possible but also help us find superior designs with a better balance between power and performance.

Our contributions are summarized as follows:

- A microarchitecture-aware active learning methodology based on transductive experimental design is introduced to attain the most representative designs from

an enormous RISC-V BOOM design space.

- A novel Gaussian process model with deep kernel learning and correlated multi-objective Bayesian optimization is leveraged to characterize the microarchitecture design space. With the help of DKL-GP, Pareto optimal solutions for performance and power are explored.

- We verify our framework with BOOM under advanced 7-nm technology. The experimental results demonstrate the outstanding performance of BOOM-Explorer on various BOOM microarchitectures.

The remainder of this chapter is organized as follows. Section 3.2 presents preliminaries for BOOM-Explorer. Section 3.3 provides detailed explanations of the framework. Section 3.4 conducts several experiments on BOOM microprocessor to confirm the outstanding performance of the proposed framework. Finally, Section 3.5 summarizes this chapter.

3.2 Preliminaries

In this section, we present some preliminaris for BOOM-Explorer, including our microarchitecture design space, Bayesian optimization, and our problem formulation based on Problem 2.

3.2.1 RISC-V BOOM Microarchitecture Design Space

As introduced in Section 2.2, microarchitecture design space of BOOM is constructed according to BOOM's overall architecture, as listed in Table 3.1. The design space

of each module is composed of various components, the structure of which can affect the performance power trade-off and deserve to be optimized. As shown in Table 3.1, different entries of RAS (RasEntry), branch counters (BranchCount), organization of I-cache/D-cache (associativity, block width, and size [84]), translation lookaside buffer (TLB) structures are considered in this section. The column of "Candidate values" in Table 3.1 denote supported hardware resources. For example, the reorder buffer is provided to support 32, 64, or 96 entries, and so on. Performance and power are negatively correlated. Assigning more hardware resources often improves performance and brings considerable power dissipation. Hence, a good microarchitecture demands an appropriate compromise across all components' hardware resources.

Some combinations in Table 3.1 are illegal. A legal combination should observe the constraints of BOOM design specifications as in Table 3.2. Otherwise, it would fail to generate reasonable RTL designs. For example, DecodeWidth defines the maximal instructions to be decoded simultaneously. If a BOOM microarchitecture breaks rule 2 in Table 3.2, the reorder buffer may not reserve enough entries for each decoded instruction or contain redundant entries that we cannot fully utilize. The last three rules in Table 3.2 are added to simplify the design space. Their incorporations do not affect the design of a design space exploration (DSE) algorithm. After we prune the design space *w.r.t.* rules in Table 3.2, the size of the legal microarchitecture design space is approximately 1.6×10^8.

3.2.2 Bayesian Optimization

Bayesian optimization [213, 183] is widely applied in design space exploration problems when an evaluation process like the VLSI flow is expensive, as shown in Equa-

Table 3.1: Microarchitecture design space of BOOM.

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

Table 3.2: Constraints of BOOM design specifications.

Rule	Descriptions
1	FetchWdith $\geq$ DecodeWidth
2	RobEntry | DecodeWidth [+]
3	FetchBufferEntry > FetchWidth
4	FetchBufferEntry | DecodeWidth
5	fetchWidth = 2$\times$ ICacheFetchBytes
6	IntPhyRegister = FpPhyRegister
7	LDQEntry = STQEntry
8	MemIssueWidth = FpIssueWidth

[+] "|" means RobEntry should be divisible by DecodeWidth.

tion (3.1), where we use minimization as an example.

$$x^* = \arg\min_{x \in \mathcal{X}} f(x), \tag{3.1}$$

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

Bayesian optimization consists of a *surrogate model* and an *acquisition function*. A surrogate model is often constructed from the Gaussian process (GP) [195]. It models f in Equation (3.1) as a generated probability distribution from the GP. The acquisition function is designed to characterize the relative rankings between differ-

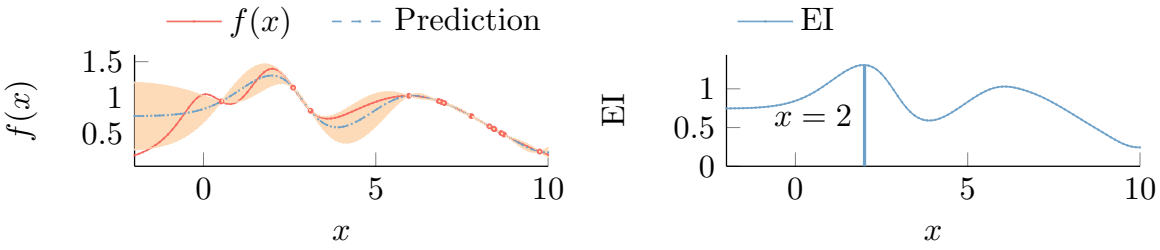


Figure 3.1: An example of Bayesian optimization (with the EI function): find x, which attains the maximal value of $f(x)$.

ent solutions based on the probability distribution. In other words, the acquisition function answers whether x_1 is better than x_2 without considering exactly how much better x_1 performs than x_2. The solution that achieves the acquisition function's optimum is selected as a potential promising/optimal solution. The potential optimal solutions are then evaluated using the VLSI flow and their corresponding performance and power values are utilized to tune the surrogate model. Better solutions are expected to be explored using the tuned surrogate model with a more accurate modeled probability distribution. Suppose $\boldsymbol{x}$ denotes a microarchitecture embedding (**??**), τ denotes the current explored best performance/power value. GP characterizes the evaluation flow with the mean $\mu(\boldsymbol{x})$ and the covariance $\sigma^2(\boldsymbol{x})$. The expected improvement (EI), a popular acquisition function, is shown in Equation (3.2),

$$\begin{aligned} \mathrm{EI}(\boldsymbol{x}) &= \mathbb{E}[\min(f(\boldsymbol{x}) - \tau, 0)] \\ &= \mathbb{E}[\min(\frac{f(\boldsymbol{x}) - \mu(\boldsymbol{x}))}{\sqrt{\sigma^2(\boldsymbol{x})}} - \frac{\tau - \mu(\boldsymbol{x})}{\sqrt{\sigma^2(\boldsymbol{x})}}, 0)] \cdot \sqrt{\sigma^2(\boldsymbol{x})} \\ &= \sigma(\boldsymbol{x})(\lambda\Phi(\boldsymbol{x}) + \phi(\boldsymbol{x})), \\ \lambda &= \frac{\tau - \xi - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}, \end{aligned} \tag{3.2}$$

where $\Phi(\boldsymbol{x})$ and $\phi(\boldsymbol{x})$ are the cumulative distribution function and the probabil-

ity density function of GP, and ξ is a coefficient to improve the numerical ability. Figure 3.1 visualizes the optimization procedure for maximizing $f(x)$ with Bayesian optimization. A surrogate model with GP is built in the $x - f(x)$ space from already-known samples represented by red dots. The red curve denotes the golden values (ground truths) for each input x, and the dashed blue curve denotes GP predictions. The band colored in orange shows the GP prediction uncertainty of each x. If the band is wider, the uncertainty is larger. The potential optimal solution is selected according to EI. Namely, point $x = 2$, which achieves the maximal EI, is chosen as a potential optimal solution, as shown in the $x -$ EI space.

3.2.3 Problem Formulation for BOOM-Explorer

Definition 1 (Microarchitecture Embedding)**.** *Microarchitecture embedding is a feature vector $\boldsymbol{x}$, denoting a combination of candidate values given in Table 3.1, and it satisfies all constraints, as referred to in Table 3.2.*

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

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

With the same benchmark, clock cycle and power are a pair of trade-off metrics. The lower the cycles, the more power will dissipate when a design integrates more hardware resources to accelerate instructions execution. Together, they reflect on whether a microarchitecture design is good or not. Power and clock cycle are denoted as $\boldsymbol{y}$.

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

$$\begin{aligned} &\forall i \in [1,n], \quad f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{x}'); \\ &\exists j \in [1,n], \quad f_j(\boldsymbol{x}) < f_j(\boldsymbol{x}'), \end{aligned} \tag{3.3}$$

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

This section aims to explore Pareto optimality defined in Definition 4 *w.r.t.* clock cycle and power for various BOOM microarchitectures. A microarchitecture whose objective values belong to the Pareto frontier cannot improve performance without sacrificing power, and vice versa. Based on the above definitions, we formulate the problem.

Problem 2 (BOOM Microarchitecture Design Space Exploration)**.** *Given a design space $\mathcal{D}$, each microarchitecture embedding $\boldsymbol{x} \in \mathcal{D}$ corresponds to objective values $\boldsymbol{y}$ in the clock cycle-power space $\mathcal{Y}$. BOOM microarchitecture design space exploration is to find the Pareto-optimal microarchitecture embeddings set $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$, whose objective values $\boldsymbol{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_n\} \subset \mathcal{Y}$ formulate the Pareto frontier.*

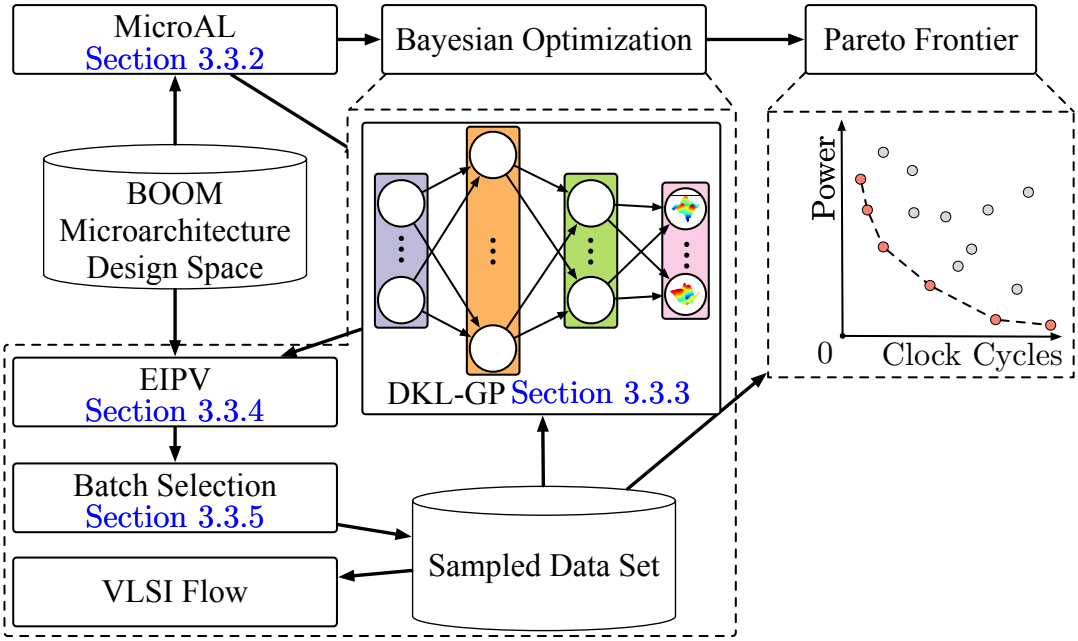


Figure 3.2: Overview of the proposed BOOM-Explorer.

3.3 The BOOM-Explorer Framework

3.3.1 Overview of BOOM-Explorer

Figure 3.2 shows an overview of BOOM-Explorer. Firstly, the active learning algorithm MicroAL (Section 3.3.2) is adopted to sample a set of initial microarchitectures from the design space. Domain-specific knowledge is embedded in the initialization based on the first learned lesson. Secondly, a Gaussian process model with deep kernel learning (DKL-GP) (Section 3.3.3) is then built on the initial set as a surrogate model. Thirdly, the expectation improvement of Pareto hypervolume (Section 3.3.4) is applied as the acquisition function. During the DSE procedure, BOOM-Explorer interacts with the VLSI flow to acquire performance and power values for the selected microarchitectures. Due to the customized algorithm flow, such interactions

Algorithm 1 TED $(\mathcal{U}, \mu, b)$

Require: $\mathcal{U}$ is the unsampled microarchitecture design space, μ is a normalization coefficient, and b is the number of samples to draw.
Ensure: $\boldsymbol{X}$: the sampled set with $|\boldsymbol{X}| = b$.
1: $\boldsymbol{X} \leftarrow \emptyset$, $\boldsymbol{K}_{\boldsymbol{uu'}} \leftarrow f(\boldsymbol{u}, \boldsymbol{u'}),\ \forall \boldsymbol{u}, \boldsymbol{u'} \in \mathcal{U}$;
2: **for** $i = 1 \rightarrow b$ **do**
3: $\quad \boldsymbol{x}_* \leftarrow \arg\max_{\boldsymbol{x} \in \mathcal{U}} \mathrm{Tr}[\boldsymbol{K}_{\mathcal{U}\boldsymbol{x}}(\boldsymbol{K}_{\boldsymbol{xx}} + \mu \boldsymbol{I})^{-1} \boldsymbol{K}_{\boldsymbol{x}\mathcal{U}}]$; ▷ $\boldsymbol{K}_{\mathcal{U}\boldsymbol{x}}$, $\boldsymbol{K}_{\boldsymbol{xx}}$ and $\boldsymbol{K}_{\boldsymbol{x}\mathcal{U}}$ are calculated via f *w.r.t.* corresponding columns in $\boldsymbol{K}$.
4: $\quad \boldsymbol{X} \leftarrow \boldsymbol{X} \cup \boldsymbol{x}_*,\ \mathcal{U} \leftarrow \mathcal{U} \setminus \boldsymbol{x}_*$;
5: $\quad \boldsymbol{K} \leftarrow \boldsymbol{K} - \boldsymbol{K}_{\mathcal{U}\boldsymbol{x}_*}(\boldsymbol{K}_{\boldsymbol{x}_*\boldsymbol{x}_*} + \mu \boldsymbol{I})^{-1} \boldsymbol{K}_{\boldsymbol{x}_*\mathcal{U}}$;
6: **end for**
7: **return** The sampled set $\boldsymbol{X}$;

make the DSE at the RTL level feasible. Moreover, we improve the exploration with diversity-guided sampling to handle different sub-regions (Section 3.3.5). Finally, The outputs of BOOM-Explorer are the predicted Pareto frontier and corresponding Pareto-optimal microarchitectures.

3.3.2 Microarchitecture-aware Active Learning Algorithm

The initial microarchitectures sampled from the design space are critical to constructing a surrogate model for later exploration. A naive solution is to sample microarchitecture embeddings randomly [91, 115]. Some advanced sampling techniques with statistic analysis like orthogonal design [118] are also applied in previous works. Nevertheless, the methods mentioned above are less effective since they require many VLSI flow interactions while the runtime cost of the VLSI flow is much higher. We follow two guidelines to design the initialization algorithm. First, the initial microarchitecture embeddings should uniformly scatter in the entire design space. Different PPA trade-offs should be captured via uniform scattering. Second, the diversity of

these microarchitecture embeddings should represent different characteristics of the design space as much as possible. Simple characteristics can be described with a few samples, while complex ones can be described with more samples. Besides, combining with prior knowledge helps us remove samples unworthy to be evaluated with the VLSI flow.

In MicroAL, we first perform clustering *w.r.t.* the decode width (DecodeWidth), then we conduct transductive experimental design with samples per each cluster. DecodeWidth in IDU, as referred to in Table 3.1, decides the maximal numbers of instructions that can be decoded simultaneously, *i.e.*, it determines the width of the pipeline, as shown in Figure 2.2. It allows more instruction execution parallelism in the pipeline if DecodeWidth is assigned a considerable value in BOOM, and we also allocate appropriate hardware resources to other components accordingly. Although large DecodeWidth leads to remarkable performance improvement, in most cases, power dissipation increases correspondingly due to more transistors integrated. We find that the impact of a PPA trade-off by configuring DecodeWidth is significant.

Figure 3.3 visualizes the objective space by clustering *w.r.t.* DecodeWdith on sampled microarchitecture embeddings. Better microarchitectures will be closer to the original point in Figure 3.3, indicating higher performance and lower power dissipation. Figure 3.3 demonstrates that the distributions of clusters are highly correlated with the potential Pareto frontier. The entire design space is discrete and non-smooth. Nonetheless, many microarchitectures with the same DecodeWidth achieve similar performance-power characteristics within their clusters. We embed this domain knowledge in the initialization algorithm. Additionally, within each cluster, we adopt the *transductive experimental design* (TED) [209] to sample microarchitectures

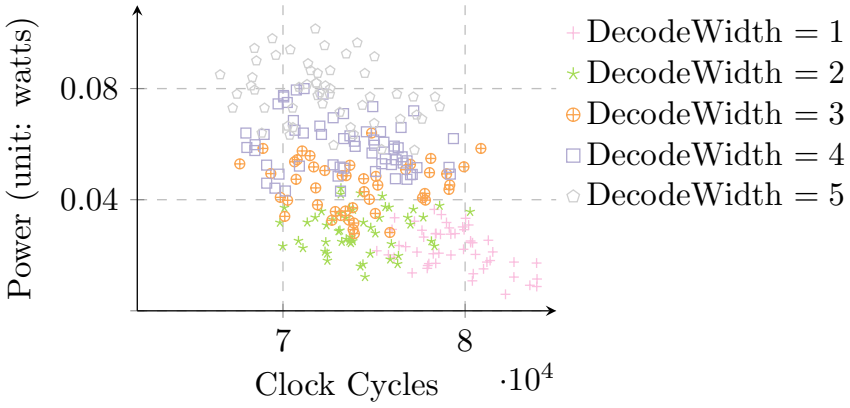


Figure 3.3: Visualization of clusters *w.r.t.* DecodeWidth.

that best reflect the characteristics of their groups.

The method of TED is a widely used algorithm to construct samples that deserve to be evaluated with an expensive flow. It tends to choose microarchitecture embeddings that can spread across the design space to retain the most information [209]. We can acquire a pool of representative samples with high mutual divergences by iteratively maximizing the trace of the distance matrix constructed on newly sampled and unsampled microarchitecture embeddings. Algorithm 1 shows the backbone of TED, where f represents the distance function used in computing the distance matrix K. It should be noted that there are no restrictions on the choice of distance functions. The microarchitecture embeddings can be clustered based on DecodeWidth, allowing us to identify clusters that are most closely related to the Pareto frontier. Within each cluster, TED is performed to create representative samples. This process leads to the formulation of MicroAL as described in detail in Algorithm 2.

In Algorithm 2, firstly, we cluster the entire design space according to Φ, which is the distance function with a higher penalty along the dimension of DecodeWidth. One possible alternative can be $\Phi = (\boldsymbol{x}_i - \boldsymbol{c}_j)^\top \boldsymbol{\Lambda} (\boldsymbol{x}_i - \boldsymbol{c}_j)$, with $i \in \{1, \cdots, |\mathcal{U}|\}$ and

Algorithm 2 MicroAL $(\mathcal{U}, \mu, b, n)$

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

Ensure: $\boldsymbol{X}$: the initial set with $|\boldsymbol{X}| = b$.

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

$j \in \{1, \cdots, k\}$, where $\boldsymbol{\Lambda}$ is a pre-defined diagonal weight matrix. The procedure runs n rounds to make the design space sufficiently clustered. Secondly, we apply TED to each cluster to perform sampling, *i.e.*, line 10 in Algorithm 2. Finally, the initial microarchitectures that are worth to be estimated with the VLSI flow are constructed. Each microarchitecture is estimated using the VLSI flow. Our acquisition function is built based on these microarchitectures and their respective performance and power values.

3.3.3 Gaussian Process with Deep Kernel Learning

It is common to build GP models for the initial microarchitectures due to the non-parametric approximation in terms of reliability in uncertainty estimation, and robust performance for many applications [124, 125].

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

$$\boldsymbol{Y} = \begin{pmatrix} y_{11} & y_{12} & \cdots & y_{1m} \\ y_{21} & y_{22} & \cdots & y_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nm} \end{pmatrix}, \tag{3.4}$$

where m denotes the number of objectives, and $\boldsymbol{Y}$ is an $n \times m$ matrix. The objective functions $\{f_1, f_2, ..., f_m\}$, which characterize how we get the objective values, map an $\boldsymbol{x}$ to $\boldsymbol{y} = \{y_1, y_2, ..., y_m\}$. We place an appropriate GP prior on each f, *i.e.*, $f_i(\boldsymbol{x}) \sim \mathcal{GP}(\mu_i, \sigma_i^2)$, where μ_i and σ_i^2 are the mean value function and the covariance function for the i-th objective, respectively. A non-trivial problem is that the objectives in our problem are not orthogonal since higher performance often comes with higher power dissipation. Characterizing individual objectives using independent GP models could degrade the overall modeling performance. Hence, we introduce multi-task GP models to handle the difficulty [194]. The main idea of multi-task GP is to model the hidden mappings based on the *objective identities* and the observed objective values for each microarchitecture. Objective identities are scalars utilized to distinguish different objectives. Hence, a GP prior is also placed between each objective function, as

shown in Equation (3.5).

$$\text{cov}(f_i(\boldsymbol{x}), f_j(\boldsymbol{x}')) = \boldsymbol{K}^f_{ij}\boldsymbol{K}^x(\boldsymbol{x}, \boldsymbol{x}'), \tag{3.5}$$

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

Given a newly-sampled microarchitecture embedding $\boldsymbol{x}^*$, the mean value representing the prediction of an objective value is shown in Equation (3.6)

$$\mu_i(\boldsymbol{x}^*) = \mu_i + (\boldsymbol{K}^f_i \otimes \boldsymbol{K}^x(\boldsymbol{x}^*, \boldsymbol{X}))^\top \Sigma^{-1}(\boldsymbol{Y}_i - \boldsymbol{\mu}_i), \tag{3.6}$$

where $\boldsymbol{K}^f_i$ is the i-th column of $\boldsymbol{K}^f$ defined in Equation (3.5), $\boldsymbol{\mu}_i$ denote a vector of i-th objective mean values, and $\boldsymbol{Y}_i$ is the i-th column of $\boldsymbol{Y}$ denoting the i-th objective values in the training data set. The uncertainty of the prediction $\mu_i(\boldsymbol{x}^*)$ in Equation (3.6) is calculated by Equation (3.7).

$$\begin{aligned}\sigma^2_i(\boldsymbol{x}^*) = (\boldsymbol{K}^f_i \otimes \boldsymbol{K}^x(\boldsymbol{x}^*, \boldsymbol{x}^*)) - \\ (\boldsymbol{K}^f_i \otimes \boldsymbol{K}^x(\boldsymbol{x}^*, \boldsymbol{X}))\Sigma^\top(\boldsymbol{K}^f_i \otimes \boldsymbol{K}^x(\boldsymbol{X}, \boldsymbol{x}^*)),\end{aligned} \tag{3.7}$$

where Σ in Equation (3.6) and Equation (3.7) is obtained from Equation (3.8).

$$\Sigma = (\boldsymbol{K}^f_i \otimes \boldsymbol{K}^x(\boldsymbol{X}, \boldsymbol{X})) + \boldsymbol{D} \otimes \boldsymbol{I}. \tag{3.8}$$

The operator $\otimes$ denotes Kronecker product. In Equation (3.8), $\boldsymbol{D}$ is an $m \times m$ diagonal matrix with σ^2_k as the k-th diagonal element, and $\boldsymbol{I}$ is the identity matrix. Therefore, the posterior distribution is $f_i(\boldsymbol{x}^* \mid \boldsymbol{X}, \boldsymbol{Y}_i, \boldsymbol{K}^f_i, \boldsymbol{K}^x) \sim \mathcal{GP}(\mu_i(\boldsymbol{x}^*), \sigma^2_i(\boldsymbol{x}^*))$.

The likelihood estimation of the multi-task GP is as shown in Equation (3.9)

$$
\begin{aligned}
l = & -\frac{mn}{2}\log 2\pi - \frac{n}{2}\log|\boldsymbol{K}^f| - \frac{m}{2}\log|\boldsymbol{K}^x| \\
& - \frac{1}{2}\mathrm{Tr}[(\boldsymbol{K}^f)^{-1}\Gamma^\top(\boldsymbol{K}^x)^{-1}\Gamma] \\
& - \frac{n}{2}\sum_{i=1}^{m}\log\sigma_i^2 - \frac{1}{2}\mathrm{Tr}[(\boldsymbol{Y}-\Gamma)\boldsymbol{D}^{-1}(\boldsymbol{Y}-\Gamma)^\top],
\end{aligned}
\tag{3.9}
$$

where Γ is a matrix of μ_{ij} corresponding to $\boldsymbol{Y}$. The parameters describing $\boldsymbol{K}^f$ and $\boldsymbol{K}^x$ are optimized via the maximization of Equation (3.9).

We use deep kernels [196] stacked by multiple linear perceptrons (MLP) with non-linear transformations to construct the Gaussian kernels, *i.e.*, $\boldsymbol{K}^f$ and $\boldsymbol{K}^x$. We term our surrogate model DKL-GP due to the construction of the GP and deep kernel functions. The use of deep kernels can result in improved performance due to their strong modeling ability. Thus far, $\boldsymbol{K}^f$ and $\boldsymbol{K}^x$ are described by Equation (3.10).

$$
\boldsymbol{K} \rightarrow (\varphi(g(\boldsymbol{x},\boldsymbol{\theta})), \varphi(g(\boldsymbol{x}',\boldsymbol{\theta}))),
\tag{3.10}
$$

where φ denotes non-linear transformation functions, *e.g.*, the ReLU activation function [16], *etc.*, and the MLP g is parameterized by $\boldsymbol{\theta}$.

3.3.4 Correlated Multi-Objective Optimization

Although DKL-GP, as demonstrated in Section 3.3.3, can be utilized to characterize the uncertainty of predicted clock cycles and power, designing a suitable acquisition function to find the Pareto frontier remains an unsolved problem. Since clock cycles and power are a pair of negatively-correlated metrics, we introduce the expected improvement of Pareto hypervolume (EIPV) [173] to model the trade-off in

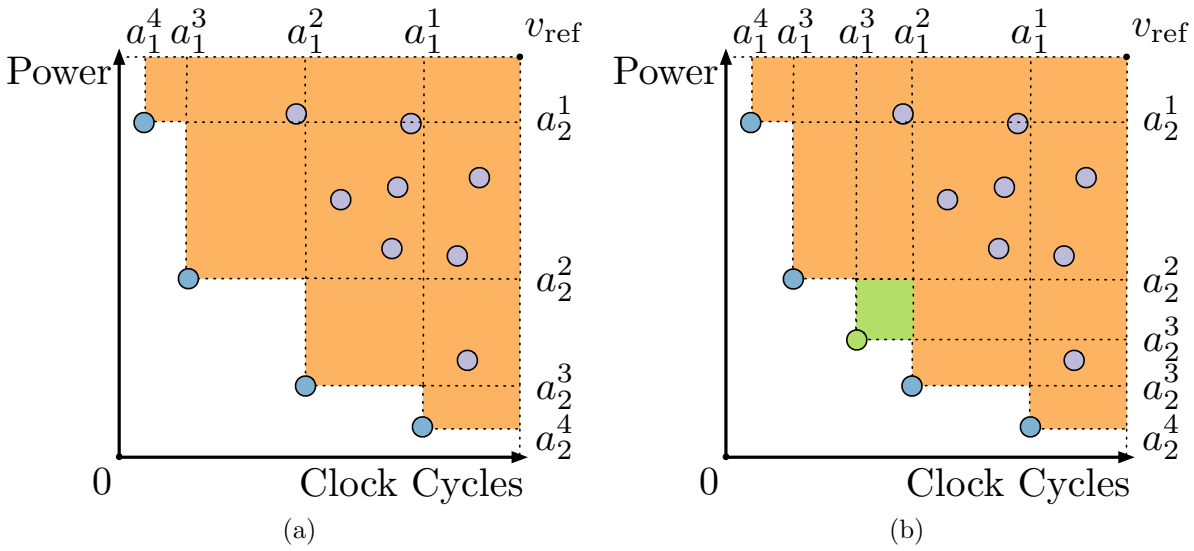


Figure 3.4: An example of hypervolume is shown in the power-performance space. (a) The region covered in orange is dominated by the currently explored Pareto-optimal objective values denoted as circles in blue. Circles in purple denote dominated objective values. (b) The circle in green denotes an explored potential point belonging to the Pareto-optimal set among the entire design space. EIPV is represented as the area of the sub-region colored in light green.

the performance-power space.

We denote the points in the performance-power space as $\boldsymbol{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ...\boldsymbol{y}_n\}$. As Figure 5.2 shows, a better performance and power trade-off lies closer to the origin. Pareto hypervolume is the volume of the area bounded by the Pareto frontier and a reference point. The reference point is a self-defined point dominated by all objective values. We use $\mathcal{P}(\boldsymbol{Y})$ to represent the Pareto frontier, *i.e.*, $\mathcal{P}(\boldsymbol{Y}) = \{\boldsymbol{y}_i \in \boldsymbol{Y} \mid \boldsymbol{y}_j \not\succeq \boldsymbol{y}_i, \forall \boldsymbol{y}_j \in \boldsymbol{Y} \setminus \{\boldsymbol{y}_i\}\}$. Given a reference point $\boldsymbol{v}_{\text{ref}}$ in the objective space, which is dominated by $\mathcal{P}(\boldsymbol{Y})$. The Pareto hypervolume [173] bounded by $\boldsymbol{v}_{\text{ref}}$ and $\mathcal{P}(\boldsymbol{Y})$, as the orange region highlighted in Figure 3.4a, can be computed by Equation (3.11).

$$\text{PVol}_{\boldsymbol{v}_{\text{ref}}}(\mathcal{P}(\boldsymbol{Y})) = \int_{\boldsymbol{Y}} \mathbb{1}[\boldsymbol{y} \succeq \boldsymbol{v}_{\text{ref}}][1 - \prod_{\boldsymbol{y}_* \in \mathcal{P}(\boldsymbol{Y})} \mathbb{1}[\boldsymbol{y}_* \not\succeq \boldsymbol{y}]]\text{d}\boldsymbol{y}, \tag{3.11}$$

where $\mathbb{1}(\cdot)$ is the indicator function, which outputs one if its argument is true and zero otherwise. The integral characterized by Equation (3.11) sums up all bounded regions. Intuitively, if a new point $\boldsymbol{y}_{n+1}$ is searched out, and $\boldsymbol{y}_{n+1}$ is not dominated by any points in $\boldsymbol{Y}$, then $\mathrm{PVol}_{\boldsymbol{v}_{\mathrm{ref}}}(\mathcal{P}(\boldsymbol{Y} \cup \{\boldsymbol{y}_{n+1}\}))$ is increased. The increased part is specified as the improvement of Pareto hypervolume. The larger the increased part, the better the improvement of Pareto hypervolume is. The EIPV is the expectation of the improvement *w.r.t.* potential solution candidates at the $(n+1)$-th optimization steps. Formally, the EIPV is computed as Equation (3.12).

$$\mathrm{EIPV}(\boldsymbol{x}_{n+1} \mid \mathcal{D}) = \mathbb{E}_{p(f(\boldsymbol{x}_{n+1})|\mathcal{D})}[\mathrm{PVol}_{\boldsymbol{v}_{\mathrm{ref}}}(\mathcal{P}(\boldsymbol{Y}) \cup f(\boldsymbol{x}_{n+1})) - \mathrm{PVol}_{\boldsymbol{v}_{\mathrm{ref}}}(\mathcal{P}(\boldsymbol{Y}))], \tag{3.12}$$

where f is the DKL-GP mentioned in Section 3.3.3, $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{n}$ is the data set, and $\boldsymbol{x}_{n+1}$ is a newly sampled microarchitecture embedding at the step $n+1$. Figure 3.4b visualizes the EIPV based on Figure 3.4a, where the green region highlights $\mathrm{EIPV}(\boldsymbol{x}_{n+1} \mid \mathcal{D})$.

In the performance-power space, we can simplify Equation (3.12) to make EIPV better computable by decomposing the space as grid cells. Assume $\boldsymbol{v}_{\mathrm{ref}} = \{(a_1^0, a_2^0)\}$. The union of grid cells can be phrased as $\mathcal{C} = [a_1^1, a_1^0) \times [a_2^1, a_2^0) \times [a_1^2, a_1^1) \times [a_2^2, a_2^1) \times ...[a_1^n), a_1^{n-1} \times [a_2^n, a_2^{n-1})$. Denote $C_{\mathrm{nd}} = \{C \in \mathcal{C} \mid \boldsymbol{y}' \not\succeq \boldsymbol{y}, \forall \boldsymbol{y} \in \mathcal{C}, \boldsymbol{y}' \in \mathcal{P}(\boldsymbol{Y})\}$ as non-dominated cells. Hence, the simplified version of EIPV computation is derived, as shown in Equation (3.13).

$$\mathrm{EIPV}(\boldsymbol{x}_{n+1} \mid \mathcal{D}) = \sum_{C \in C_{\mathrm{nd}}} \int_C \mathrm{PVol}_{\boldsymbol{v}_C}(\boldsymbol{y}) p(\boldsymbol{y} \mid \mathcal{D}) \mathrm{d}\boldsymbol{y}. \tag{3.13}$$

The relative relations between microarchitecture embeddings $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, *i.e.*, whether $\boldsymbol{x}_1$ is better than $\boldsymbol{x}_2$ can be precisely described by EIPV. The microarchitecture em-

bedding $\boldsymbol{x}^*$, which achieves the maximal EIPV, explicitly demonstrates that $f(\boldsymbol{x}^*)$ is the potential Pareto-optimal solution, as shown in Equation (3.14).

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}\in\mathcal{D}} \mathrm{EIPV}(\boldsymbol{x} \mid \mathcal{D}). \tag{3.14}$$

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

3.3.5 Diversity-Guided Parallel Exploration

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

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

optimization budget. In other words, exploring with EIPV misses such outliers when insufficient optimization is applied.

Two methodologies can be integrated to explore those outliers with higher Pareto hypervolume. First, improve the number of samples while maintaining an unchanged optimization budget with batch optimization. Applying parallel VLSI estimations in the batch optimization instead of original sequential optimization is necessary. And the parallelism depends on the number of available EDA tools licenses. Second, embedding domain knowledge or heuristics in the exploration is a good supplement to the exploration with original acquisition design (discussed in Section 3.3.4). Consequently, we propose a parallel-based exploration by combining two pathways.

In the exploration, we select multiple microarchitectures simultaneously to conduct parallel estimations using the VLSI flow. Intuitively, a straightforward way to select microarchitectures can be achieved according to the ranking of EIPV *w.r.t.* each design, as introduced in Section 3.3.4. Conversely, we provide some prior knowledge in the selection. DecodeWidth (mentioned in Section 3.3.2) contributes more to the performance and power trade-off than other components since modifying DecodeWidth can lead to distinct clusters shown in Figure 3.3. Except for DecodeWidth, another component IntPhyRegister, determining the number of integer physical registers, can also distinctly affect performance and power if a benchmark contains considerable integer-related instructions. Hence, the Pareto frontier is scattered in multiple sub-regions, as shown in Figure 3.5. In Figure 3.5, the DecodeWidth equals 1, and other components differ. We can observe that the Pareto frontier spread across multiple sub-regions, highlighted in red ribbons in Figure 3.5. An algorithm should inspect those sub-regions and sample microarchitecture embeddings across them, circumvent-

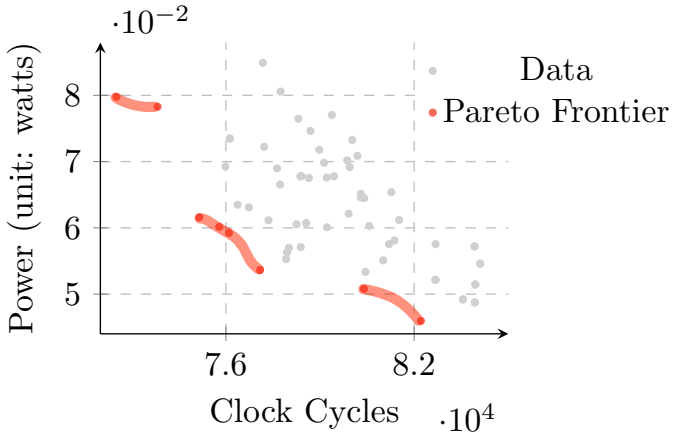


Figure 3.5: The DecodeWidth equals 1 for the sampled microarchitectures, and the Pareto frontier they formulated disperses across different sub-regions, colored in red.

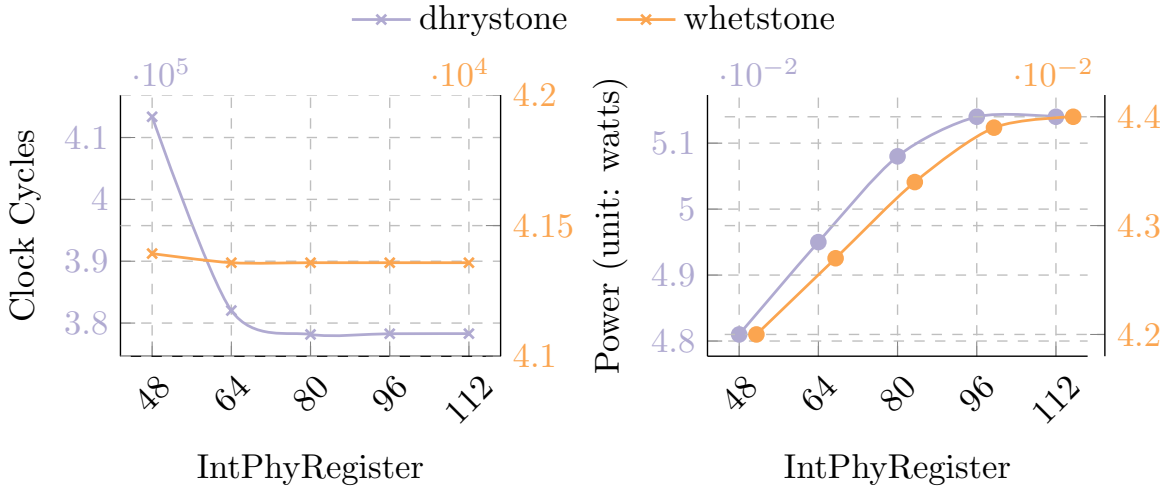


Figure 3.6: The change of performance and power dissipation *w.r.t.* IntPhyRegister on `dhrystone` and `whetstone`.

ing trapping into local optimum. Therefore, we propose the diversity-guided parallel exploration.

MicroAL leverages DecodeWidth as a critical factor to capture the main characteristics of the design space. Similarly, we also notice that IntPhyRegister can significantly affect the trade-off. Figure 3.6 visualizes such impacts. `Dhrystone` and

Algorithm 3 Partition ($\mathcal{U}$)

Require: $\mathcal{U}$ is the data set already clustered by MicroAL.
Ensure: The partitioned data set $\mathcal{U}'$.
1: $\mathcal{U}' \leftarrow \emptyset$;
2: $\mathcal{C} \leftarrow$ extract_clusters ($\mathcal{U}$); ▷ $\mathcal{C}$ is obtained from line 8 of Algorithm 2
3: **for** $\mathcal{C}_i \subset \mathcal{C}$ **do**
4: **for** $\boldsymbol{x} \in \mathcal{C}_i$ **do**
5: $\mathcal{U}'[\mathcal{C}_i][\boldsymbol{x}.\text{IntPhyRegister}] \leftarrow \boldsymbol{x}$;
6: **end for**
7: **end for**
8: **return** $\mathcal{U}'$;

`whetstone` are integer instructions-intensive and floating-point instructions-intensive benchmarks, respectively. When the microarchitecture selects IntPhyRegister from 48 to 112, the performance is improved by 8.52% and dissipates 6.86% more power. Since `whetstone` is mainly composed of floating-point instructions, increasing IntPhyRegister introduces no change to the performance but worsens power dissipation more. BOOM implements the unified integer physical register files design, *i.e.*, the registers hold both committed and speculative values/states.

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

We leverage IntPhyRegister to partition the clustered design space by MicroAL, as demonstrated in Algorithm 3. In Algorithm 3, the clusters are obtained from MicroAL (line 2), and partitions within each cluster are formulated according to

Algorithm 4 Diversity-Guided BOOM-Explorer $(\mathcal{D}, T, \mu, b, n)$

Require: $\mathcal{D}$ is the microarchitecture design space, T is the number of maximal iterations, μ is a normalization coefficient and b is the number of samples to draw, n is the number of pre-determined iterations for Algorithm 2.

Ensure: The microarchitectures $\boldsymbol{X}$ that form the Pareto optimality in $\mathcal{D}$.

1: $\boldsymbol{X}_0 \leftarrow$ **MicroAL** $(\mathcal{D}, \mu, b, n)$; ▷ Algorithm 2
2: Push $\boldsymbol{X}_0$ to the VLSI verification flow to obtain corresponding clock cycles and power values $\boldsymbol{Y}$;
3: $\mathcal{L} \leftarrow \boldsymbol{X}_0$; $\mathcal{U} \leftarrow \mathcal{D} \setminus \mathcal{L}$;
4: $\mathcal{P} \leftarrow$ **Partition** $(\mathcal{U})$; ▷ Algorithm 3
5: **for** $i = 1 \rightarrow T$ **do**
6: Establish and train DKL-GP with $(\mathcal{L}, \boldsymbol{Y})$;
7: Select b unvisited partitions randomly from $\mathcal{P}$;
8: **for** $j = 1 \rightarrow b$ **do**
9: $\boldsymbol{x}_{j*} \leftarrow \arg\max_{\boldsymbol{x} \in \mathcal{P}_j} \mathrm{EIPV}(\boldsymbol{x} \mid \mathcal{P}_j)$; ▷ Equation (3.13)
10: **end for**
11: $\boldsymbol{x}_* \leftarrow \{\boldsymbol{x}_{1*}, \boldsymbol{x}_{2*}, ..., \boldsymbol{x}_{b*}\}$;
12: Push $\boldsymbol{x}_*$ to the VLSI flow to obtain corresponding clock cycles and power values and add results to $\boldsymbol{Y}$;
13: $\mathcal{L} \leftarrow \mathcal{L} \cup \boldsymbol{x}_*$, $\mathcal{U} \leftarrow \mathcal{U} \setminus \boldsymbol{x}_*$;
14: **end for**
15: Construct $\boldsymbol{X}$ from $\mathcal{L}$ that form the Pareto optimality, according to Equation (3.3);
16: **return** Pareto-optimal set $\boldsymbol{X}$;

IntPhyRegister (line 5). Different sub-regions are constructed based on Algorithm 3. We sample microarchitecture embeddings according to the highest EIPV from each sub-region, respectively, as Equation (3.15) shows,

$$\boldsymbol{x}^* = \{\arg\max_{\boldsymbol{x}' \in \mathrm{P}_j} \mathrm{EIPV}(\boldsymbol{x}' \mid \mathrm{P}_j) \mid j = 1, 2, ..., N\}, \tag{3.15}$$

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

Algorithm 4 details the whole framework. Line 4 performs the partition. Combined with Table 3.1 and MicroAL, the partition operation incurs 25 different sub-

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

Other solutions could also be leveraged in Algorithm 3 to further improve the performance. First, some partitions can be directly pruned by the expertise. Architects have predetermined performance or power design goals. The selection of partitions can be made based on these goals. For example, we prefer to use a wider microarchitecture to pursue higher performance. So, we can only focus on partitions with decode width attaining the maximal value while neglecting other partitions, *i.e.*, we only focus on "important" partitions. Second, the exploration-exploitation heuristic can be utilized, similar to the action selection in reinforcement learning [184]. We decide whether to exploit the current best-known partition that effectively achieves higher Pareto hypervolume or explore new partitions. The decision could be based on epsilon-greedy heuristics, upper confidence bound, *etc.* We leave these discussed solutions in our future work.

3.4 Experiments

In this section, we conduct comprehensive experiments to evaluate BOOM-Explorer.

3.4.1 Experiments Settings

We conduct comprehensive experiments to evaluate the proposed BOOM-Explorer. Chipyard framework [19] is leveraged to compile various BOOM RTL designs. We utilize 7-nm ASAP7 PDK [190] for the VLSI flow. Cadence Genus 18.12-e012_1 is used to synthesize every sampled RTL design, and Synopsys VCS M-2017.03 is used to simulate the design running at 2GHz with different benchmarks. PrimeTime PX R-2020.09-SP1 is finally used to get power value for all benchmarks.

We utilize Chipyard [19] to generate various BOOM RTL designs. And we use 7-nm ASAP7 PDK [53, 190] for the VLSI flow. Cadence Genus 18.12-e012_1 is used to synthesize every sampled RTL design with 1 GHz timing constraints, and Synopsys VCS M-2017.03 is used to simulate the design with different benchmarks. PrimeTime PX R-2020.09-SP1 is leveraged to get power values for all benchmarks. All experiments are conducted in 80 cores of Intel(R) Xeon(R) CPU E7-4830 v2 @ 2.20GHz with 1 TB main memory.

In the settings of BOOM-Explorer, DKL-GP is stacked with three hidden layers, each of which has 1000, 500, and 50 hidden neurons, respectively, and it adopts ReLU as the non-linear transformation for deep kernels. The Adam optimizer [107] is used, with an initial learning rate equal to 0.001. BOOM-Explorer performs Bayesian exploration with 9 iterations. All experiments together with baselines are repeated 10 times, and we report corresponding average results.

3.4.2 Benchmarks, Baselines & Evaluation Metrics

To assess each microarchitecture, we employ a set of benchmarks selected from bare models [1]. These benchmarks include `median`, `mt-vvadd`, `whetstone`, `mm`, *etc.*

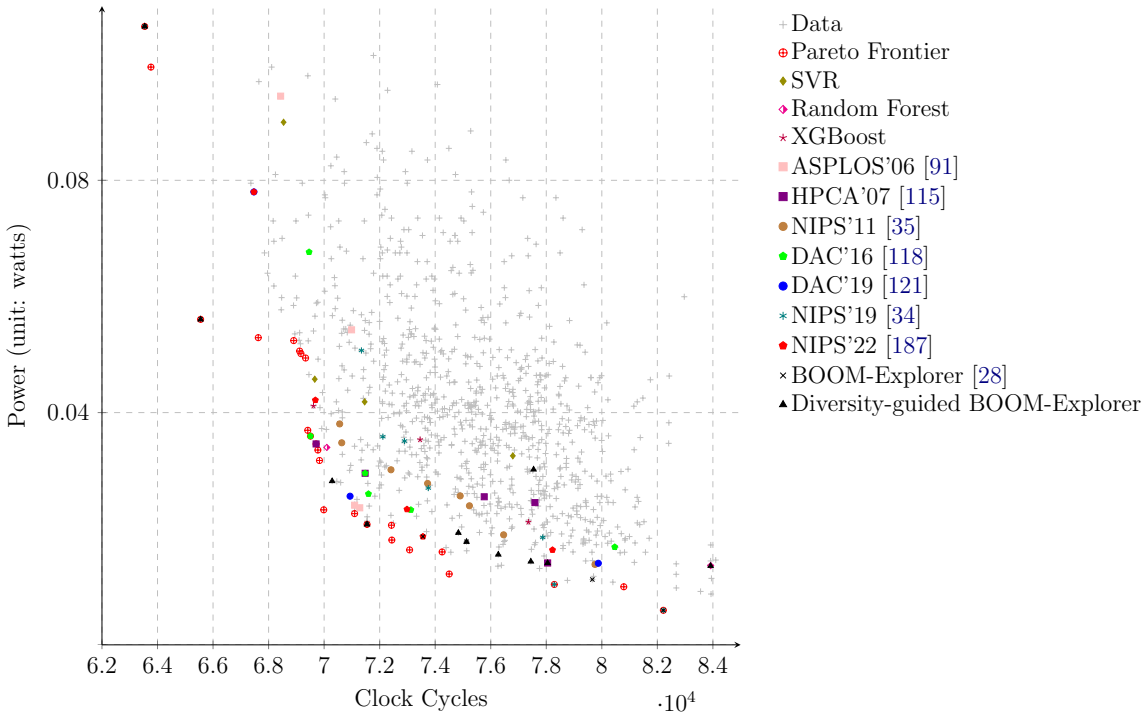


Figure 3.7: Comparisons between the predicted Pareto frontier and the real Pareto frontier of BOOM microarchitectures.

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

Several representative baselines are compared with the diversity-guided BOOM-Explorer. The ANN-based method [91] (shorted as ASPLOS'06) stacks ANN as the performance model for a multiprocessor to conduct DSE. The regression-based method [115] (termed HPCA'07) leverages regression models with non-linear transformations to explore the performance-power Pareto frontier for POWER microprocessors. The AdaBoost-RT-based method [118] (abbreviated as DAC'16) utilizes the orthogonal design sampling [65] and active learning-based AdaBoost regression tree models [179] to explore an Alpha21264-like microprocessor [36, 123]. The arts mentioned above proved effective in exploring microarchitecture parameters in their works, respectively. Therefore, it is requisite to compare these methodologies with our proposed methodology. The high-level synthesis (HLS) predictive model-based method [121] (named DAC'19) exploring the HLS design is also chosen as our baseline. Although the starting point is different, their method proved robust and transferable. We also compare diversity-guided BOOM-Explorer with traditional machine learning models, including support vector regression (SVR) and tree-based approaches such as random forest, XGBoost [47], and tree-structured Parzen estimator approach (abbreviated as NIPS'11) [35]. In addition, we compare previous state-of-the-art Bayesian optimization approaches [34, 187] with our proposed methodology since both methods adopt the same optimization framework but are distinct in the design of initialization, surrogate model, and acquisition function. We name the two state-of-the-art approaches as NIPS'19 and NIPS'22, respectively. For fair comparisons, the experimental settings of the baselines are the same as those mentioned in their papers. In traditional machine learning algorithms (SVR, random forest, and XGBoost), since they are not fit for Bayesian optimization due to unavailable predictive

uncertainties, we leverage simulated annealing [189] to explore the design space. We also compare the proposed methodology with the sequential optimization version of BOOM-Explorer [28]. All algorithms explore the same design space, as defined in Table 3.1. In our future work, we will delve into more intricate design spaces that encompass branch prediction algorithms, prefetching, and additional objectives, such as area metrics. However, one caveat is that there is a resemblance between microarchitecture and area values, akin to power values, where a larger area usually corresponds to higher power dissipation. Meeting higher performance requirements often entails employing more hardware resources, which, in turn, leads to a larger area.

To compare the diversity-guided BOOM-Explorer with all baselines, we utilize multiple metrics. These metrics include the Pareto hypervolume, the average distance to the reference set (ADRS), and the overall running time (ORT). Pareto hypervolume and ADRS are two widely used metrics in estimating the performance of DSE among multiple objectives. As mentioned in Section 3.3.4, the Pareto hypervolume measures the volume of the space enclosed by all solutions on the explored Pareto frontier and a user-defined reference point, where the computation of Pareto hypervolume is defined in Equation (3.11). The ADRS computes the average distance between the predicted Pareto frontier and the real Pareto frontier, providing insights into the quality and proximity of the solutions to the golden results. Equation (3.16) lists the computation of ADRS.

$$\mathrm{ADRS}(\Gamma, \Omega) = \frac{1}{|\Gamma|} \sum_{\gamma \in \Gamma} \min_{\omega \in \Omega} f(\gamma, \omega), \tag{3.16}$$

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

Table 3.3: Normalized Experimental Results for Pareto hypervolume, ADRS & ORT.

Methodologies	Norm. Pareto Hypervolume		Norm. ADRS		Norm. ORT
	Val.	Ratio	Val.	Ratio	
SVR	1.1519	1.0000×	0.2400	1.0000×	1.0000×
Random Forest	1.1794	1.0238×	0.2263	0.9430×	0.9763×
XGBoost	1.3152	1.1417×	0.2171	0.9046×	1.0102×
ASPLOS'06 [91]	1.3266	1.1516×	0.1948	0.8116×	0.9436×
HPCA'07 [115]	1.3218	1.1475×	0.1907	0.7949×	0.8544×
NIPS'11 [35]	1.3547	1.1760×	0.1723	0.7181×	0.7506×
DAC'16 [118]	1.3886	1.2055×	0.1473	0.6141×	3.0102×
DAC'19 [121]	1.3395	1.1628×	0.1884	0.7852×	0.8973×
NIPS'19 [34]	1.5496	1.3452×	0.1178	0.4908×	0.3567×
NIPS'22 [187]	1.5625	1.3564×	0.1426	0.5944×	0.4436×
BOOM-Explorer w/o MicroAL [28]	1.4665	1.2731×	0.1441	0.6006×	**0.3307**×
BOOM-Explorer [28]	1.6280	1.4132×	0.1145	0.4773×	0.3555×
Diversity-guided BOOM-Explorer	**1.6362**	**1.4203**×	**0.0915**	**0.3815**×	0.3533×

hypervolume, the lower the ADRS and ORT, the better the DSE algorithm is.

3.4.3 Evaluation Results

Figure 3.7 shows the predicted Pareto frontier obtained by the baselines and diversity-guided BOOM-Explorer. The results show that the Pareto frontier generated by BOOM-Explorer and its diversity-guided version is much closer to the actual Pareto frontier in general. Moreover, the diversity-guided BOOM-Explorer improves the results visually.

The normalized Pareto hypervolume, ADRS, and ORT results are listed in Table 3.3. Three summaries can be drawn from Table 3.3. First, BOOM-Explorer achieves an average of 18.75% higher Pareto hypervolume, 35.47% less ADRS, and 65.38% less ORT compared to all baselines. Specifically, BOOM-Explorer outperforms ASPLOS'06, HPCA'07, DAC'16, and DAC'19 by 70.13%, 66.55%, 28.65%, and 64.54% in ADRS, respectively. Meanwhile, it accelerates the exploration by more than

88.19% compared with DAC'16. The improvement achieved by our method over previous approaches stems from a customized algorithm design explicitly tailored to the problem at hand. Our method asks for fewer estimations using the VLSI flow while effectively modeling the design space with as few samples as possible. Second, MicroAL contributes 11.00% and 20.53% to exalt the Pareto hypervolume and ADRS. The results are obtained from the comparison between BOOM-Explorer and BOOM-Explorer w/o MicroAL. It also illustrates that without MicroAL, the performance of BOOM-Explorer would be close to DAC'16. If more time budget is allowed, we expect better results received from BOOM-Explorer. Third, incorporating diversity guidance as a key enhancement, we achieve a significant improvement in the ADRS, surpassing BOOM-Explorer by 20.09% with comparable ORT. The improvement on ADRS is larger than the Pareto hypervolume, demonstrating that "outliers" can be explored. As discussed in Section 3.3.5, these outliers pertain to similar or smaller EIPV but closer to the real Pareto frontier. It is worth noting that we partition the design space *w.r.t.* IntPhyRegister, as discussed in Section 3.3.5. Hence, the proposed method has a better effect when we target to optimize for integer-intensive applications. Additionally, previous state-of-the-art approaches (NIPS'19 and NIPS'22) receive a relatively good Pareto hypervolume and smaller ADRS more efficiently than other baselines. Like random forest and XGBoost, the NIPS'11 baseline [35] adopts tree-based structures as the surrogate model but uses Bayesian optimization. The NIPS'19 [34] and NIPS'22 baselines [187] leverage information-theoretic acquisition function designs. Although, these baselines perform well in general DSE problems. They achieve mediocre results compared to diversity-guided BOOM-Explorer largely due to the missing customization in the algorithm design such as tightly coupled with

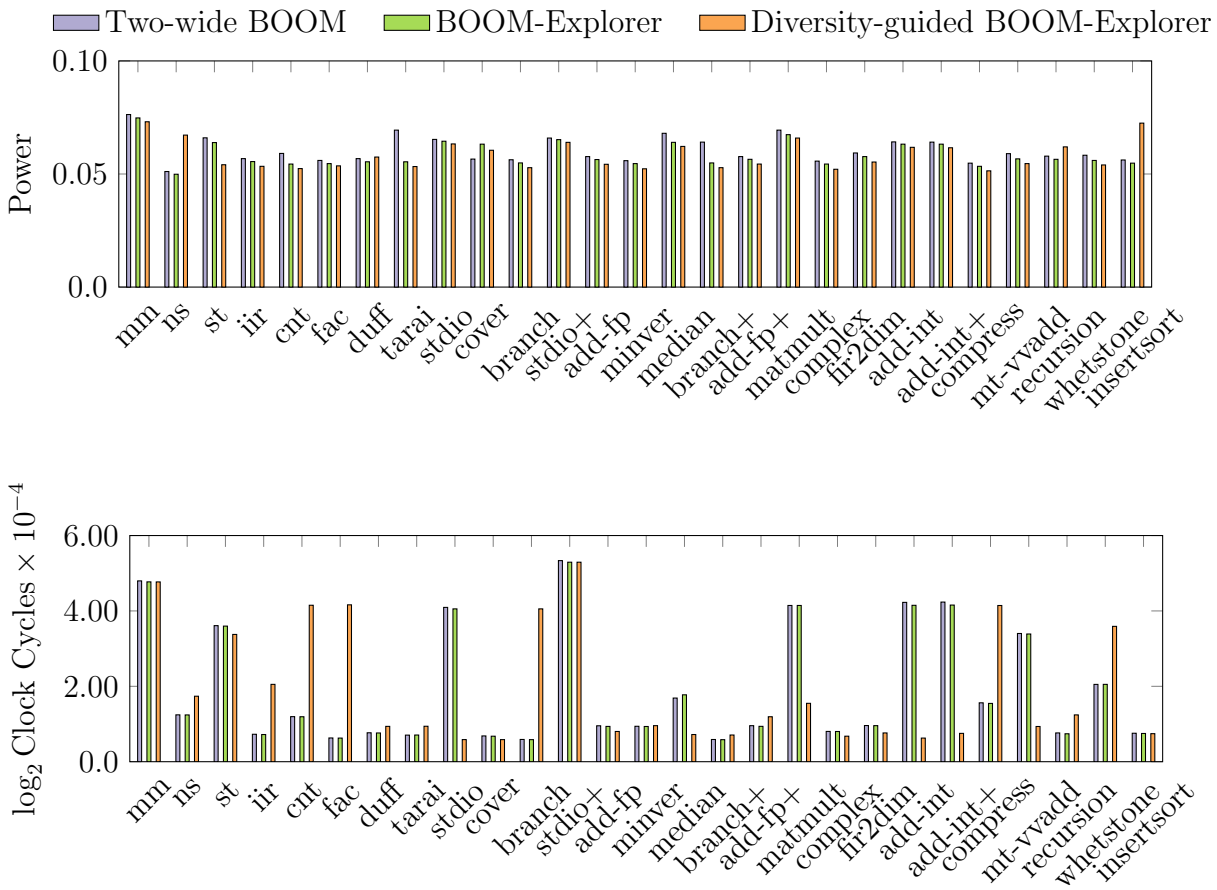


Figure 3.8: Performance and power comparisons of explored BOOM microarchitectures for different methodologies with more benchmarks.

expert knowledge.

3.4.4 Comparison of Pareto-Optimal BOOM Microarchitectures

We compare Pareto-optimal microarchitectures explored by proposed algorithms and human implementations [22, 44, 214] on more benchmarks to study how each BOOM

Table 3.4: Performance and power comparison with the two-wide BOOM.

Methodology	Microarchitecture Embedding [+]	Average Clock Cycles	Average Power (watts)
Two-wide BOOM [22, 44, 214]	$[4, 16, 32, 12, 4, 8, 2, 2, 64, 80, 64, 1, 2, 1, 16, 16, 4, 2, 8]$	74915.2963	6.0700×10^{-2}
BOOM-Explorer [28]	$[4, 16, 16, 8, 2, 8, 2, 2, 32, 64, 64, 1, 3, 1, 24, 24, 8, 4, 8]$	73333.7407	5.8600×10^{-2}
Diversity-guided BOOM-Explorer	$[4, 16, 16, 8, 4, 8, 2, 2, 32, 64, 64, 1, 2, 1, 24, 24, 8, 4, 8]$	73279.2820	5.8200×10^{-2}

[+] The parameters are in the same order as Table 3.1

microarchitecture balance the performance and power. The Pareto-optimal microarchitectures have similar parameter settings, as listed in Table 3.4.

Pareto-optimal designs found by BOOM-Explorer and diversity-guided BOOM-Explorer have the same decode width as the two-wide BOOM. However, the Pareto-optimal design reduces hardware components on the branch predictor (*i.e.*, RasEntry, BranchCount, *etc.*), entries of the reorder buffer, *etc.*, but enlarges instructions issue width, load queue (LDQ), store queue (STQ), *etc.* Moreover, it has different cache organizations, *e.g.*, different associate sets. Because LSU introduced in Section 2.2 tends to become a bottleneck of the microarchitecture, the Pareto-optimal design increases hardware resources for LDQ and STQ, increasing associate sets and miss status handling register (MSHR) entries for D-cache to overcome more data conflicts. Furthermore, the Pareto-optimal design found by diversity-guided BOOM-Explorer reduces the resources of return address stack (RAS) and branch target buffer (BTB) since they affect the trade-off less for the same branch prediction algorithm [172]. It also reduces the instruction issue slot but increases the I-cache associative sets. Both Pareto-optimal designs achieve a better trade-off on power and performance by reducing redundant hardware resources while increasing necessary components on critical paths [68].

We evaluate these microarchitectures with more benchmarks. Table 3.4 shows the average clock cycles and power values for all these benchmarks. These benchmarks are chosen from different application scenarios, *e.g.*, `add-int`, `add-fp`, *etc.* are from ISA basic instructions, `iir`, `firdim`, *etc.* are from DSP-oriented algorithms [216], `compress`, `duff`, *etc.* are from real-time computing applications [112], *etc.* Figure 3.8 shows the comparison of performance and power values, respectively. For all of these

benchmarks, BOOM-Explorer's design runs approximately 2.11% faster and, at the same time, dissipates 3.45% less power than the two-wide BOOM. The solution from diversity-guided BOOM-Explorer achieves 2.18% faster and 3.99% better on power dissipation than human implementations.

3.4.5 Effectiveness of MicroAL

To assess the effectiveness of MicroAL, we conduct an ablation study on the effectiveness of MicroAL. We investigate the integration of MicroAL to some baselines, which allows us to modify the initialization algorithm that is not tightly coupled with the exploration procedure. Specifically, we conduct MicroAL with SVR, random forest, XGBoost, ASPLOS'06 [91], HPCA'07 [115], NIPS'11 [35], NIPS19 [34], and NIPS'22 [187].

Figure 3.9 lists the comparative results on the Pareto hypervolume and ADRS when we integrate MicroAL to baselines. Two summaries are drawn from Figure 3.9. First, integrating MicroAL into baselines can improve an average of 8.06% Pareto hypervolume, and 12.15% ADRS. For example, when we integrate MicroAL into SVR, the Pareto hypervolume can be increased by approximately 18.08% while the ADRS is reduced by a large margin. Second, MicroAL can have negative effects on some benchmarks like NIPS'19 [34]. Although integrating MicroAL to NIPS'19 [34] results in a similar Pareto hypervolume, it degrades the ADRS by 25.16%. We argue that the reason behind the phenomenon is that we leverage single-task GP models to individually model the performance and power. In contrast, the samples generated by MicroAL are highly different. Therefore, the built surrogate models are rather "weak". On the other hand, objective-related features can be utilized to

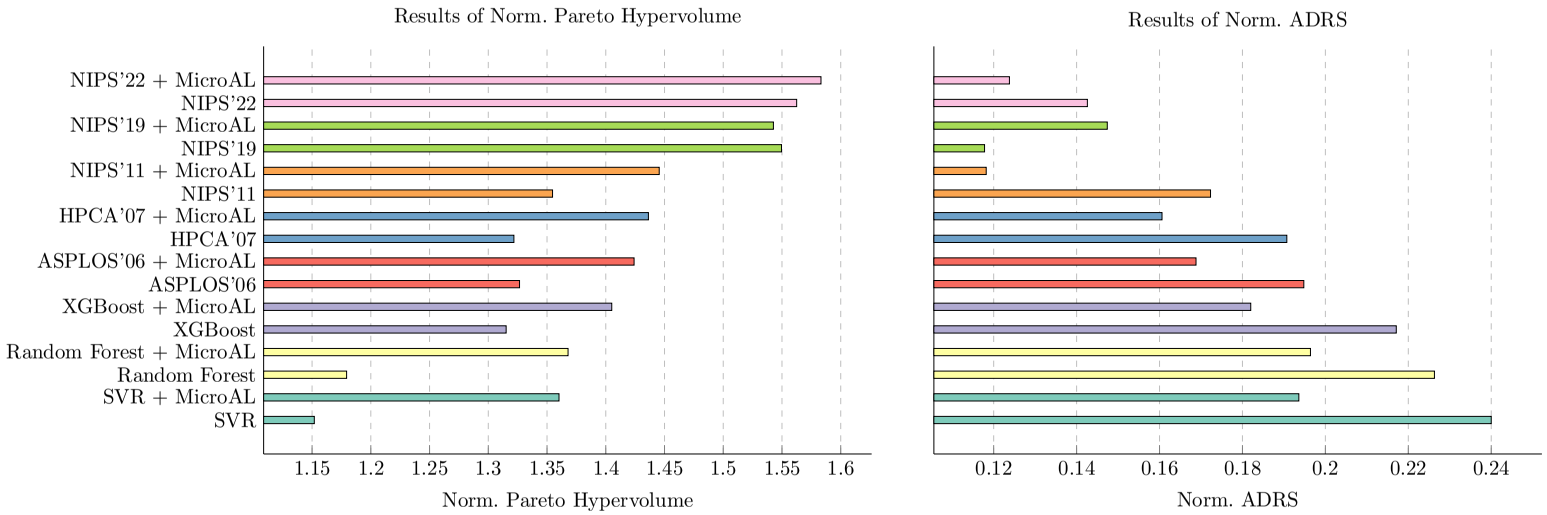


Figure 3.9: We integrate MicroAL into chosen baselines to investigate the MicroAL's effectiveness on the normalized Pareto hypervolume and ADRS.

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

3.5 Summary

In this chapter, BOOM-Explorer is proposed to search for Pareto optimality among the microarchitecture design space within a short time. We develop MicroAL and DKL-GP embedded in Bayesian optimization, through which we can learn good designs with a better trade-off between power and performance via maximization of EIPV. To the best of our knowledge, this is the first work introducing automatic design space exploration solution to the RISC-V community. We expect to see a lot of researches in our community to further improve microarchitecture design space explorations of processors.

Chapter 4

Reinforcement Learning Pathway

4.1 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. The microarchitecture 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, trial-and-error, and non-trivial procedure due to two factors. First, the design space is enormous and complicated. It comes from the high complexity of a microarchitec-

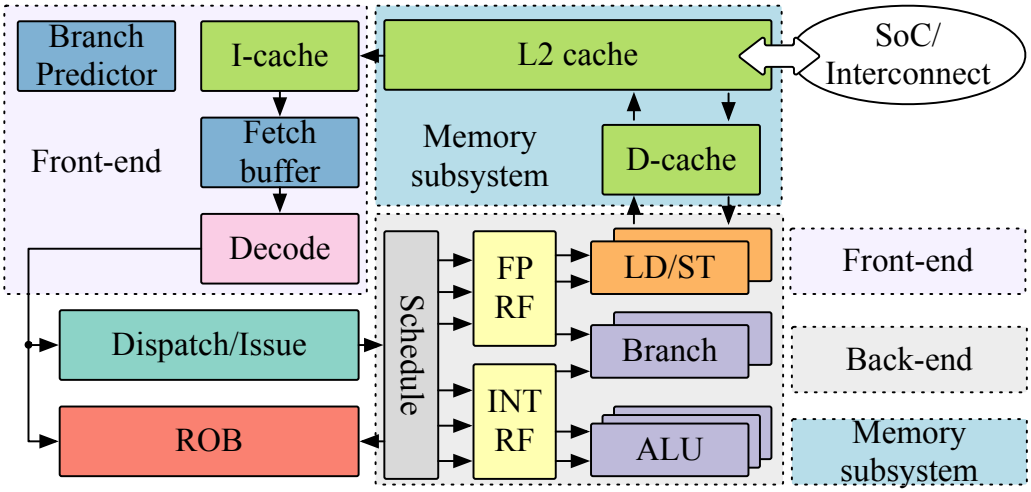


Figure 4.1: An overview of an example microarchitecture. Instructions fetched from I-cache are sent to functional units (*e.g.*, ALU, LD/ST, etc.) for execution. Register files (RF) save temporary data. Reorder buffer (ROB) achieves precise interrupts [181]. Components are highlighted with diverse colors, and the same color denotes a similar function.

ture, such as shown in Figure 4.1, which includes different *components* responsible for implementing specific functions [46, 78]. Second, 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 [120, 210]. 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 characteristics of most designs [27].

In this chapter, 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 to promote chip agile design methodology [117] due to the forecast that RISC-V could motivate competitive commercial products over x86, ARM, etc., for many applications in the future. 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 compared to using EDA tools only [211].

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× higher efficiency. And the solution qualities outperform human implementations by at most 2.03× in the PPA trade-off.

The remainder of this chapter is organized as follows. Section 4.2 presents preliminaries for the reinforcement learning pathway. Section 4.3 provides detailed descriptions of our reinforcement learning framework. Section 4.4 illustrates the rationales of our methodology. Section 4.5 conducts several experiments on BOOM microprocessor to confirm the outstanding performance of the proposed framework. Finally, Section 4.6 summarizes this chapter.

4.2 Preliminaries

In this section, we introduce preliminaries for the reinforcement learning pathway, including microarchitecture PPA modeling, microarchitecture scaling graph, and corresponding problem formulation.

4.2.1 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 includes steps like logic synthesis, placement and routing, netlist simulation, 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. In this section, 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 [36, 123], a performance simulator, and McPAT [119] as our fundamental PPA modeling tools.

4.2.2 Microarchitecture Scaling Graph

Since the microarchitecture scaling graph first appeared [63], computer architects relied on it to study mechanistic microexecutions [42, 43].

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

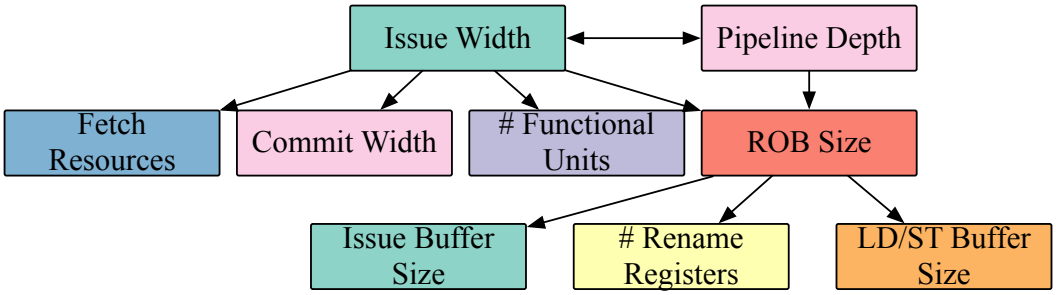


Figure 4.2: An overview of the microarchitecture scaling graph. Colors are matched with Figure 4.1.

The microarchitecture scaling graph is directed, elucidating the scaling precedence constraints between components, as shown in Figure 4.2. Nodes are components, and directed edges are scaling precedences. According to Figure 4.2, 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.

4.2.3 Problem Formulation for Reinforcement Learning Pathway

Given the microarchitecture design space, the problem is to find the solution to Equation (4.1) within a limited time budget.

$$\underset{\boldsymbol{s}\in\mathbb{D}^n}{max}[\text{Perf}(\boldsymbol{s}), -\text{Power}(\boldsymbol{s}), -\text{Area}(\boldsymbol{s})], \tag{4.1}$$

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

4.3 Methodology

In this section, we introduce our reinforcement learning framework.

4.3.1 Overview

We propose an RL solution framework with customized MDP (S, A, P, R) formulation, as shown in Figure 4.3. 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 5.1. The components' types 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.

4.3.2 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 relations 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 [168].

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. 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

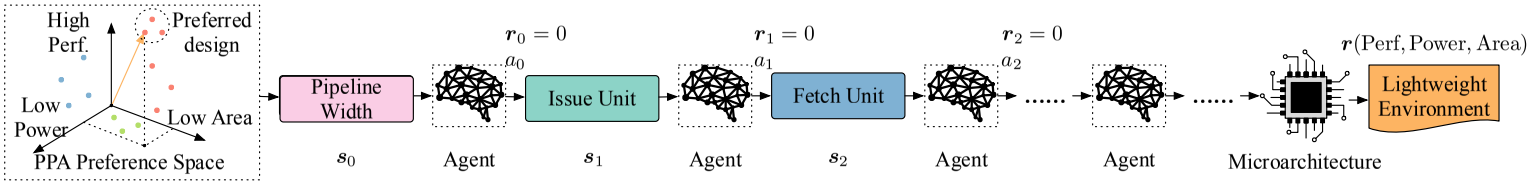


Figure 4.3: Overview of our RL framework. $\boldsymbol{s}$ denotes a state, a is an action, and $\boldsymbol{r}$ represents an immediate vectorized reward.

for the back-end (see Figure 4.1). Hence, our episode design explicitly provides such domain knowledge for the agent. As shown in Figure 4.3, 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 4.2), we determine their structures in a fixed order within an episode.

4.3.3 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 (4.2).

$$r = \boldsymbol{r}(\text{Perf}, \text{Power}, \text{Area}) \cdot (\alpha, \beta, \gamma)^\top \tag{4.2}$$

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.

4.3.4 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 (4.2) [158, 157]. Hence, a single agent can maximize r without retraining or fine-tuning in the DSE when $\boldsymbol{\phi}$ is changed.

CCS is the convex subset of the Pareto frontier, as formulated in Equation (4.3).

$$\begin{aligned} \text{CCS} = \{\boldsymbol{r} \in \mathcal{PF}(R) \mid \\ \exists \boldsymbol{\phi} \in \Phi, \boldsymbol{r}\boldsymbol{\phi}^\top \geq \boldsymbol{r}'\boldsymbol{\phi}^\top, \forall \boldsymbol{r}' \in \mathcal{PF}(R)\}, \end{aligned} \tag{4.3}$$

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]. Pareto frontier is a set of microarchitectures whose PPA values represent the best trade-off. Equation (4.3) indicates that optimal solutions can attain the maximal r (see Equation (4.2)) for a specific $\boldsymbol{\phi}$.

To facilitate the agent's learning of the CCS during training, a generalized Bellman optimality equality is applied [204]. 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 (4.4).

$$\begin{aligned} \boldsymbol{Q}(\boldsymbol{s}, a, \boldsymbol{\phi}) &= \boldsymbol{r}(\boldsymbol{s}, a) + \zeta \mathbb{E}_{\boldsymbol{s}' \sim \mathcal{P}(\cdot|\boldsymbol{s},a)} \mathcal{T}(\boldsymbol{Q}(\boldsymbol{s}', a, \boldsymbol{\phi})), \\ \mathcal{T}(\boldsymbol{Q}(\boldsymbol{s}', a, \boldsymbol{\phi})) &= \underset{\boldsymbol{Q}}{arg} \ \underset{a' \in A, \boldsymbol{\phi}' \in \Phi}{max} \boldsymbol{Q}(\boldsymbol{s}', a', \boldsymbol{\phi}')\boldsymbol{\phi}^\top, \end{aligned} \tag{4.4}$$

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

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

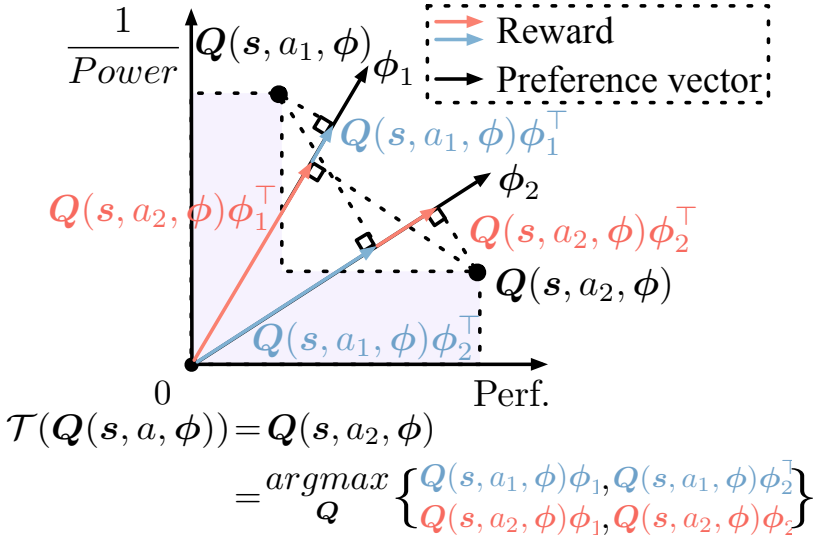


Figure 4.4: Optimization procedure with Equation (4.4).

maximal r via traversing the action space A, and sampled $\boldsymbol{\phi}'$ [3].

Figure 4.4 details an example optimization with Equation (4.4) in the performance-power space, given $\boldsymbol{\phi}$. Before applying Equation (4.4), 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) [131] in favor of high training efficiency over PPO [170] or SAC [81]. 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

[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.

the actor $\boldsymbol{\theta}_a$ is listed in Equation (4.5).

$$\begin{aligned}\bigtriangledown\boldsymbol{\theta}_a =&\kappa \bigtriangledown_{\boldsymbol{\theta}_a} H(\pi(\boldsymbol{s}_t;\boldsymbol{\theta}_a))+\\ &\mathbb{E}_{\xi\sim\pi}[\sum_{t=0}^{\infty}\bigtriangledown_{\boldsymbol{\theta}_a}\log\pi_{\boldsymbol{\theta}_a}(a_t \mid \boldsymbol{s}_t)A(\boldsymbol{s}_t,a_t,\boldsymbol{\phi}')\boldsymbol{\phi}^\top],\end{aligned} \tag{4.5}$$

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 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 (4.6) gives the loss function with an L2 normalization applied between two state-action vectors.

$$\begin{aligned}L_c =&\rho\|(\boldsymbol{Q}^* - \boldsymbol{Q}(\boldsymbol{s},a,\boldsymbol{\phi}';\boldsymbol{\theta}_c))\boldsymbol{\phi}^\top\|_2^2+\\ &(1-\rho)\|\boldsymbol{Q}^* - \boldsymbol{Q}(\boldsymbol{s},a,\boldsymbol{\phi}';\boldsymbol{\theta}_c)\|_2^2,\end{aligned} \tag{4.6}$$

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 (4.4). The first term in Equation (4.6) enforces optimizing the critic network w.r.t. the maximal reward shown in Equation (4.2). The *n-step* TD errors [150] is leveraged. However, Equation (4.5) 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 [169], as listed in Equation (4.7).

$$\boldsymbol{r}_t = \sum_{n=0}^{N}(\lambda\zeta)^{N-n}(\boldsymbol{r}_{t+k} + \zeta V_{t+1+k}(\boldsymbol{s}_t,\boldsymbol{\phi}') - V_{t+k}(\boldsymbol{s}_t,\boldsymbol{\phi}')), \tag{4.7}$$

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

4.3.5 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 [13].

4.3.6 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 [211]. We leverage the EDA flow in the calibration as a 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 [47] separately, with the loss function defined in Equation (4.8).

$$L = |f(\boldsymbol{s}, \boldsymbol{e}, \boldsymbol{p}) - y_{\text{gt}}|_2^2, \tag{4.8}$$

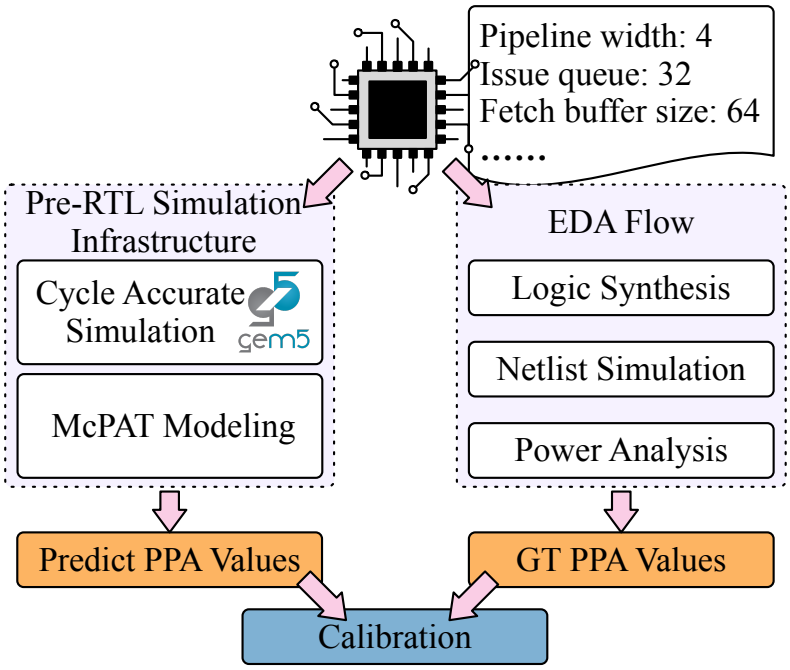


Figure 4.5: Overview of the PPA calibration.

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 sub-threshold power, etc.), respectively. y_{gt} is the ground truth. Figure 4.5 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.

4.3.7 Training Details

Algorithm 5 shows the pseudo-code of the actor-learner thread of our RL algorithm based on A3C [131]. Firstly, the replay buffer $\mathcal{B}$, weight parameters for the actor and critic, and a PPA preference vector are initialized (lines 1 to 2, and line 5). Secondly,

Algorithm 5 RL Training of an Actor-learner Thread

Require: $\mathbb{D}^n$: microarchitecture design space, Φ: the PPA preference space, N: the number of steps for TD error estimation, n: the size of a minibatch, E: maximal training budget.

Initialize the replay buffer $\mathcal{B}$.
Parameterize the actor and critic with $\boldsymbol{\theta}_a$ and $\boldsymbol{\theta}_c$;
for $e = 1 \rightarrow E$ **do**
    Synchronize thread-specific parameters $\boldsymbol{\theta}_a^-$ and $\boldsymbol{\theta}_c^-$ from $\boldsymbol{\theta}_a$ and $\boldsymbol{\theta}_c$;
    Sample a PPA preference vector $\boldsymbol{\phi} \in \Phi$;
    for $t = 0 \rightarrow N - 1$ **do**
        observe state $\boldsymbol{s}_t \in \mathbb{D}$;
        Sample an action $a_t \sim \pi(a_t \mid \boldsymbol{s}_t, \boldsymbol{\phi}; \boldsymbol{\theta}_c^-)$;
        Receive an immediate vector reward $\boldsymbol{r}_t$ and the following state $\boldsymbol{s}_{t+1}$ from the lighweight environment;
        Store transition $(\boldsymbol{s}_t, a_t, \boldsymbol{r}_t, \boldsymbol{s}_{t+1})$ in $\mathcal{B}$;
        if *update* **then**
            Sample a minibatch of transitions $(\boldsymbol{s}_j, a_j, \boldsymbol{r}_j, \boldsymbol{s}_{j+1})$ from $\mathcal{B}$;
            Sample n PPA preferences $W = \{\boldsymbol{\phi}_1, ..., \boldsymbol{\phi}_n\} \subset \Phi$;
            Update values
            $\mathcal{T}(\boldsymbol{V})_{ij} = \begin{cases} \boldsymbol{r}_j, & \boldsymbol{s}_{j+1} \text{ is the terminal state;} \\ \boldsymbol{r}^*, & \boldsymbol{s}_{j+1} \text{ is not the terminal state.} \end{cases}$, where $\boldsymbol{r}^* = \boldsymbol{r}_j + \zeta \underset{\boldsymbol{V}}{arg} \underset{a' \in A, \boldsymbol{\phi}' \in \Phi}{max} \boldsymbol{V}(\boldsymbol{s}_{j+1}, \boldsymbol{\phi}'; \boldsymbol{\theta}_c) \boldsymbol{\phi}_i^\top$, and $i = \{1, 2, ..., n\}$.
            Compute $\bigtriangledown \boldsymbol{\theta}_c$ with $\boldsymbol{\theta}_c^-$; ▷ Equation (6)
            Compute $\bigtriangledown \boldsymbol{\theta}_a$ with $\boldsymbol{\theta}_a^-$; ▷ Equation (5)
            Perform asynchronous update of $\boldsymbol{\theta}_c$ using $\bigtriangledown \boldsymbol{\theta}_c$, and update of $\boldsymbol{\theta}_a$ using $\bigtriangledown \boldsymbol{\theta}_a$.
        end if
    end for
end for

the trajectories of an episode are stored following the PPA preference vector $\boldsymbol{\phi}$ (lines 7 to 10). Thirdly, we perform a generalized Bellman optimality equality to update state vector values according to Equation (4) (lines 12 to 14). Finally, gradients w.r.t. the actor and critic are computed, and asynchronous updates are leveraged (lines 15 to 17).

4.4 Why RL?

Previous data-driven methods apply statistical analysis [118], Gaussian process [28], 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 [28, 195]. On the contrary, we find the relation does not hold generally, and demonstrate it with an anti-example shown in Figure 6.2. M1 is the baseline microarchitecture. M2 changes the branch predictor [172], M3 reduces the decode width, and M4 decreases branch speculation tags. t-SNE [188] 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.

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

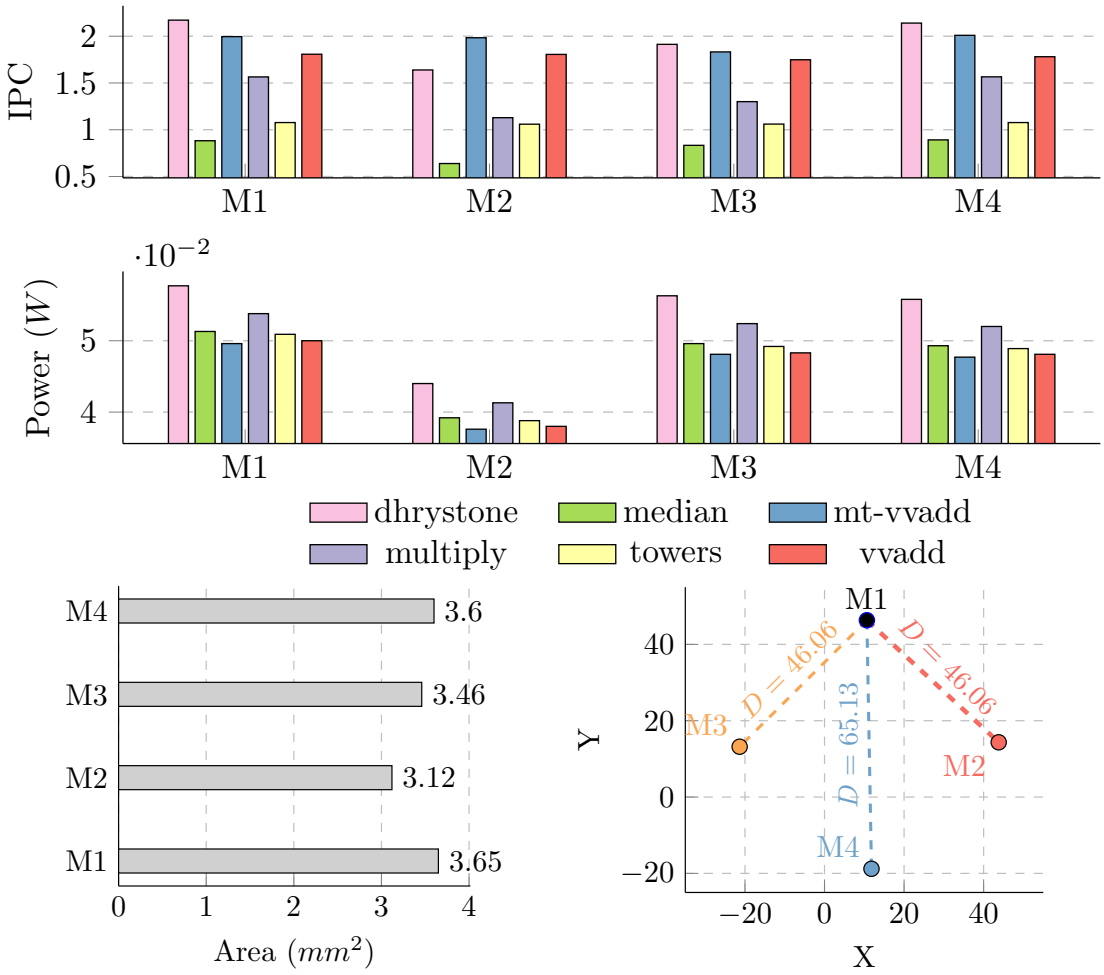


Figure 4.6: Four SonicBOOM microarchitectures' PPA values of six benchmarks reported from EDA tools, and the visualization of embeddings distances.

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.

4.5 Experiments

In this section, we conduct comprehensive experiments to evaluate our reinforcement learning framework.

4.5.1 Our Microarchitecture Design Space Specification

We evaluate the proposed RL framework with representative in-order and out-of-order RISC-V microprocessors, Rocket [23] and different scales of SonicBOOM [214] (categorized by "pipelineWidth"), as shown in Table 5.1. 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.

4.5.2 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, Synopsys PrimeTime PX R-2020.09-SP1, and Cadence Genus 18.12-e012_1 with 7-nm technology [53]. Code is publicly available at `https://github.com/baichen318/rl-explorer`.

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	fetchBufferEntries	$6:46:2$
		fetchWidth	$4,8$
		ftq.nEntries	$12:64:4$
	pipelineWidth		$1:5:1$
	ROB		$24:160:4$
	PRF	intPhysRegisters	$40:176:8$
		numFpPRF	$34:132:6$
	ISU	fpPhysRegisters	$1:5:1$
		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.

Table 4.1: RISC-V Microarchitecture Design Space

The coefficient κ in Equation (4.5) is set as 1, ρ in Equation (4.6) is 0.5, λ in Equation (4.7) is 0.95 and the discount factor ζ in Equation (4.7) 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 [28] (ICCAD'21), Adaboost-based [118] (DAC'16), ranking-based [48] (ISCA'14), and human efforts [23, 214]. The baselines are implemented according to the original papers. We use `towers`, `vvadd`, `spmv` from official RISC-V tests as workloads in the DSE. Results on more workloads 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.

4.5.3 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 4.7 and Figure 4.8 list the accuracy of lightweight PPA XGBoost models and the ratio of microarchitectures in the training data set leveraged in the calibration flow for Rocket and Large SonicBOOM. In the first row, the blue line "GT = Pred" visualizes the error when PPA models are trained using the entire training data set. For Rocket, the lightweight PPA models achieve MAPE values of 1.3539%, 1.6482%, and 2.7452%, respectively, along with Kendall τ values of 0.95, 0.82, and 0.91. For Large SonicBOOM, 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.

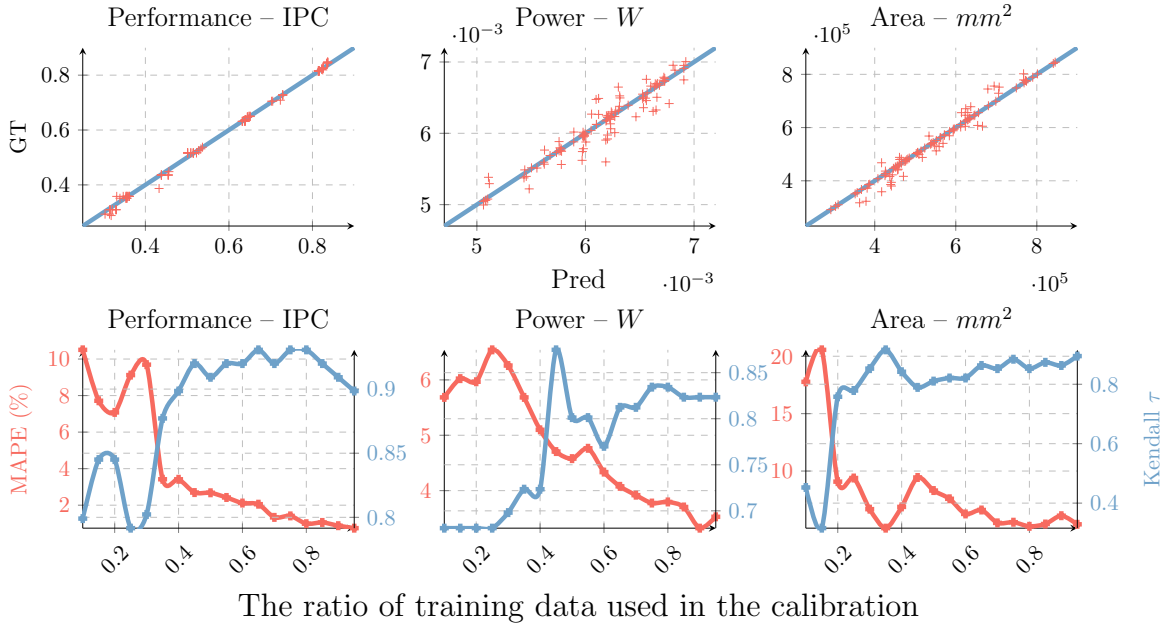


Figure 4.7: The accuracy of lightweight PPA models, and MAPE and Kendall τ curves w.r.t. the calibration data size.

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 4.7 and Figure 4.8. For Rocket, it is worth noting that by leveraging over 80% training data set, *i.e.*, approximately 640 different designs, the two curves (MAPE and Kendall τ) tend to be stable. By leveraging around 800 $\sim$ 900 SonicBOOM microarchitecture designs, the Kendall τ for PPA modeling results can achieve higher than 0.92.

4.5.4 RL Training

Figure 4.10 displays the RL training metric curves for Large SonicBOOM, which include PPA values and specific values of PPA preference vectors. Different PPA

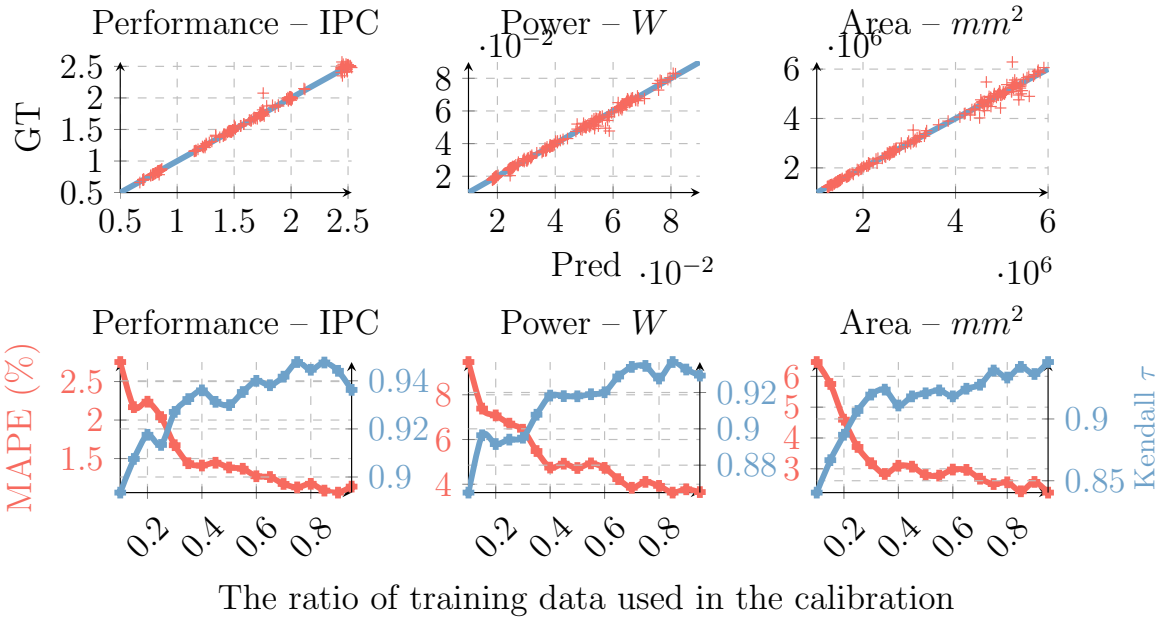


Figure 4.8: The accuracy of lightweight PPA models, and MAPE and Kendall τ curves w.r.t. the calibration data size.

preference vectors are sampled throughout the training, resulting in perturbed 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.

?? lists the RL training metric curves for Rocket and the different scales of other SonicBOOM. Similar to Figure 4.10, PPA values change in response to different PPA preference vectors.

When the PPA preference vector is fixed during training, we expect to observe increasing IPC values alongside decreasing power and area values. However, due to the sampling of divergent PPA preference vectors during training, the PPA values that achieve the optimal trade-off also change, resulting in curve fluctuations. And

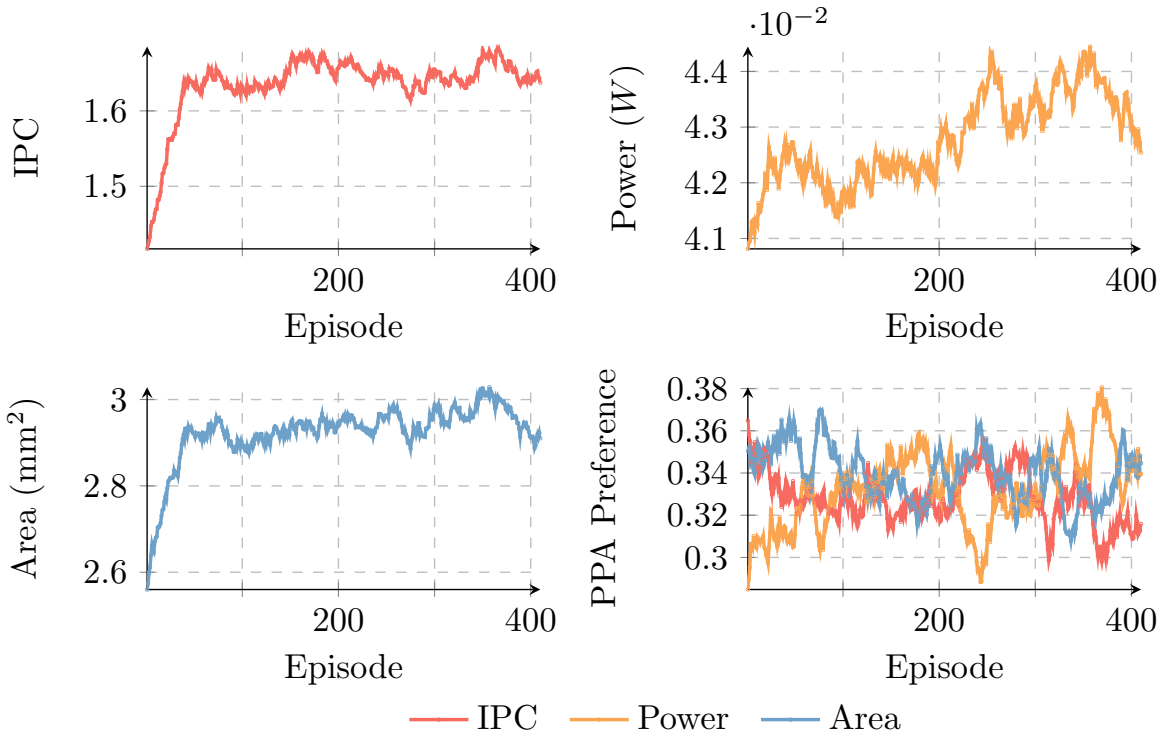


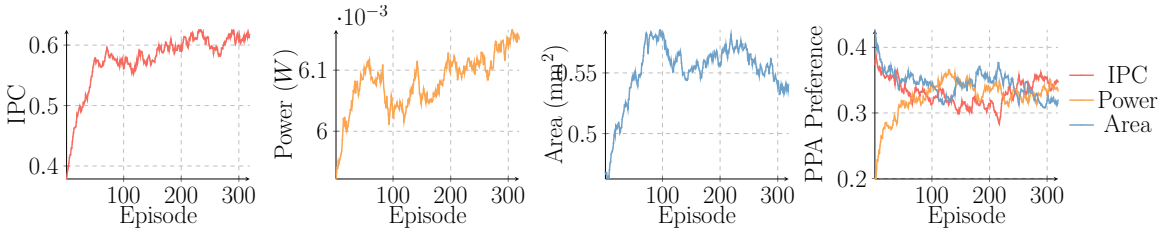
Figure 4.9: RL training status of Large SonicBOOM.

for different designs, the changes in these curves are different.

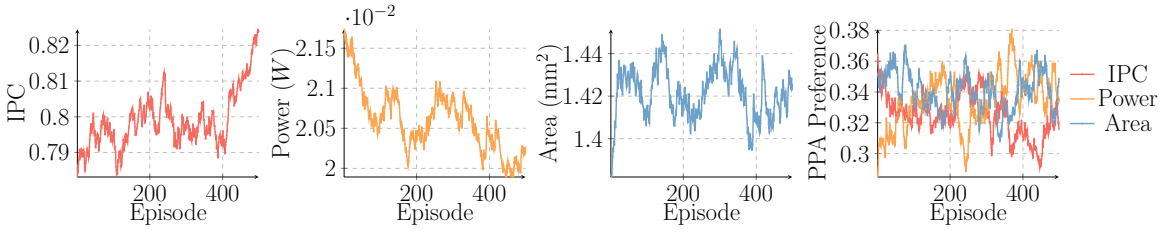
4.5.5 Comparison w. Human Efforts & Prior Arts

Three metrics: Perf/Power, Perf/Area, and (Perf × Perf)/(Power × 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.

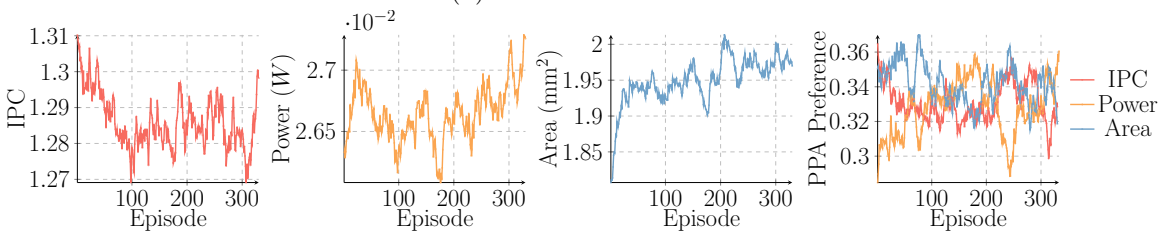
Table 4.2 lists the results. The relative runtime for exploration is also reported. Explored Rocket and nearly all scales of SonicBOOM by our method are better than prior works and human efforts. In summary, our solutions achieve an average of



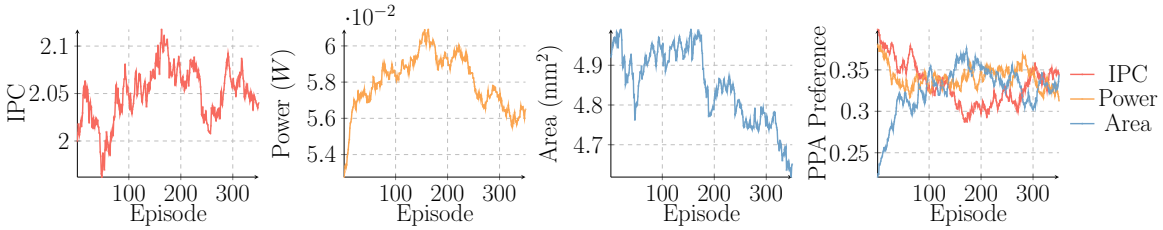
(a) Rocket



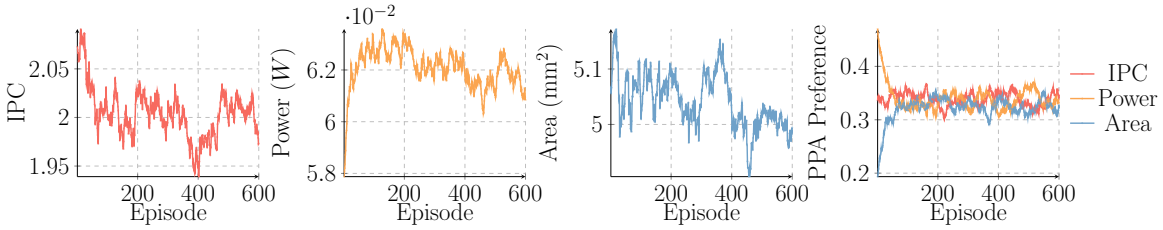
(b) Small SonicBOOM



(c) Medium SonicBOOM



(d) Mega SonicBOOM



(e) Giga SonicBOOM

Figure 4.10: RL training status of Rocket and different scales of SonicBOOM.

Design	Method	Performance IPC	Power mW	Area mm^2	Perf / Power		Perf / Area		(Perf × Perf) / (Power × Area)		Runtime
					Val.	Ratio	Val.	Ratio	Val.	Ratio	
Rocket	Human Efforts	0.734	2.700	0.908	0.272	– [1]	0.808	–	0.220	–	–
	ISCA'14	0.816	2.300	0.794	**0.355**	**1.305**×	1.027	1.271×	0.364	1.659×	8.611×
	DAC'16	0.549	1.800	0.534	0.305	1.121×	1.028	1.272×	0.313	1.426×	5.896×
	ICCAD'21	0.728	2.100	0.745	0.347	1.275×	0.977	1.209×	0.339	1.542×	1.501×
	Ours	0.728	2.300	0.576	0.316	1.164×	**1.263**	**1.563**×	**0.400**	**1.820**×	**1.000**
Small SonicBOOM	Human Efforts	0.784	20.300	1.505	0.039	–	0.521	–	0.020	–	–
	ISCA'14	0.820	15.000	1.284	0.055	1.415×	0.638	1.226×	0.035	1.735×	5.803×
	DAC'16	0.808	14.700	1.251	0.055	1.423×	0.645	1.239×	0.035	1.764×	4.792×
	ICCAD'21	0.847	20.000	1.503	0.042	1.097×	0.564	1.082×	0.024	1.187×	1.305×
	Ours	0.840	15.200	1.254	**0.055**	**1.432**×	**0.670**	**1.287**×	**0.037**	**1.843**×	**1.000**
Medium SonicBOOM	Human Efforts	1.194	25.600	1.933	0.047	–	0.618	–	0.029	–	–
	ISCA'14	1.236	19.600	1.624	**0.063**	**1.353**×	**0.761**	**1.233**×	**0.048**	**1.667**×	5.688×
	DAC'16	1.376	25.400	1.925	0.054	1.161×	0.715	1.157×	0.039	1.344×	4.697×
	ICCAD'21	1.445	27.100	2.158	0.053	1.144×	0.670	1.084×	0.036	1.240×	1.279×
	Ours	1.287	20.600	1.735	0.062	1.340×	0.742	1.201×	0.046	1.610×	**1.000**
Large SonicBOOM	Human Efforts	1.487	44.600	3.206	0.033	–	0.464	–	0.015	–	–
	ISCA'14	1.490	30.900	2.542	0.048	1.446×	0.586	1.263×	0.028	1.827×	5.892×
	DAC'16	1.492	32.400	2.674	0.046	1.381×	0.558	1.202×	0.026	1.661×	4.865×
	ICCAD'21	1.916	40.900	3.672	0.047	1.405×	0.522	1.125×	0.024	1.581×	1.325×
	Ours	1.588	31.400	2.564	**0.051**	**1.517**×	**0.619**	**1.335**×	**0.031**	**2.025**×	**1.000**
Mega SonicBOOM	Human Efforts	1.950	57.800	4.806	0.034	–	0.406	–	0.014	–	–
	ISCA'14	2.496	56.600	5.368	0.044	1.307×	0.465	1.146×	0.021	1.498×	5.544×
	DAC'16	2.500	56.200	5.380	0.044	1.318×	0.465	1.145×	0.021	1.510×	4.578×
	ICCAD'21	2.482	60.700	4.701	0.041	1.212×	**0.528**	**1.301**×	0.022	1.578×	1.247×
	Ours	2.523	55.700	5.251	**0.045**	**1.343**×	0.480	1.184×	**0.022**	**1.590**×	**1.000**
Giga SonicBOOM	Human Efforts	1.872	71.600	5.069	0.026	–	0.369	–	0.010	–	–
	ISCA'14	2.253	62.200	6.001	0.036	1.386×	0.375	1.017×	0.014	1.409×	5.632×
	DAC'16	2.252	77.300	5.600	0.029	1.115×	**0.402**	**1.089**×	0.012	1.214×	4.651×
	ICCAD'21	2.265	74.500	5.865	0.030	1.163×	0.386	1.046×	0.012	1.216×	1.267×
	Ours	2.269	59.500	5.746	**0.038**	**1.459**×	0.395	1.070×	**0.015**	**1.560**×	**1.000**

[1] "–" denotes not applicable.

Table 4.2: Comparison with Human Efforts and Prior Arts

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 (Perf × Perf)/(Power × 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.

4.5.6 Analysis w. More Workloads

We analyze explored microarchitectures with more workloads to study how RL solutions outperform other methods and human implementations. Figure 4.11 and Figure 4.12 list related results for Rocket and different scales of SonicBOOM.

In the case of Small SonicBOOM, our solution demonstrates an average improvement of 8.80%, 8.38%, and 16.86% in Perf/Power, Perf/Area, and (Perf × Perf)/(Power × Area), respectively, compared to state-of-the-art approaches. For Medium SonicBOOM, our solution shows improvements of 10.03% and 14.81% in performance per watt and PPA trade-off respectively. In Mega SonicBOOM, our solution achieves an average PPA trade-off improvement of 6.13% over baseline algorithms, while for Giga SonicBOOM, the corresponding improvement is 14.27%. Notably, in certain workloads, our solution prioritizes power and area trade-offs, leading to higher performance per watt or increased area efficiency. In larger scales of SonicBOOM, our solution obtains an average of higher performance, demonstrating that our RL agent can target different preferred solutions by given PPA design goals.

Specifically, for Large SonicBOOM, the areas for human implementations, ISCA'14, DAC'16, ICCAD'21 and ours are 3.21, 2.54, 2.67, 3.67, and 2.56 in mm^2, respec-

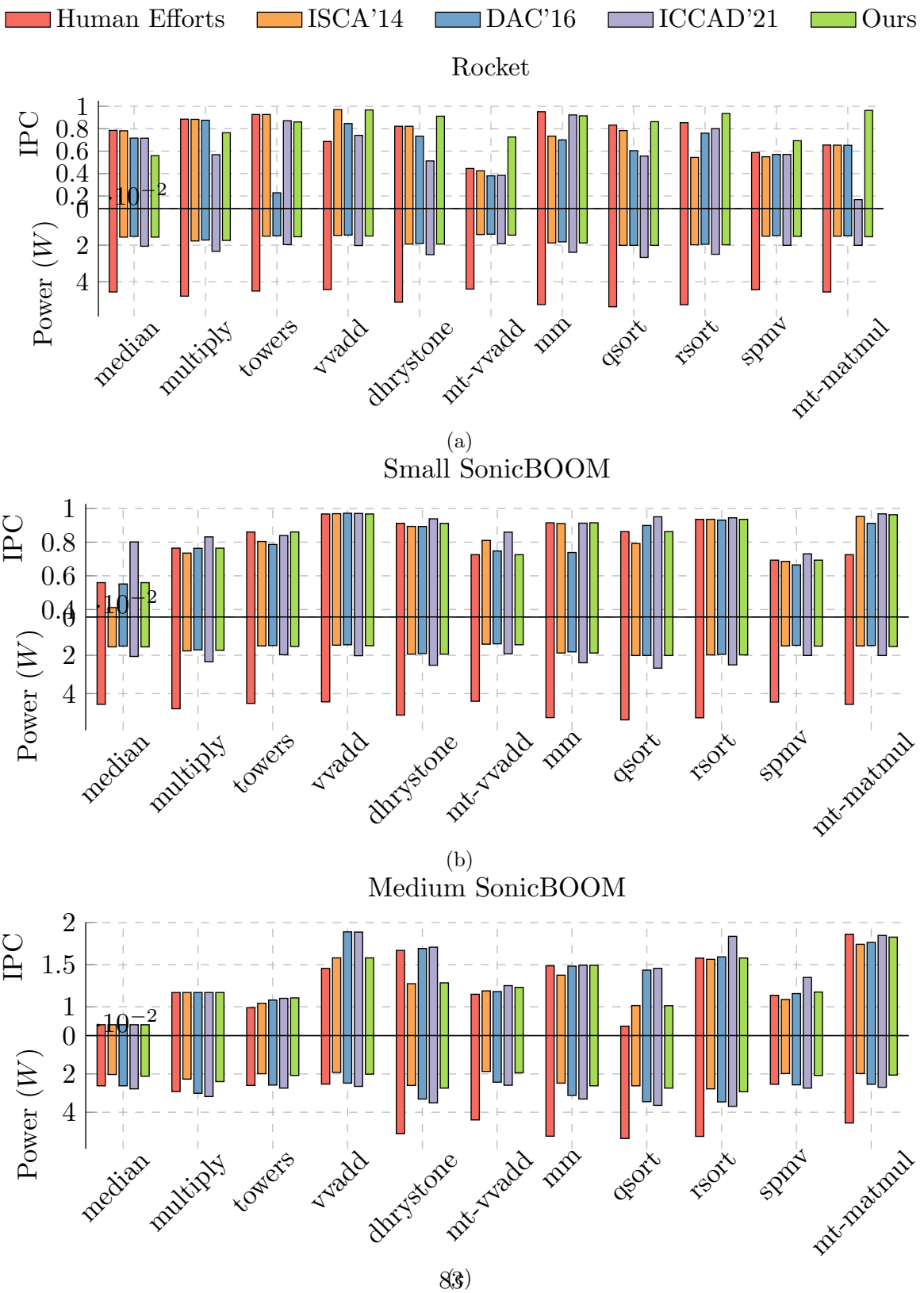


Figure 4.11: Analysis w. more workloads on Rocket and different scales of SonicBOOM I.

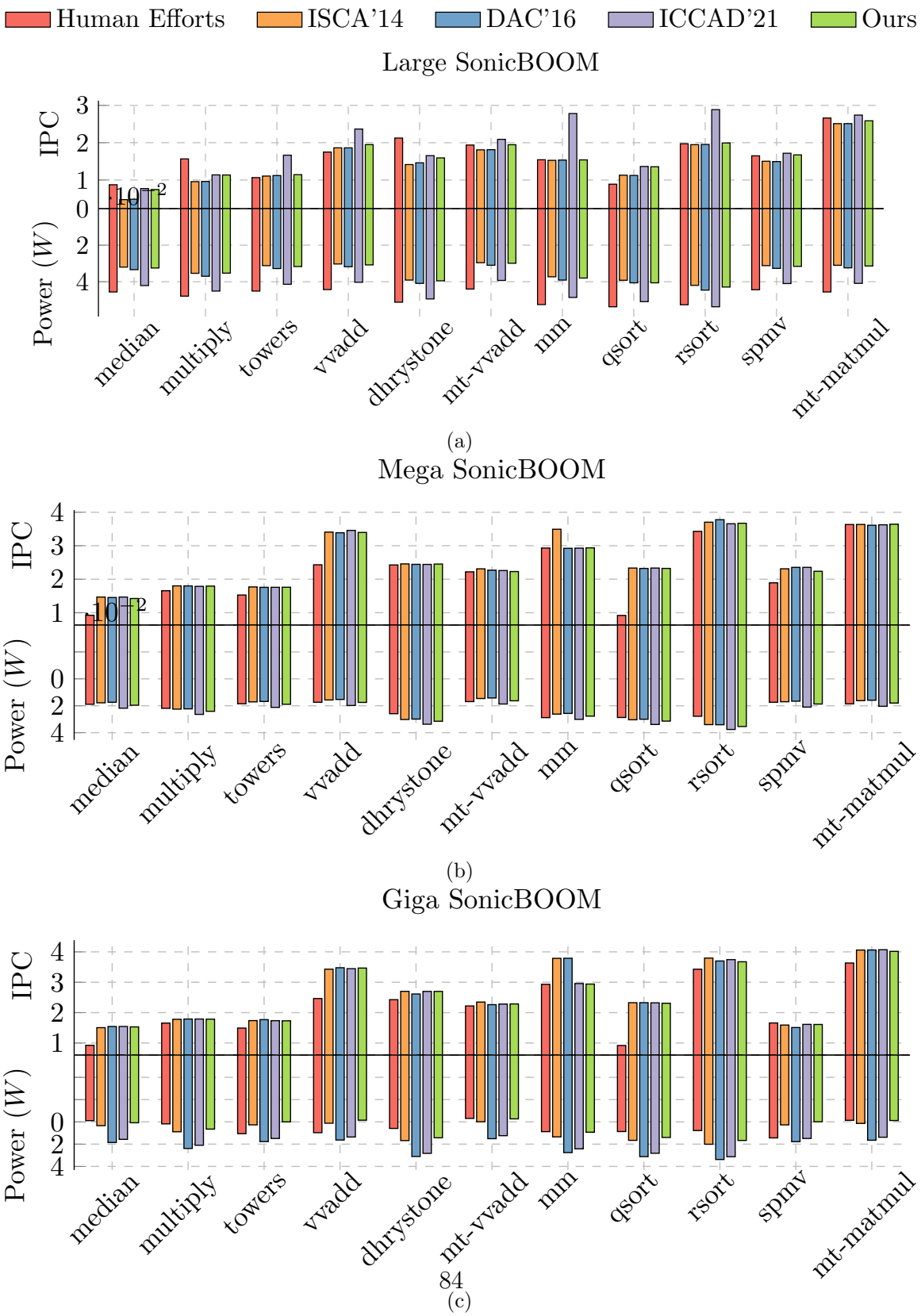


Figure 4.12: Analysis w. more workloads on different scales of SonicBOOM II.

tively. 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.

4.6 Summary

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. And the solution qualities outperform human implementations by at most 2.03× in the PPA trade-off.

Chapter 5

Microarchitecture DSE Open Benchmarking Platform

5.1 Introduction

The chip development cycle involves many steps, including architecture specification, hardware design implementation, logic synthesis, place and routing, verification, *etc.*, which requires a high workforce and cost input [162]. Microarchitecture exploration, *i.e.*, making decisions on various structures of processor components, aiming to deliver a product that can balance performance, power, and area (PPA) well, is the critical step in the chip development cycle. It is often conducted on software simulators at the early design stage, targeting to study different trade-offs among various components at a coarse granularity.

As technology node advances, the design complexity of a chip (# gates / cm^2) continues to grow at a compound annual growth rate (CAGR) of around 58%, while

the design productivity (# gates / staff-month) is at approximately 23% [2]. Inspired by the agile development paradigm in software engineering, which facilitates fast product prototype delivery, agile chip design is empowered by high-level hardware description language to mitigate the gap between the design complexity and productivity [117, 154, 127]. We can conduct microarchitecture design space exploration at the circuit level via electronic design automation tools following the agile chip design paradigm [28]. Such exploration is conducted at a finer granularity, provides more optimization opportunities at the front-end design stage when software simulators are not constructed, and alleviates the non-recurring engineering costs in the back-end design stage.

The problem is non-trivial to solve due to two difficulties. On the one hand, the microarchitecture design space is large. In industry, although the design space can be pruned by expert knowledge from CPU architects, the left solution space size is still relatively large to handle [133, 103]. On the other hand, evaluating a single microarchitecture requires a high runtime cost. Because of these challenges, researchers and engineers have proposed various methodologies to figure them out by using analytical models [101] or data-driven black-box models [91, 115, 118, 28].

To boost electronic design automation (EDA) research, we build up microarchitecture DSE open benchmarking platform for the contest of computer-aided design (CAD) at the International Conference on Computer-Aided Design (ICCAD) [6]. The ICCAD CAD Contest allows industrial companies to share various design problems and cases. At the same time, it encourages researchers in academia to study the state-of-the-art integrated circuits design challenges and advance problem-solving techniques. We serve as the topic chair of the ICCAD CAD Contest in 2022, formulate

the research topic of microarchitecture exploration as a contest problem, and deliver a dataset based on RISC-V Berkeley Out-of-Order Machine (BOOM) [22, 44, 214] and a microarchitecture DSE open benchmarking platform for all contestants [49]. Contestants can innovate and estimate their algorithms using the dataset and platform. We hope the contest platform can promote a research passion for the research topic and expect to see practical algorithms proposed by researchers can help the community and industry embrace more profits from the microarchitecture exploration.

The remainder of this chapter is organized as follows. Section 5.2 presents the contest objective. Section 5.3 provides detailed descriptions of our open benchmarking platform. Section 5.4 illustrates the evaluation with our benchmarking platform. Finally, Section 5.5 summarizes this chapter.

5.2 Contest Objective

This contest problem aims to develop a practical, efficient, and accurate microarchitecture design space exploration algorithm. In this contest, we provide large-scale microarchitecture benchmarks to evaluate contestants' solutions. We expect novel ideas to be inspired and applied in industrial product delivery. We also hope that this problem can facilitate innovative researches on microarchitecture design space exploration.

5.2.1 Problem Formulation for CAD Contest

We use a vector $\boldsymbol{x}$ to denote a microarchitecture embedding (Section 5.3.1), the element of which specifies assigned hardware resources of a component for the mi-

croarchitecture.

Based on Definition 4, our problem is formulated, as shown below.

Problem 3 (Microarchitecture Design Space Exploration)**.** *Given a design space* $\mathcal{D} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$*, experiment* f *maps the design space* $\mathcal{D}$ *to PPA metric value space* $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_n\}$*. The microarchitecture design space exploration is to find* $\boldsymbol{X} \subseteq \mathcal{D}$*, whose objective values are* $\mathcal{P}(Y)$ *as far as possible.*

5.3 Benchmark Suite

We construct a design space for RISC-V BOOM. The design space contains $15,633$ different microarchitectures and corresponding PPA metric values. In this section, we detail the microarchitecture design space and the data set format.

5.3.1 Microarchitecture Design Space in the Benchmark Suite

We divide RISC-V BOOM microarchitectures into nine boxes *w.r.t.* their functions for convenience in determining the design space. Moreover, we have leveraged expert knowledge to prune the microarchitecture design space and remove illegal designs, *i.e.*, designs that fail to generate a register-transfer-level model. The microarchitecture design space is as listed in Table 5.1.

Each number in Table 5.1 is an index, which specifies the detailed parameters, as listed in Table 5.2 and Table 5.3. For example, if the ISU chooses an number 2 in Table 5.1, then the structure of the ISU is MEM.DW = MEM.IW = INT.DW = INT.IW = FP.DW = FP.IW = 1, and MEM.QE = INT.QE = FP.QE = 6. There are $15,633$ combinations of different boxes according to Table 5.1, *i.e.*, $15,633$ RISC-V

Table 5.1: Microarchitecture Design Space of RISC-V BOOM

Design	Boxes									Total
	Fetch	Decoder	ISU	IFU	ROB	PRF	LSU	I-cache/MMU	D-cache/MMU	
sub-design-1	1	1	$1, 2, 3$	$1, 2, 3$	$1, 2, 3, 4$	$1, 2, 3$	$1, 2, 3$	$1, 2, 3, 4$	$1, 2, 3, 4$	
sub-design-2	1	2	$4, 5, 6$	$4, 5, 6$	$5, 6, 7$	$4, 5, 6$	$4, 5, 6$	$1, 2, 3, 4$	$1, 2, 3, 4$	
sub-design-3	2	3	$7, 8, 9$	$7, 8, 9$	$8, 9, 10$	$7, 8, 9$	$7, 8, 9$	$5, 6, 7$	$5, 6, 7$	15633
sub-design-4	2	4	$10, 11, 12$	$10, 11, 12$	$11, 12, 13$	$10, 11, 12$	$10, 11, 12$	$5, 6, 7$	$8, 9, 10$	
sub-design-5	2	5	$13, 14, 15$	$12, 13, 14$	$14, 15, 16$	$11, 12, 13$	$10, 11, 12$	$5, 6, 7$	$8, 9, 10$	

BOOM microarchitectures are created.

5.3.2 Dataset Format

The dataset is a two-dimensional matrix with the shape of $n \times (m+4)$, where n is the total number of microarchitectures, *i.e.*, 15633. The number m equals the dimensions of a concatenated vector. Specifically, the concatenated vector is formulated from different microarchitecture features, as shown in Equation (5.1)

$$V = CONCAT(\text{FetchWidth},\ \text{DecoderWidth},\ ...,\ \text{D-TLBWays}), \tag{5.1}$$

where FetchWidth, DecoderWidth, *etc.*, are values obtained from Table 5.2 and Table 5.3 *w.r.t.* corresponding columns. The four additional columns denote the VLSI flow's performance, power, area, and runtime cost.

The PPA metric values of each microarchitecture are obtained from electronic design automation tools, a specific technology node, and a set of benchmarks. The performance and power values are the averages of the benchmarks since a microarchitecture should be optimized against various benchmarks/applications rather than a specific test case. The runtime cost of the VLSI flow includes RTL generation, logic synthesis, netlist simulation, power analysis, *etc.* For different scales of microarchitectures, the runtime cost varies. Since the dataset is constructed before performing the design space exploration, optimizers mimic the VLSI flow by accessing the dataset to retrieve the PPA metric values and the VLSI runtime cost.

Table 5.2: Components I

Index	Fetch	Decoder	LSU [1]		I-cache/MMU				D-cache/MMU					
	Width	Width	LDQ	STQ	Sets	Ways	I-TLBSets	I-TLBWays	Sets	Ways	RP [2]	MSHR	D-TLBSets	D-TLBWays
1	4	1	8	8	64	4	1	32	64	4	0	2	1	8
2	8	2	6	6	32	8	1	32	64	4	0	2	1	32
3		3	12	12	32	4	1	16	64	6	1	2	1	8
4		4	16	16	64	1	1	16	64	4	1	2	1	32
5		5	12	12	64	8	1	32	64	2	0	4	1	32
6			20	20	64	8	2	16	64	4	1	4	1	32
7			24	24	64	8	2	32	64	4	1	8	1	32
8			22	22	64	8	2	32	64	8	0	8	2	32
9			28	28					64	8	1	8	2	32
10			32	32					64	8	1	8	1	32
11			28	28										
12			36	36										

[1] LDQ and STQ are shored for load and store queue entries, respectively.
[2] RP is shorted for a replacement policy. Specifically, 0 denotes LRU, and 1 represents Pseudo LRU.

Table 5.3: Components II

Index	ISU [1]									IFU [2]			ROB	PRF [3]	
	MEM.DW	MEM.IW	MEM.QE	INT.DW	INT.IW	INT.QE	FP.DW	FP.IW	FP.QE	Tag	FBE	FTQ	Entries	INT	FP
1	1	1	8	1	1	8	1	1	8	8	8	16	32	52	48
2	1	1	6	1	1	6	1	1	6	6	6	14	30	42	38
3	1	1	10	1	1	12	1	1	12	10	12	20	34	62	58
4	2	1	12	2	2	20	2	1	16	12	16	32	36	80	64
5	2	2	12	2	2	20	2	2	16	10	14	30	64	70	54
6	2	2	14	2	2	24	2	2	20	14	20	36	60	90	74
7	3	1	16	3	3	32	3	1	24	16	24	32	72	100	96
8	3	2	16	3	3	32	3	2	24	14	21	30	96	90	86
9	3	2	20	3	3	36	3	3	28	18	30	36	90	110	106
10	4	2	24	4	4	40	4	2	32	20	32	40	108	118	118
11	4	2	28	4	4	44	4	4	36	22	36	44	128	128	128
12	4	2	22	4	4	36	4	2	28	24	40	48	132	138	138
13	5	2	24	5	5	40	5	2	32	20	35	40	136	146	146
14	5	2	26	5	4	44	5	4	36	26	45	50	130		
15	5	2	28	5	5	48	5	5	40				120		
16													140		

[1] DW, IW, and QE are shorted for dispatch width, issue width, and queue entries.
[2] FBE and FTQ are shorted for fetch buffer entries and fetch target queue entries, respectively.
[3] INT and FP are shorted for the number of integer and floating-point physical registers, respectively.

5.4 Evaluation

This section introduces the microarchitecture DSE open benchmarking platform and evaluation metrics for contestants' submissions.

5.4.1 Overview of Benchmarking Platform

We provide a microarchitecture DSE open benchmarking platform for the problem to facilitate convenient implementation of the solver without considering the cumbersome handling procedure of the input and output. The open benchmarking platform is open-sourced to the public, allowing contestants to check the overall design space exploration flow. Contestants can easily install the benchmarking platform via "pip3 install iccad-contest".

Figure 5.1 illustrates the design space exploration flow *w.r.t.* online and offline cases. The offline optimization flow constructs a model once and uses the model to sweep the design space and retrieve the target microarchitectures, as shown in Figure 5.1a. Since the computation runtime with the model for a design is much lower than the VLSI flow, we can efficiently sweep the design space. According to Figure 5.1b, the online optimization flow constructs a model with an initial design set and corresponding PPA metric values. The flow samples designs from the solution space according to the model. The PPA metric values of sampled designs are then obtained from the VLSI flow. We can tune the model on a design set with known PPA metric values, *i.e.*, designs which have already been estimated with the VLSI flow, including the initial designs. If the algorithm's termination condition is satisfied, the Pareto-optimal microarchitectures are acquired from already explored designs.

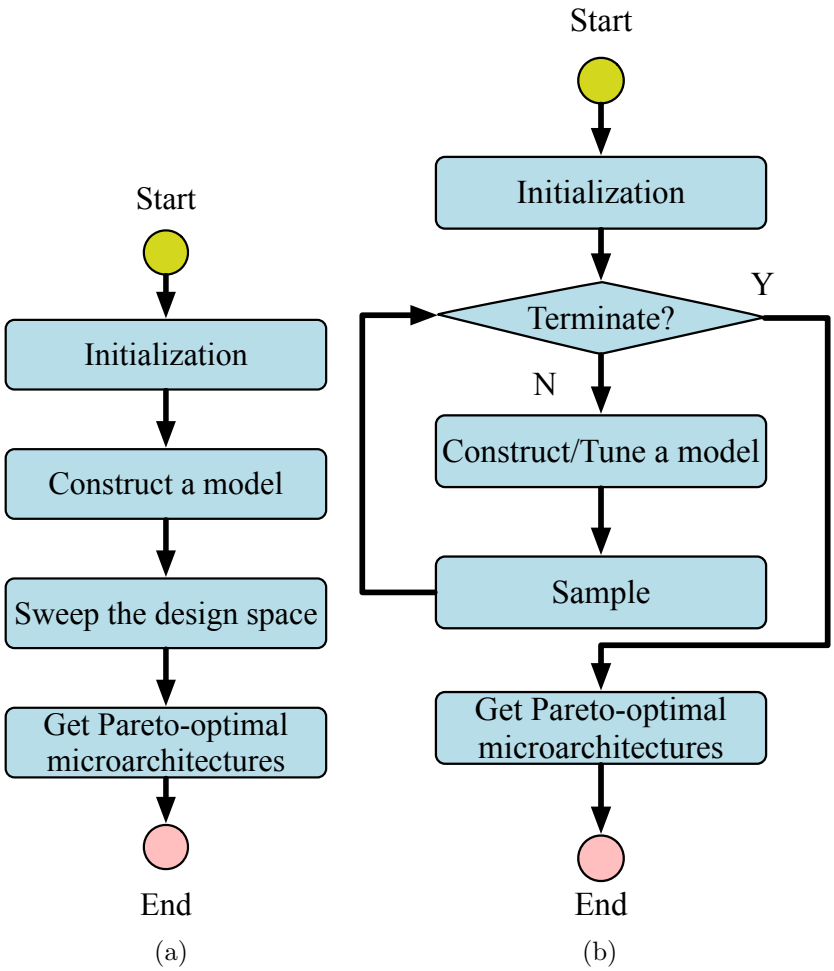


Figure 5.1: (a) Offline design space exploration flow. (b) Online design space exploration flow.

5.4.2 Benchmarking Platform Solution Implementation

Contestants should implement the solver with the contest benchmarking platform via Python programming language.

```
from iccad_contest.abstract_optimizer import AbstractOptimizer
from iccad_contest.design_space_exploration import experiment

class YourAlgorithm(AbstractOptimizer):
    primary_import = "iccad_contest"

    def __init__(self, design_space):
        """
            build a wrapper class for an optimizer.

            parameters
            ----------
            design_space: <class "MicroarchitectureDesignSpace">
        """
        AbstractOptimizer.__init__(self, api_config)
        # do whatever other setup is needed
        # ...

    def suggest(self):
        """
            get a suggestion from the optimizer.
```

```
22
23              returns
24              -------
25              next_guess: <list> of <list>
26                  list of `self.n_suggestions` suggestion(s).
27                  each suggestion is a microarchitecture embedding.
28          """
29          # do whatever is needed to get the parallel guesses
30          # ...
31          return x_guess
32
33      def observe(self, X, y):
34          """
35              send an observation of a suggestion back to the optimizer.
36
37              parameters
38              ----------
39              x: <list> of <list>
40                  the output of `suggest`.
41              y: <list> of <list>
42                  corresponding values where each `x` is mapped to.
43          """
44          # update the model with new objective function observations
45          # ...
46          # no return statement needed
```

```python

if __name__ == "__main__":
    # this is the entry point for experiments, so pass the class to
    ↪ `experiment_main_entry` to use this optimizer.
    # this statement must be included in the wrapper class file:
    experiment(YourAlgorithm)
```

Code 5.1: Template of the solution implementation using `AbstractOptimizer` as the base class.

```python
descriptions = {
        "sub-design-1": {
                "Fetch": [1],
                "Decoder": [1],
                "ISU": [1, 2, 3],
                "IFU": [1, 2, 3],
                ...
        }
}

components_mappings = {
        "Fetch": {
                "description": ["FetchWidth"],
                "1": [4]
        },
        "Decoder": {
```

```
17                  "description": ["DecodeWidth"],
18                  "1": [1]
19          },
20          "ISU": {
21                  "description": [
22                          "MEM_INST.DispatchWidth", "MEM_INST.IssueWidth"
23                          "MEM_INST.NumEntries", "INT_INST.DispatchWidth",
24                          "INT_INST.IssueWidth", "INT_INST.NumEntries",
25                          "FP_INST.DispatchWidth", "FP_INST.IssueWidth",
26                          "FP_INST.NumEntries"
27                  ],
28                  "1": [1, 1, 8, 1, 1, 8, 1, 1, 8],
29                  "2": [1, 1, 6, 1, 1, 6, 1, 1, 6],
30                  "3": [1, 1, 10, 1, 1, 12, 1, 1, 12]
31          },
32          "IFU": {
33                  "description": ["BranchTag", "FetchBufferEntries",
                    ↪  "FetchTargetQueue"]
34                  "1": [8, 8, 16],
35                  "2": [6, 6, 14],
36                  "3": [10, 12, 20]
37          },
38          ...
39  }
```

Code 5.2: Data structure definition of the design space.

Listing 5.1 details the template of the solution implementation with the contest benchmarking platform. Contestants need to implement an optimizer inherited from `AbstractOptimizer`. The base class provides two critical functions, *i.e.*, `suggest` and `observe`. The `suggest` function generates samples, while will be evaluated with the VLSI flow, *i.e.*, access the dataset introduced in Section 5.3.2. The `observe` takes action the benchmarking platform retrieves the PPA metric values from suggestions provided by `suggest`. Within `observe`, contestants can visualize each suggestion's PPA metric values and design the optimization strategy accordingly. The base class also provides a variable to control the early stopping criterion for contestants, *i.e.*, by setting the variable, the optimizer can terminate before the pre-determined stopping iteration numbers.

```
$ python3 random-search-optimizer.py -h

usage: random-search-optimizer.py [-h] [-o OUTPUT_PATH] [-u UUID] [-s
↪  SOLUTION_SETTINGS] [-q NUM_OF_QUERIES]

ICCAD'22 Contest Platform - solutions evaluation

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT_PATH, --output-path OUTPUT_PATH contest output path
↪  specification
  -u UUID, --uuid UUID  universally unique identifier (UUID) specification
```

```
  -s SOLUTION_SETTINGS, --solution-settings SOLUTION_SETTINGS solution submission specification
  -q NUM_OF_QUERIES, --num-of-queries NUM_OF_QUERIES the number of queries specification
```

Code 5.3: Benchmarking platform help menu.

```
$ python3 random-search-optimizer.py -o output -u "00ef538e88634ddd9810d034b748c24d" -q 20

...
[INFO]: summary for the solution, the best Pareto hypervolume: 37.59654046710655, the best Pareto hypervolume difference: 66.67984662813456 cost: 164902.3422778734.
...
```

Code 5.4: Example command of benchmarking platform.

5.4.3 Application Programming Interface for Design Space

The application programming interface (API) for microarchitecture design space is accessed via `self.design_space` in the base class `AbstractOptimizer`.

The variable `self.design_space` provides a data structure to organize the design space as illustrated in Section 5.3.1.

The design space is automatically parsed to construct the data structure by the benchmarking platform, as shown in Listing 5.2. Contestants leverage APIs offered by

the benchmarking platform to access the data structure. Furthermore, they can define dedicated operations with the data structure in their optimizer implementations.

5.4.4 Benchmarking Platform Command Usage

We provide a series of example optimizers for contestants to familiarize themselves with the benchmarking platform. These example optimizers include exploration with online/offline linear models, Gaussian process models, *etc.*

Contestants can benchmark their implementations with a few arguments. Assume the source code of the implemented optimizer is with the name "random-search-optimizer.py". The help menu is as shown in Listing 5.3

According to the help menu, an example command to benchmark the implementation is shown in Listing 5.4.

The command will initialize the benchmarking platform with a random seed provided by the "-u" option and query the dataset 20 times. It can generate many meta information and the implementation results, which are saved inside the "output" directory.

5.4.5 Benchmarking Platform Evaluation Metrics

The optimizers implemented by contestants are evaluated with two metrics, *i.e.*, Pareto hypervolume difference and the overall running time (ORT).

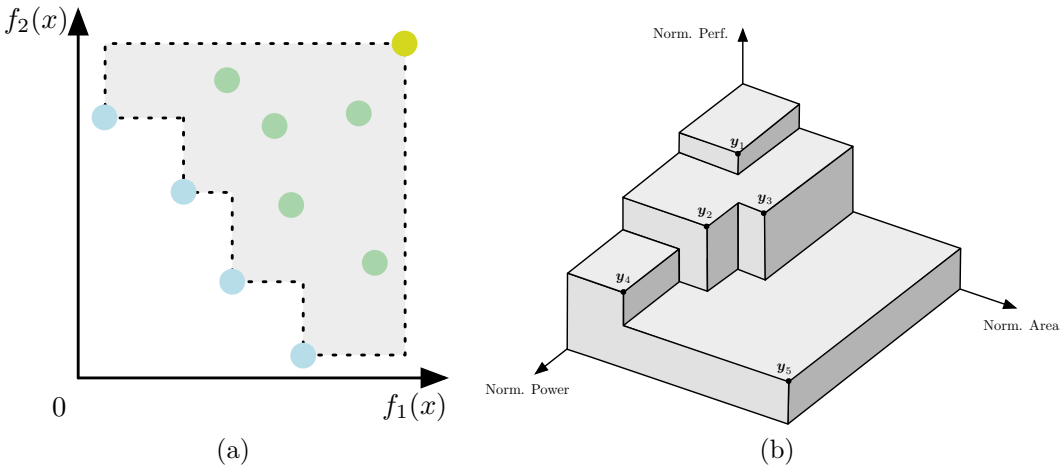


Figure 5.2: (a). An example overview of the Pareto hypervolume in the two dimensional space (b). An example overview of the Pareto hypervolume in the three dimensional space, *i.e.*, power, performance, and area.

Pareto Hypervolume

Pareto hypervolume is the *Lebesgue measure* of the space dominated by the Pareto-optimal set $\mathcal{P}(Y)$ bounded by a reference point $\boldsymbol{v}_{\text{ref}}$ [28], as shown in Equation (5.2).

$$\text{PVol}_{\boldsymbol{v}_{\text{ref}}}(\mathcal{P}(Y)) = \int_{Y} \mathbb{1}[\boldsymbol{y} \not\succeq \boldsymbol{v}_{\text{ref}}][1 - \prod_{\boldsymbol{y}_* \in \mathcal{P}(Y)} \mathbb{1}[\boldsymbol{y}_* \not\succeq \boldsymbol{y}]]\text{d}\boldsymbol{y}, \tag{5.2}$$

where $\mathbb{1}(\cdot)$ is the indicator function, which outputs 1 if its argument is true and 0 otherwise. The integral characterized by Equation (6.3) sums up all regions bounded by $\mathcal{P}(Y)$ and $\boldsymbol{v}_{\text{ref}}$.

Figure 5.2a visualizes the Pareto hypervolume in $f_1 - f_2$ two-dimensional space. Suppose we want to minimize f_1 and f_2. Thus, we say x is better if its corresponding point $(f_1(x), f_2(x))$ is closer to the coordinate origin. The orange point is a reference point $\boldsymbol{v}_{\text{ref}}$. Four points are colored in purple and denote non-dominated points; green points are dominated points. The area shaded with gray is the Pareto hyper-

volume. Intuitively, if a new point is searched out and not dominated by any points, $\mathrm{PVol}_{\boldsymbol{v}_{\mathrm{ref}}}(\mathcal{P}(\boldsymbol{Y} \cup \{\boldsymbol{y}_{\mathrm{new}}\}))$ is increased, *i.e.*, the shaded area is enlarged. In our contest problem, we are dealing with PPA metric values. A better microarchitecture has higher performance, but lower power dissipation and smaller area. Figure 5.2b visualizes the Pareto hypervolume in the PPA metric space. The contest platform has normalized all PPA values for microarchitectures. Thus, a better design has higher normalized performance, power, and area values.

Overall Running Time

Overall running time (ORT) measures the total time of algorithms, including the submission of contestants' algorithms and the time spent on the VLSI flow.

Total Score

Based on industry experience, we design an approximate scoring function *w.r.t.* Pareto hypervolume difference and ORT.

$$\mathrm{score} = \mathrm{PVol}_{\boldsymbol{v}_{\mathrm{ref}}} \cdot \begin{cases} \alpha - \dfrac{\mathrm{ORT} - \theta}{\theta}, \mathrm{ORT} \geq \theta \\ \alpha + |\dfrac{\mathrm{ORT} - \theta}{\theta}|, \mathrm{ORT} < \theta \end{cases}, \tag{5.3}$$

where α is an ORT score baseline, equal to 6, and θ is a pre-defined ORT budget, equivalent to 2625000. It is worth noting that if the ORT is six times larger than θ, then the final score will be negative. Hence, a better solution achieves a better total score, *i.e.*, lower Pareto hypervolume difference and ORT as much as possible.

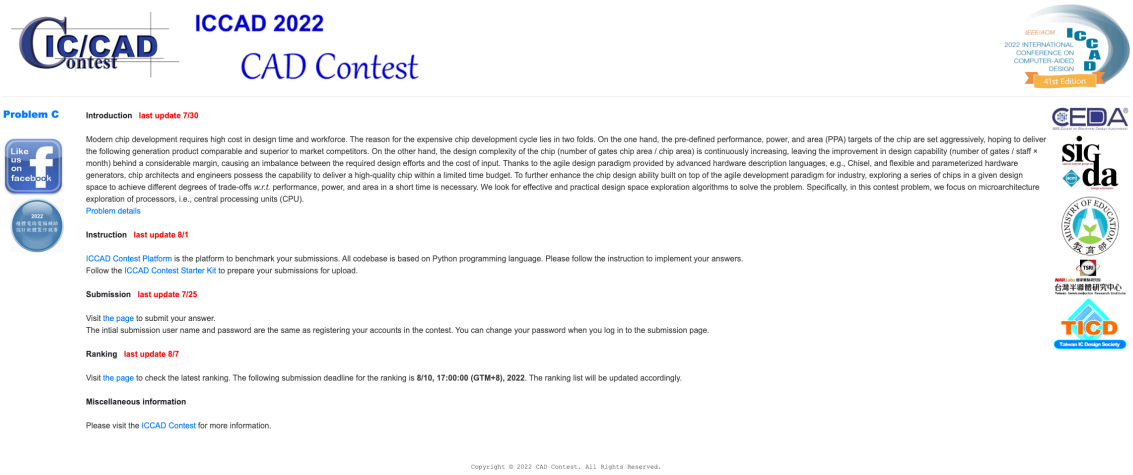


Figure 5.3: An overview of the online ranking.

5.4.6 Online Ranking

We provide contestants with an online ranking platform since the dataset discussed in Section 5.3.2 is not disclosed to the public. The ranking platform, as shown in Figure 5.3, accepts contestants' answers submissions, and releases corresponding results, *i.e.*, Pareto hypervolume difference, ORT, and the ranking among the participated contest teams to the public. Contestants can submit their optimizers multiple times and adjust the implementation strategies according to the published results.

5.5 Summary

Microarchitecture design space exploration is an important research topic. We formulate the topic as a ICCAD CAD contest problem. To facilitate the contest problem, we provide a benchmark suite, a benchmarking platform, and the ranking platform for contestants. We expect a practical, efficient, and accurate solution to further research on this topic.

Chapter 6

ArchExplorer

6.1 Introduction

The microprocessor design cycle at the concept phase (logic level details are unavailable) requires abundant performance-power-area (PPA) trade-off evaluations. Architects rely on practical algorithms to explore optimal architecture parameters, achieving sweet PPA balance within a limited time budget. For example, five of Tenstorrent's competitive RISC-V microprocessor implementations are fast prototyped based on one design within a year to face diverse PPA design targets [8].

Microarchitecture exploration is a design space exploration (DSE) problem aiming to find performance-power-area Pareto-optimal microprocessor parameters. Architects are often confronted with billions or trillions of parameter combinations. And one combination takes high runtime to acquire the PPA results via detailed simulations. The problem is not new, and the industry and academia have proposed many solutions.

In industry, microarchitecture parameters are determined upon extensive simulation results analysis and the expertise of architects. However, the fact that architects' domain knowledge can fall into a personal bias is a concern. The question remains whether the solution given by architects is optimal or how many benefits we can gain based on a sub-optimal solution. In academia, researchers adopt mechanistic models or black-box methodologies. Mechanistic models investigate the relations between PPA values and microarchitecture parameters by unfolding the microexecutions. Interpretable equations are constructed for a single component [84, 152, 85, 182] or the entire microprocessor [141, 101, 63]. Microarchitecture exploration is conducted by sweeping the design space or fast evaluation with the mechanistic model. Nevertheless, the model requires immense domain knowledge to build and verify. More commonly-applied solutions use black-box methodologies [111, 103, 208, 39, 91, 115, 59, 25, 48, 118, 60, 28]. The methods train a black-box model with machine-learning techniques via a large data set. Researchers prefer them since better results are often achieved [91, 115, 48, 118, 28]. However, black-box methods are not a silver bullet. They require high computing resources to construct the data set for training [91, 48]. Another criticism of the black-box method is that blindly (purely driven by the algorithm rather than tightly coupled with expertise) exploring microarchitectures seems naive since architects already know the characteristics of most designs.

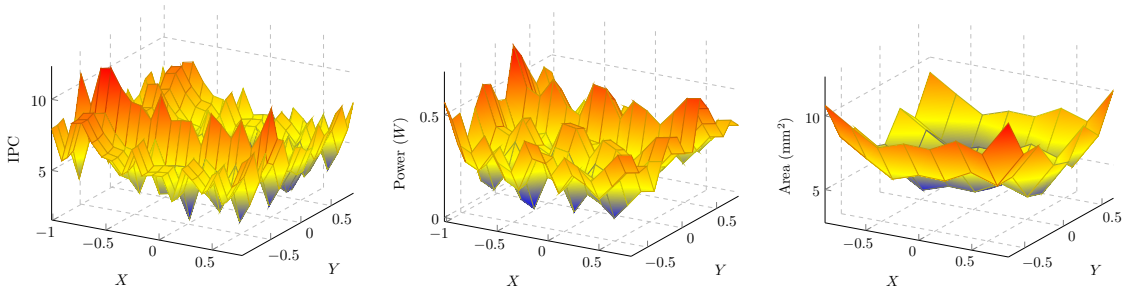
Goal and Approach: We aim to solve the problem by evading the limitations of current mainstream methodologies. Specifically, we circumvent the massive domain knowledge access required by building mechanistic models and mitigate the high computing demands of black-box methods. The key to achieving the goal is *DSE*

via automated bottleneck analysis, where the bottlenecks are the factors that hinder the program execution progress. We reduce the demand for expert knowledge via automated bottlenecks detection and elimination. Meanwhile, our method asks for fewer computing demands compared to black-box methodologies.

Rationales: Assume a perfect machine has unlimited hardware resources. The factors constraining the perfect machine's performance are only the program's true data dependencies (*viz.*, read-after-write dependencies). While in a real machine, architecture parameters like the instruction queue entries set the resource constraints. Due to the constraints, contentions for limited resources exist in the microexecution. And the contention produces two distinct types of resources: 1) deficient and exhausted and 2) abundant and idle. Hence, a real machine leads to unbalanced usage of resources. The usage dependencies of deficient resources are another factor that blocks instructions from progressing. A *balanced microarchitecture* can simultaneously maximize the utilization of each hardware resource. In other words, the microarchitecture makes the best use of every resource. We refer to a bottleneck as insufficient hardware resource, which is exhausted by instructions and results in high program runtime. Accurate and efficient identification of types of hardware resources is the first principle to finding a balanced microarchitecture.

Findings and Design Principles: Jouppi decoupled the microprocessor performance into benchmark and machine parallelism [95]. Our observation is that the relations between resource constraints and machine parallelism are similar to the cask effect [1]. Namely, the most insufficient resource largely determines machine par-

[1]The cask effect is a terminology from the Peter principle [153]. It states that in a hierarchy, every employee tends to rise to its level of incompetence. We adopt the term to broadly summarize the insight of our approach.



(a) Performance (b) Power (c) Area

Figure 6.1: A visualization of the design space for 458.sjeng. Each microarchitecture is reduced to two-dimension through t-SNE [188] to facilitate the visualization of PPA distributions.

allelism. For example, assigning more to such resources can achieve 23.05% performance improvement, and the PPA trade-off becomes 27.42% better. To identify the type of resource, two requirements should be satisfied. The utilization status of each resource in the microexecution should be captured. And whether the overlapping events matter for the execution time should be considered. The latter requirements call for a global view of the entire microexecution, which the critical path can represent [68]. We summarize two design principles for DSE via bottleneck analysis. *First, the dependencies contributing to execution time should be captured as much as possible.* Approximate the resource utilization more accurately benefits the DSE. *Second, concurrent events should be distinguishable.* The distinguishability is essential in accurately estimating bottleneck contributions to the execution time. Accordingly, we propose a new graph model formulation based on critical path analysis [116, 143, 144, 186, 75] to implement the two design principles.

Contributions:

1) The design principles and a new graph model formulation to characterize the microexecution.

2) The induced graph model and an optimal critical path construction algorithm based on dynamic programming to investigate overlapped events.

3) Evaluations show that our DSE method ArchExplorer can find better Pareto PPA microarchitectures with an average of 6.80% higher Pareto hypervolume using at most 74.63% fewer simulations compared to state-of-the-art approaches.

Section Organization: The rest of this chapter is organized as the following. Section 6.2 introduces the motivation. Section 6.3 states lessons and design principles. Section 6.4 provides the ArchExplorer approach. Section 6.5 and Section 6.6 are for experiments. Section 6.7 present some discussions. Section 6.8 supplements additional related works and Section 6.9 summarizes the chapter.

6.2 Motivation

In this section, we introduce the motivation of our approach.

6.2.1 Bottleneck Analysis Matters in DSE

Although the design space is complicated, improving machine parallelism by removing microarchitecture bottlenecks can significantly enhance the PPA trade-off, as demonstrated by an example of comparative simulations. Table 6.1 lists a baseline microarchitecture, and we evaluate it with SPEC CPU2017 Simpoints [151]. The average performance and power values of all workloads are reported. Comparative simulations are conducted for each new microarchitecture by doubling individual parameters, as shown in Figure 6.2. Firstly, doubling parameters like the number of floating-point arithmetic logic units (FpALU) worsens power and area without improving perfor-

Table 6.1: A baseline microarchitecture specification

Components	Hardware Resources
Pipeline width	4
Fetch buffer size in bytes	64
Fetch queue size in μ-ops	32
Branch predictor unit	local/global/choice predictor of the tournament: 2048/8192/8192 RAS: 16, BTB: 4096
ROB/IQ/LQ/SQ	50/32/24/24
Physical register	Int RF: 50, Floating-point RF: 50
Functional unit	IntALU: 3, IntMultDiv: 1, FpALU: 2 FpMultDiv: 1, RdWrPort: 1
L1 I$	2-way, 32 KB, 2 cycles
L1 D$	2-way, 32 KB, 2 cycles
IPC/Power/Area	0.9418/0.2027 W/5.6609 mm^2

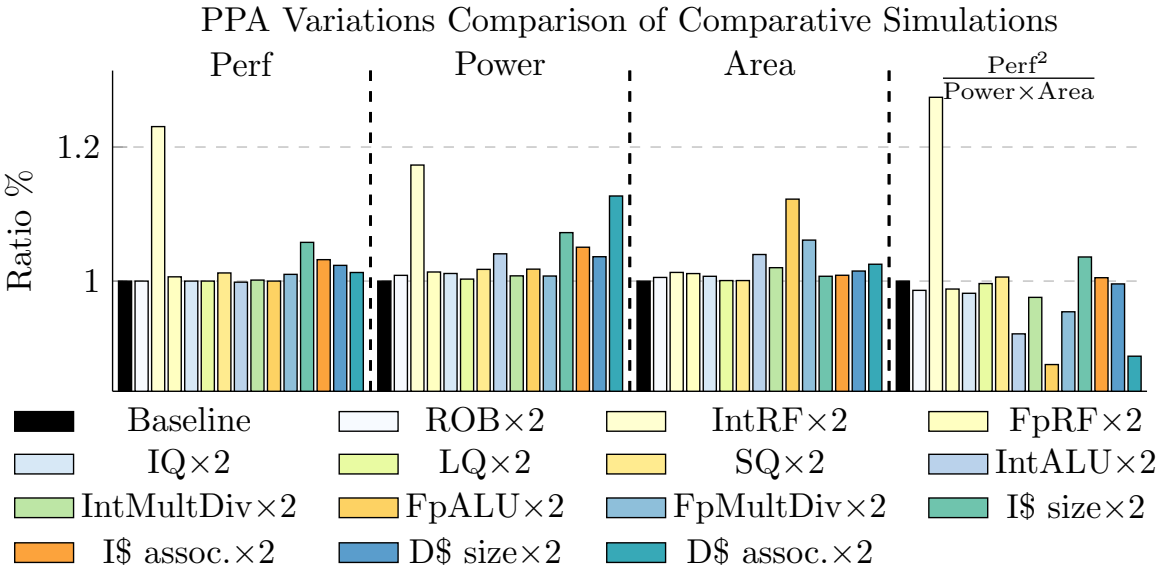


Figure 6.2: Each bar represents the microarchitecture's metric in %. The bar, *e.g.*, "ROB ×2", indicates the microarchitecture is the same as the baseline except that it doubles ROB. Perf^2/(Power × Area) denotes the PPA trade-off.

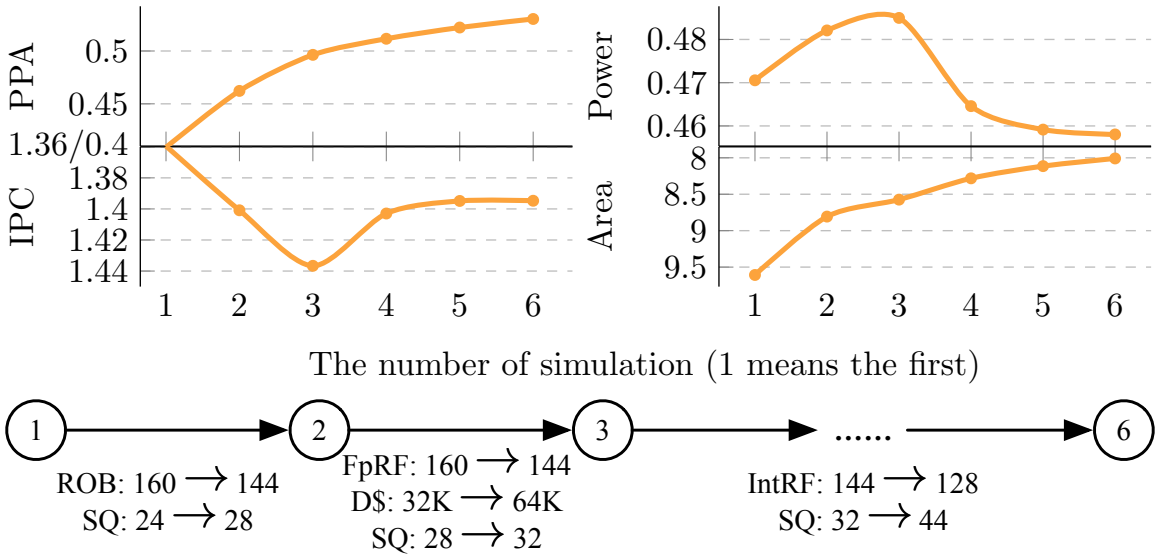


Figure 6.3: Search following series of small changes stepwise. PPA denotes $\text{Perf}^2/(\text{Power} \times \text{Area})$.

mance, indicating redundant resources waste the design budget. Secondly, doubling the number of physical integer registers (IntRF) improves performance by 23.05% and enhances the PPA trade-off by 27.42%. We investigate the simulation trace to find out the root cause. We find that most instructions are stalled in the rename stage due to insufficient physical integer registers. For instance, this bottleneck results in 25.71% of instructions in 657.xz_s and 18.94% for 625.x264_s getting stalled during renaming.

We apply a straightforward heuristic to find a balanced microarchitecture: *assigning necessary hardware resources and reducing redundant ones.* We adjust resources based on the degree of necessity, which is the ratio of delayed instructions due to the resource's insufficiency. If the necessity is top-ranked, we increase the resource, and if it is zero, we decrease it. Figure 6.3 shows a stepwise search by reusing the design space (Table 6.4), where we analyze the simulation trace manually to calculate the

necessity for each resource. With only six simulations, we improve performance by 2.59%, reduce power and area by 2.66% and 16.64%, respectively, and enhance the PPA trade-off by 29.72%.

6.2.2 Critical Path Analysis

Unlike performance counters analysis [20, 62, 205] and pipeline stall analysis (interval analysis) [61, 63], the critical path analysis can answer which events we should blame for the cycle loss regarding the entire microexecution.

The critical path analysis is a methodology based on the dynamic event-dependence graph (DEG). Since its first appearance [68], it has been widely applied [116, 143, 144, 186, 75]. A DEG is a directed acyclic graph, as shown in Figure 6.4, with vertices denoting the pipeline stages and edges representing the dependencies. Edge weights indicate delayed cycles. The longest path from the fetch of the first instruction to the commit of the last instruction is the critical path (108 cycles, highlighted in red in Figure 6.4). Critical events can be obtained from the critical path, which are dependencies contributing to the critical path's length. For instance, a D-cache miss is a critical event contributing 100 cycles to the microexecution.

Although the critical path provides a global view, it is limited to clarify which resource is deficient or abundant. *Firstly, in the former DEG formulation, the dependence and weights assignment are static without adhering to actual microexecution.* This is a significant concern since microexecution contains dynamic behaviors, such as instruction scheduling and resource usage dependencies. Previous DEG formulations statically assign edges and weights following predetermined rules without considering runtime information, leading to deviations between critical path length and actual

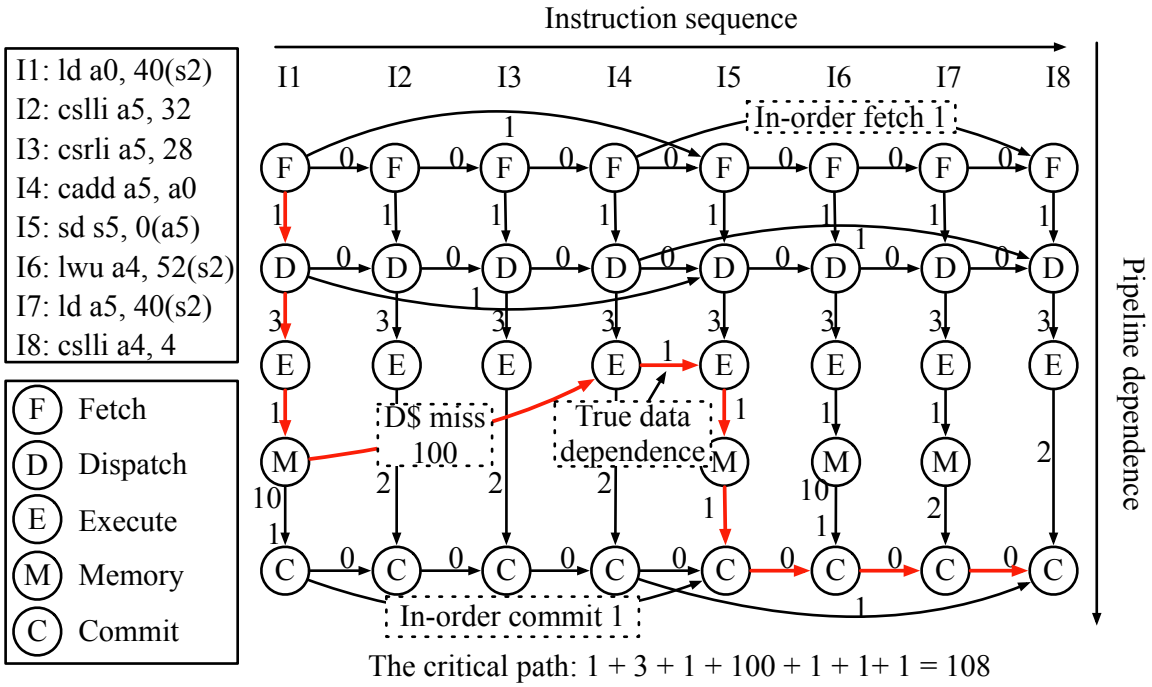


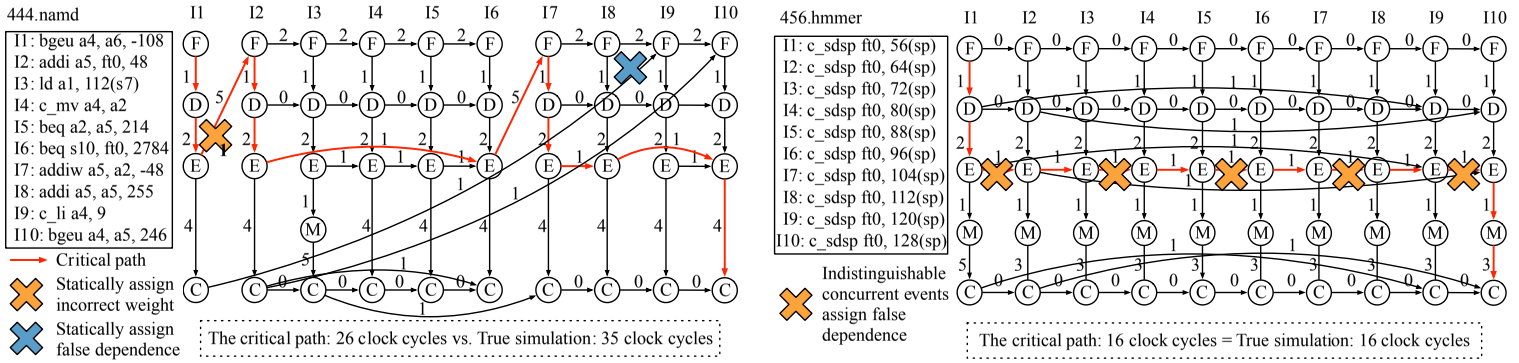
Figure 6.4: An overview of the dynamic event-dependence graph.

runtime. *Secondly, the critical path cannot accurately characterize the bottlenecks' contributions to the overall runtime, even if the modeled critical path length is strictly identical to the simulation runtime.* This is because the previous DEG formulation cannot distinguish overlapped events in microexecution, resulting in double-counting of bottleneck contributions.

6.3 Lessons Learned & Design Principles

We scrutinize the previous DEG formulation and illustrate its limitations in detail. We summarize the lessons learned and provide our design principles.

Firstly, the previous DEG formulation [68, 116, 186, 75] statically assigns weights and edges without following the actual microexecution, leading to inaccurate critical



(a) Previous DEG formulation statically assigns edges and weights without following the actual microexecution.

(b) Previous DEG formulation cannot distinguish overlapped events.

Figure 6.5: (a) and (b) uses Calipers [75], the representative DEG formulation, to demonstrate three kinds of error sources, with critical paths highlighted in red. (a) illustrates errors including incorrect weights and false dependence due to static assignment without following actual microexecutions. (b) manifests false dependence owing to indistinguishable concurrent events.

path length estimation. As shown in Figure 6.5a, we demonstrate the point with 444.namd. The first error is using a static penalty to represent variant durations. For example, a branch misprediction penalty depends on the number of in-flight instructions in the wrong execution path. In Figure 6.5a, the penalty of the first branch misprediction is 3 instead of 5. The error roots in a fundamental limitation of the former formulation, *i.e.*, the lack of events' timing in the DEG construction. The second error comes from false dependence. In Figure 6.5a, the DEG formulation inserts an edge between the commit of I1 and the fetch of I9 to denote ROB dependence, contrary to the fact that no delay exists since I1 has freed the ROB entry before I9 consumes it. Although the incorrect insertion of ROB dependence is not included in the critical path (hidden by parallel events), the length of the critical path is shorter than the actual simulation time due to the lack of correct pipeline dependence delay. The ignorance of events timing information in the formulation leads to underestimating the critical path's length by 25.71%.

Secondly, bottlenecks' contributions are incorrectly estimated due to the inability to distinguish concurrent events. Figure 6.5b demonstrates the third error with 456.hmmer. Consecutive execution stages are connected (E(I1) $\rightarrow$ E(I10)) to denote read/write port contention. The contention contributes nine cycles to the execution time according to the critical path. However, the contribution is four cycles in the actual microexecution. The formulation overestimates the read/write port's insufficiency by 125%. The redundant five cycles are hidden by parallel events, *e.g.*, I3 and I4 get their read/write ports simultaneously.

Embedding the events' timing information and finding a way to distinguish concurrent events is crucial for making the DEG formulation practical. This approach

enables the critical path to accurately identify whether a resource is deficient or abundant. Therefore, we provide two design principles for DSE via bottleneck analysis: 1) *The dependencies contributing to execution time should be captured as much as possible.* Capturing more resource usages improves the utilization approximation. 2) *Concurrent events should be distinguishable.* The distinguishability unveils whether we matter a concurrent event for bottleneck contributions to the overall execution time.

6.4 The ArchExplorer Approach

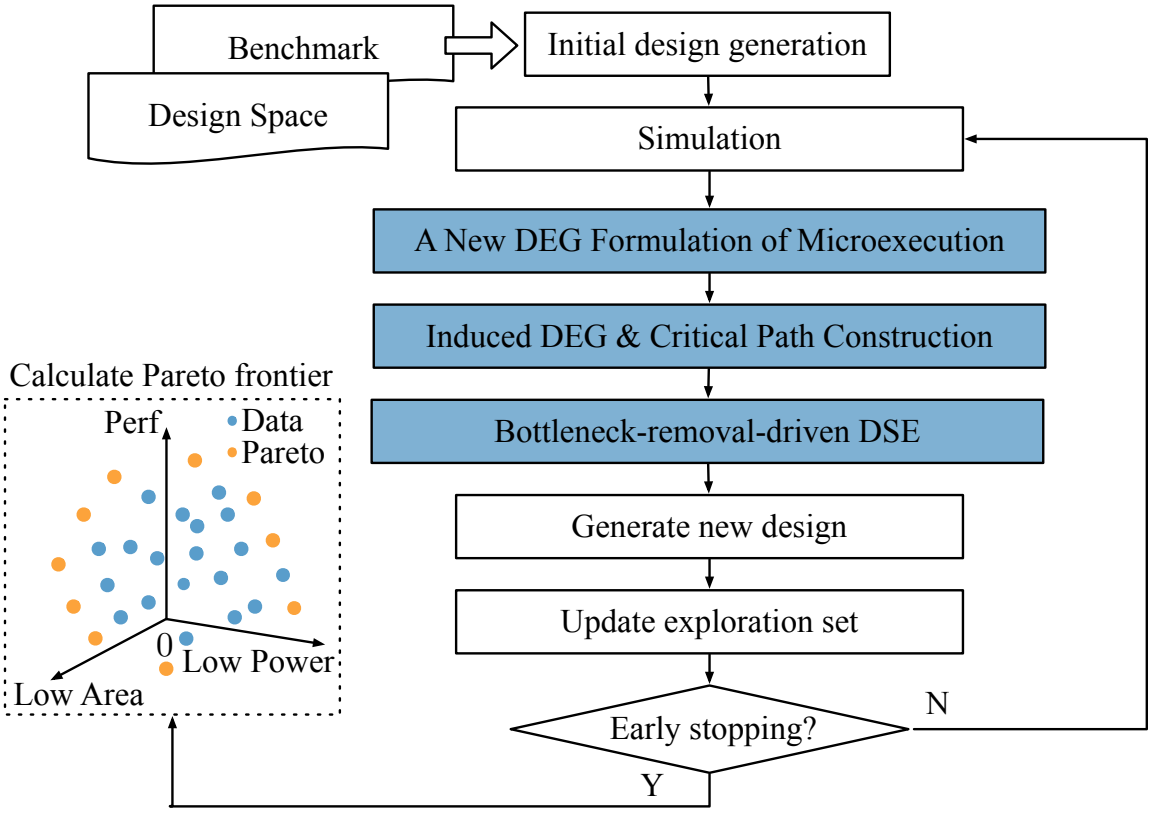


Figure 6.6: An overview of the ArchExplorer approach.

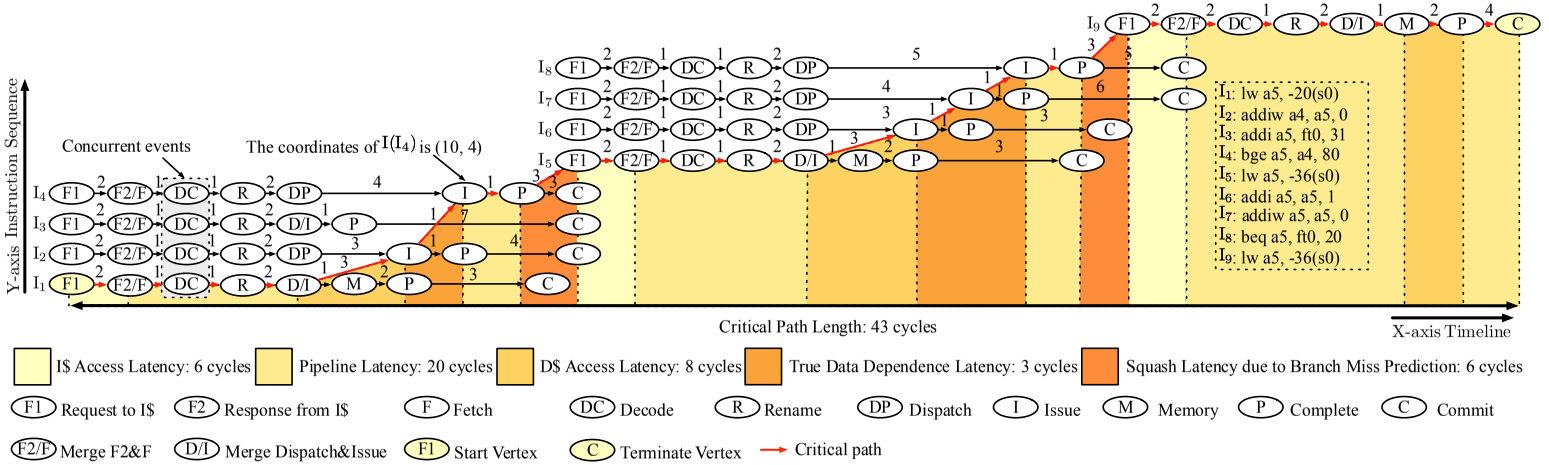


Figure 6.7: An overview of the new DEG formulation of microexecution. The critical path is highlighted in red. The cause of each edge in the critical path is attributed to a particular resource (shaded with colors for visualization purposes).

Following the design principles summarized in Section 6.3, we propose ArchExplorer, with an overview shown in Figure 6.6.

ArchExplorer involves three stages. Given the design space, the initial microarchitecture parameters are sampled or chosen with prior knowledge. In stage 1, we introduce a new DEG formulation (Section 6.4.1). The new formulation removes previous limitations. Based on the new DEG, we propose the induced DEG and apply critical path construction in stage 2 (Section 6.4.2). In stage 3 (Section 6.4.3), we generate a bottleneck analysis report by calculating the resource contribution to the microexecution runtime. We reassign resource according to the report, producing a new design. The promising solutions are explored until the early-stopping criterion is met, and the Pareto frontier is obtained from the explored set.

6.4.1 New DEG Formulation of Microexecution

We propose a new DEG representation of microexecution to mitigate the limitations of the previous formulation. The backbone looks similar to the original proposal [68]. Vertices denote pipeline stages, and edges represent dependencies. However, a fundamental limitation of the former formulation is the lack of events' timing information, causing inaccurate modeling of events like speculative scheduling, resource contention, *etc.* Lacking the timing information also makes the concurrent events difficult to discriminate. We explicitly embed the events' timing information into the formulation and propose new rules to assign edges and weights dynamically. First, we give an overview of the new DEG formulation. Next, we illustrate how the new DEG is dynamically constructed. Last, we manifest how we distinguish between overlapped events.

Figure 6.7 is an outline of the new DEG formulation. Instead of aligning instructions with pipeline stages, as shown in Figure 6.4, we align instructions *w.r.t.* the time. Each vertex is accessed with two-dimensional coordinates (x, y). The X-axis denotes the timeline, and the Y-axis represents the instruction sequence. The instruction-level parallelism is shown by fixing the X coordinate. And the lifetime of an instruction is reported by fixing the Y coordinate. For edge weights, instead of using static numbers to denote, dynamic time intervals between two vertices are used in the new DEG. In the following context, we use $\mathrm{P}(\mathrm{I}_i)$ to represent the instruction I_i's pipeline stage P [2].

We categorize four kinds of dependencies. See Table 6.2. The pipeline dependencies are horizontal edges. Misprediction, hardware resource, and true data dependencies are "skewed" edges, *i.e.*, they denote interactions between instructions.

- **Pipeline dependence:** It characterizes the microarchitecture pipeline implementations. More vertices are added to model architecture parameters. For example, to study the I-cache and fetch buffer size, we incorporate F1 and F2, representing when the front-end sends requests and receives replies from the I-cache. The duration of $\mathrm{F1}(\mathrm{I}_i) \rightarrow \mathrm{F2}(\mathrm{I}_i)$ equals the I-cache access latency, while F describes the moment instructions are copied to the fetch target queue. Misaligned accesses are modeled by $\mathrm{F2}(\mathrm{I}_i) \rightarrow \mathrm{F}(\mathrm{I}_i)$. To aid visualization, we merge F2 and F to formulate F2/F when the events occur simultaneously. This merging is also employed for other vertices.

- **Misprediction dependence:** It is used to model branch or memory address dependence mispredictions [52].

[2] P could be F1, F2, DC, *etc.*, as shown in Figure 6.7. For example, $\mathrm{DC}(\mathrm{I}_3)$ denotes the decode stage of instruction I_3.

- **Hardware resource dependence:** The previous formulation uses the "producer-consumer" model [68] to build usage dependence on resources such as physical registers, ROB, *etc.* For example, $C(I_i) \rightarrow I(I_j)$ can represent ROB dependence, as a new ROB entry is produced at the commit stage of I_i and consumed at the issue stage of I_j. In contrast, we unify dependencies as rename to rename edges $R(I_i) \rightarrow R(I_j)$. This is because the new DEG aligns instructions strictly with time, so a "producer-consumer" edge is always assigned zero delays since newly-released resources can be used immediately. A zero delay edge prevents the critical path from capturing critical resource usage dependence. Mainstream microarchitectures often use one stage (*e.g.*, rename) to check whether required resources are available before dispatching the instruction. If a requisite resource is exhausted, a stall is incurred, otherwise, the instruction is pushed to the issue queue and waits to schedule. The rename to rename edge precisely reflects the resource usage dependence, and the time interval between these two rename stages equals the resource's duty cycles (the resource is busy during the interval).

- **True data dependence:** It characterizes the read-after-write dependence within the issue window.

Another significant difference from the previous formulation is that we construct the new DEG dynamically. That is, the new DEG is built adhering to the actual microexecution. With the time coordinate, we know whether a misprediction event occurs and how many penalties are produced. For example, if a branch instruction I_i is mispredicted, microarchitecture starts squashing and refilling the pipeline with instructions in the correct execution path. Denote the first refilled instruction as I_j.

Table 6.2: The dependence specification

Type	Edge	Description
Pipeline dependence	$F1(i) \rightarrow F2(i)$	Send a request to I\$, and get a response for instruction i.
	$F2(i) \rightarrow F(i)$	I\$ puts the instruction i in the fetch buffer, and the fetch stage performs pre-decode or predictions.
	$F(i) \rightarrow DC(i)$	The fetch stage send instruction i to the decoder.
	$DC(i) \rightarrow R(i)$	The decode stage send μ-ops of instruction i to the rename.
	$R(i) \rightarrow DP(i)$	The rename stage send instruction i to dispatch.
	$DP(i) \rightarrow I(i)$	Schedule instruction i to issue.
	$I(i) \rightarrow P(i)$ $I(i) \rightarrow M(i) \rightarrow P(i)$	Execute instruction i with suitable functional units like ALUs or read/write ports.
	$P(i) \rightarrow C(i)$	Commit instruction i after it is finished execution.
Misprediction dependence	$P(i) \rightarrow F1(i+1)$	Instruction i encounters a branch/memory address dependence misprediction.
Hardware resource dependence	$R(i) \rightarrow R(j)$	Insufficient resources delays instruction j, and j requires those resources that instruction i releases. The edge insertion is according to the scoreboard. The resources include ROB, IQ, LQ, SQ, as well as physical integer and floating-point registers.
	$I(i) \rightarrow I(j)$	Insufficient resources delays instruction j, and j requires those resources that instruction i releases. The resources are functional units, *e.g.*, integer/floating-point ALUs, dividers, *etc.*
True data dependence	$I(i) \rightarrow I(j)$	The true data dependence. The delayed cycles are either due to D\$ access or the execution of functional units.

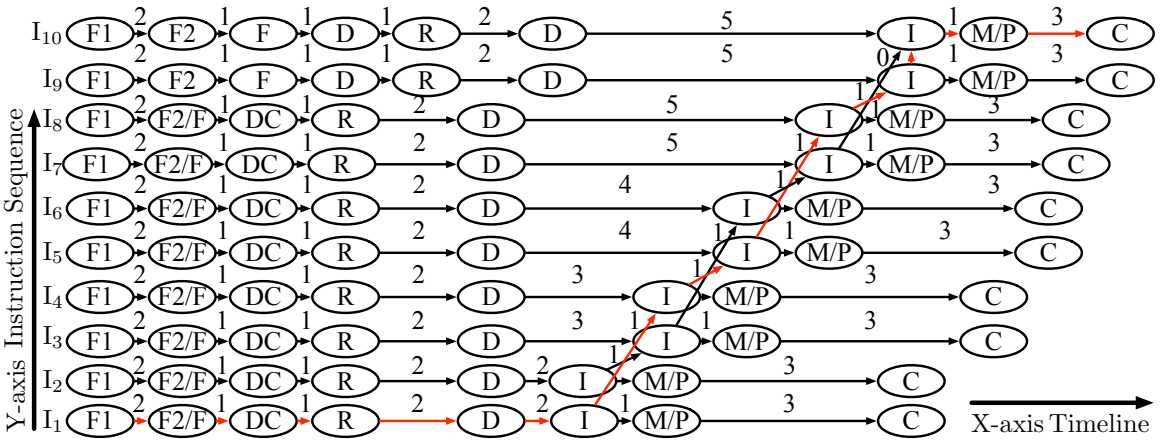


Figure 6.8: The new DEG formulation is applied *w.r.t.* the code snippet as shown in Figure 6.5b. And it identifies the true read/write ports usage dependencies, *i.e.*, $I(I_1) \rightarrow I(I_4)$, $I(I_4) \rightarrow I(I_5)$, $I(I_5) \rightarrow I(I_8)$, and $I(I_8) \rightarrow I(I_9)$.

We assign the edge $P(I_i) \rightarrow F1(I_j)$ once we find the time of $F1(I_j)$ is larger than $P(I_i)$ from the new DEG. The interval between $P(I_i)$ and $F1(I_j)$ is the actual squash latency due to the misprediction.

However, acquiring the pipeline stages' timing information is inadequate to identify the correct edges and weights for resource usage dependencies (such as issue queues and ROB). Although time is useful for identifying stalls, it is not sufficient to determine these edges. Which resources are preemptively occupied by which instructions? To address this issue, we record the correspondence between assigned resources and instructions with a scoreboard. The scoreboard uses resource entry as the granularity and indexes each entry with a unique ID number. We demonstrate how we detect and assign the correct edge with the scoreboard. Consider a microarchitecture that uses the rename stage to check whether required resources are available for each instruction, and the rename stage takes two cycles to finish. If all required resources are idle, the instruction can be dispatched immediately. Otherwise, the instruction

is stalled at the rename stage. We record which resource is responsible for the stall and assign a rename-to-rename edge by comparing the ID number. Specifically, the resource ID number of the dependent instruction should match that of the stalled instruction according to the scoreboard. If an instruction is stalled due to multiple insufficient resources, the rename-to-rename edges are built in turn.

With the new DEG formulation, Figure 6.8 elucidates how we distinguish between concurrent events *w.r.t.* the same code snippet listed in Figure 6.5b. Since $I(I_1)$ and $I(I_2)$ are aligned along the X-axis, no function unit contention exists [3]. Compared to Figure 6.5b, the critical path highlighted in red in Figure 6.8 removes duplicated read/write port dependencies.

6.4.2 Induced DEG & Critical Path Construction

The critical path is a serialization representation of parallel events. It highlights which overlapping events matter for the overall microexecution.

To facilitate the critical path construction, we introduce the *induced DEG* – a *connected* DEG (Section 6.4.1) consisting of horizontal, "skewed", and *virtual* edges. Horizontal edges are pipeline dependencies. "Skewed" edges (non-horizontal) signify non-pipeline dependencies, which could be resource dependencies. The virtual edges are added to make the new DEG connected. Unlike the prior DEG [68], our new DEG formulation removes consecutively connected fetch edges, commit edges, *etc.* This is because these edges are rooted in instruction execution sequences rather than resource dependencies, as they can hinder critical path construction. However, their

[3] $I(I_3)$ and $I(I_4)$, $I(I_5)$ and $I(I_6)$, $I(I_7)$ and $I(I_8)$, *etc.*, are also aligned with time, meaning they are concurrent events.

removal may result in a disconnected graph (no path from F1(I_1) to the last instruction's commit) if instructions do not have non-pipeline dependencies in highly parallel workloads. Thus, we introduce virtual edges in the induced DEG.

Formally, assume two "skewed" edges are annotated with $s_i \rightarrow e_j$ and $s_k \rightarrow e_l$ ($j < k$), where s_i, e_j, s_k, e_l are vertices in the new DEG formulation, *i.e.*, a stage of I_i, I_j, I_k, and I_l, respectively. i, j, k, and l are different instruction sequence numbers. Virtual edges could be $s_i \rightarrow s_k$ (or $e_j \rightarrow s_k$) as long as either of the following two rules is met [4].

Rule 1 **(Connect via time):** s_i is connected to s_k if the time of s_k is the closest to s_i.

Rule 2 **(Connect via instruction sequence):** s_i is connected to s_k if the instruction sequence k is the closest to i.

If multiple stages satisfy at least one of these rules, we connect them to s_k to generate more virtual edges. Virtual edges are not true dependencies but make the DEG connected. The connection of "skewed" edges allows for the exposure of consecutive resource utilization status. We connect "skewed" edges with the closest time and instruction sequence for two reasons. First, since the induced DEG is highly connected due to virtual edges, we can investigate many combinations of resource usage dependencies to formulate the critical path. Second, the closest connections allow us to directly eliminate many sub-optimal solutions, as connections that are not the closest are obviously sub-optimal according to the first design principle. The induced DEG facilitates the critical path densely composed of resource usage dependencies.

A walking example is provided in Figure 6.9 to demonstrate the idea more clearly.

[4]We use $s_i \rightarrow s_k$ as an example. The rules are the same for $e_j \rightarrow s_k$.

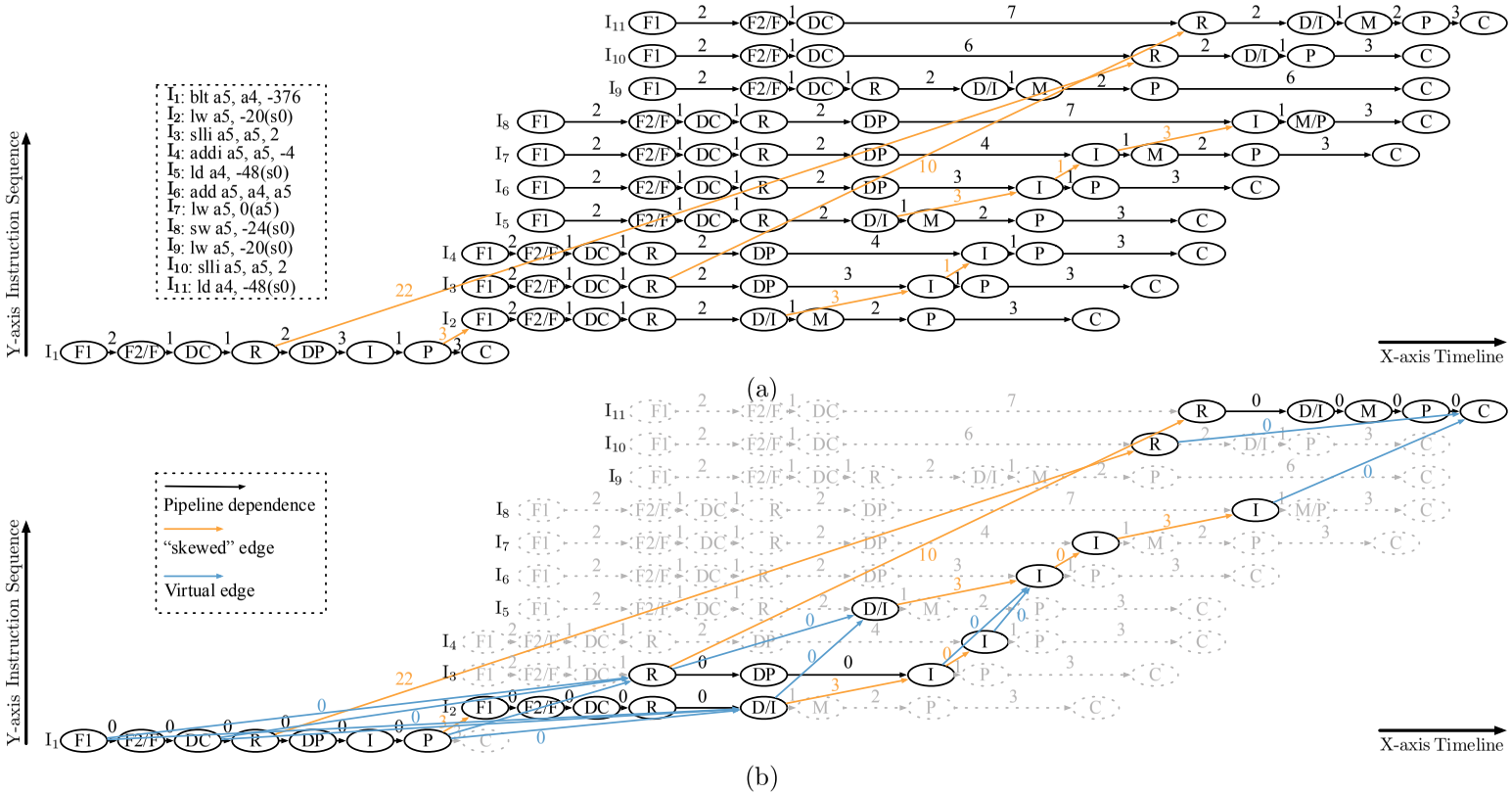


Figure 6.9: (a) An example code snippet and its corresponding new DEG formulation. (b) The overview of induced DEG with edge cost extracted from DEG.

The new DEG is applied to model 11 instructions in a microexecution (Figure 6.9a), with "skewed" edges colored in orange. $R(I_1) \rightarrow R(I_{10})$ and $R(I_3) \rightarrow R(I_{11})$ denote dependencies of integer physical registers. $D/I(I_2) \rightarrow I(I_3)$, $D/I(I_5) \rightarrow I(I_6)$, and $I(I_7) \rightarrow I(I_8)$ are D-cache misses. $I(I_3) \rightarrow I(I_4)$ and $I(I_7) \rightarrow I(I_8)$ are true data dependencies due to register a_5. $P(I_1) \rightarrow F1(I_2)$ represents a branch misprediction. Although I_9 and I_{10} have a true data dependence due to register a_5, there is no stall in actual microexecution because I_9 and I_{10} are not in the same issue window (I_{10} is not dispatched at the time of $D/I(I_9)$). $F1(I_1)$ is not reachable to $C(I_{11})$. Figure 6.9b illustrates the corresponding induced DEG with virtual edges colored in blue. Specifically, we use $R(I_1) \rightarrow R(I_3)$ and $R(I_1) \rightarrow D/I(I_2)$ as two examples to elucidate rules 1 and 2. In Figure 6.9a, a virtual edge $R(I_1) \rightarrow R(I_3)$ is created as $R(I_3)$'s time is the closest to $R(I_1)$ among all "skewed" edges, and I_3 and I_1 are different instructions (rule 1). Similarly, a virtual edge $R(I_1) \rightarrow D/I(I_2)$ is created since the instruction sequence of $D/I(I_2)$ is the closest to that of $R(I_1)$ among all "skewed" edges (rule 2).

Although the induced DEG connects the graph, it cannot answer which concurrent event should be blamed for the microexecution. We propose a dynamic programming-based critical path construction algorithm to decide accordingly. To capture more resource usage dependencies and not miss "important" edges, we define *costs* to edges as follows: (1) All horizontal edges have zero cost. (2) All virtual edges have zero cost. (3) All true data dependencies have zero cost. (4) All "skewed" edges, except true data dependencies, have costs equal to the interval between two vertices. We set horizontal edges with zero cost to capture the true resource usage dependence rather than pipeline stalls. Using the induced DEG with edge costs, we apply the longest

Algorithm 6 Critical Path Construction

Require: G: The induced DEG with the edge cost;
1: node = `topological_sort`(G);
2: Initialize edge cost vector $\boldsymbol{d}$ with all zero;
3: Initialize the path vector $\boldsymbol{p}$ with all zero;
4: **for** $n \leftarrow$ node **do**
5: **if** $\mathcal{N}_G(n) \neq \emptyset$ **then** ▷ $\mathcal{N}_G(n)$ are predecessors of n.
6: $\boldsymbol{d}[n] = \arg \max_{v \in \mathcal{N}_G(n)} d[v] + \text{cost}$; assign $\boldsymbol{p}[n]$ with v;
7: **else**
8: $\boldsymbol{d}[n] = 0$; $\boldsymbol{p}[n] = n$;
9: **end if**
10: **end for**
11: **return** `reverse`($\boldsymbol{p}$);

path search with linear time complexity in Algorithm 6.

In Algorithm 6, line 6 uses dynamic programming to record the temporary longest path to n. The critical path can be obtained by reversing $\boldsymbol{p}$ at line 11. For the example in Figure 6.9b, the critical path is F1(I_1) → F2/F(I_1) → DC(I_1) → R(I_1) → R(I_{10}) → C(I_{11}). From the critical path, we discover that the top bottleneck is the insufficiency of physical integer registers, and the branch misprediction cannot hide it. The length of the critical path is precisely the same as the simulation time.

6.4.3 Bottleneck-removal-driven DSE

We conduct the bottleneck-removal-driven DSE by reassigning resources via eliminating bottlenecks gradually. Scarce resources are increased to mitigate performance bottlenecks, while over-provisioned resources are reduced to balance power and area. Identifying deficient or abundant resources is achieved by computing their contribution to the overall runtime through the constructed critical path. The higher the contribution, the scarcer the resource and the greater the necessity to assign more of

them. Conversely, redundant resource leads to zero contribution, and can be appropriately reduced.

The main idea of computing the resource contribution is to attribute the cause of each edge in the critical path to a particular resource. This attribution is straightforward, as we define the meaning of "skewed" edges in Table 6.2. We do not attribute virtual edges to resources. Figure 6.7 illustrates some attribution highlighted with diverse colors, where each edge is a non-overlapping segment in the microexecution. We introduce the resource contribution in two cases, and describe our resource assignment strategies.

Formally, for a critical path p with length L and containing N (non-overlapping) edges, a resource b's contribution $c(b)$ is computed according to Equation (6.1),

$$c(b) = \sum_{i=1}^{N} l_i \mathbb{1}[p(i) = b]/L, \tag{6.1}$$

where $\mathbb{1}(\cdot)$ is the indicator function, which outputs 1 if its argument is true and 0 otherwise. l_i is the delay (not edge cost as referred to in Section 6.4.2) of the i-th edge $p(i)$, attributed to the resource b. We determine the type of resource according to $c(b)$, *i.e.*, whether the resource is top-ranked deficient or abundant. For multiple workloads evaluations, as shown in Equation (6.2), we do a weighted-average for the contributions of each resource.

$$\bar{c}(b) = \sum_{i=1}^{|B|} w_i \cdot c_i(b), \quad \sum_{i=1}^{|B|} w_i = 1, \tag{6.2}$$

where $|B|$ is the number of workloads, and $c_i(b)$ is computed from Equation (6.1). The symbol $\bar{c}(b)$ is the average resource contribution, where w_i is the i-th weighted coefficient for a specific workload that encodes the designer's preference. Each resource's

contribution is summarized in an output report.

We use $\bar{c}(b)$ to guide the DSE procedure. The reassigned parameter values are decided based on the design space specification. Specifically, we select the next larger candidate value from the specification if we need to increase it, and we decrease them to the next smaller candidate value if they do not have a contribution.

Unlike resources such as ROB entries, branch predictors and caches are special. Assigning more resources may not decrease their contributions, as benchmarks contain hard-to-predict branches or specific data access patterns that are fundamental to the prediction algorithm. Increasing the branch target buffer (BTB), return address stack (RAS), *etc.*, may not improve the branch predictor. Similarly, larger capacities and associativities may not remove caches' contributions on complex access patterns. We argue that performance cannot be effectively enhanced by only reassigning more resources to them. The solution to the problem requires a suitable branch prediction algorithm or a better cache replacement policy. We stop reassigning more resources to branch predictors and caches in the DSE if the PPA improvement is limited.

Figure 6.10 provides an overview of a search path conducted by ArchExplorer. In the second step, the report indicates that the microarchitecture lacks sufficient store queues (SQ), which contribute 38% to the execution time. Thus, in the third step, more SQ is assigned, resulting in an 8% alleviation of the bottleneck. In the fourth step, increasing SQ further mitigates the bottleneck, while decreasing redundant resources saves the area overhead. The DSE uncovers the reasons for PPA improvements stepwise, highlighting potential optimization opportunities for the load-store unit, with assigning more SQ being one optimization. Architects can gain valuable insights into acquiring good solutions gradually. Black-box methods cannot offer the

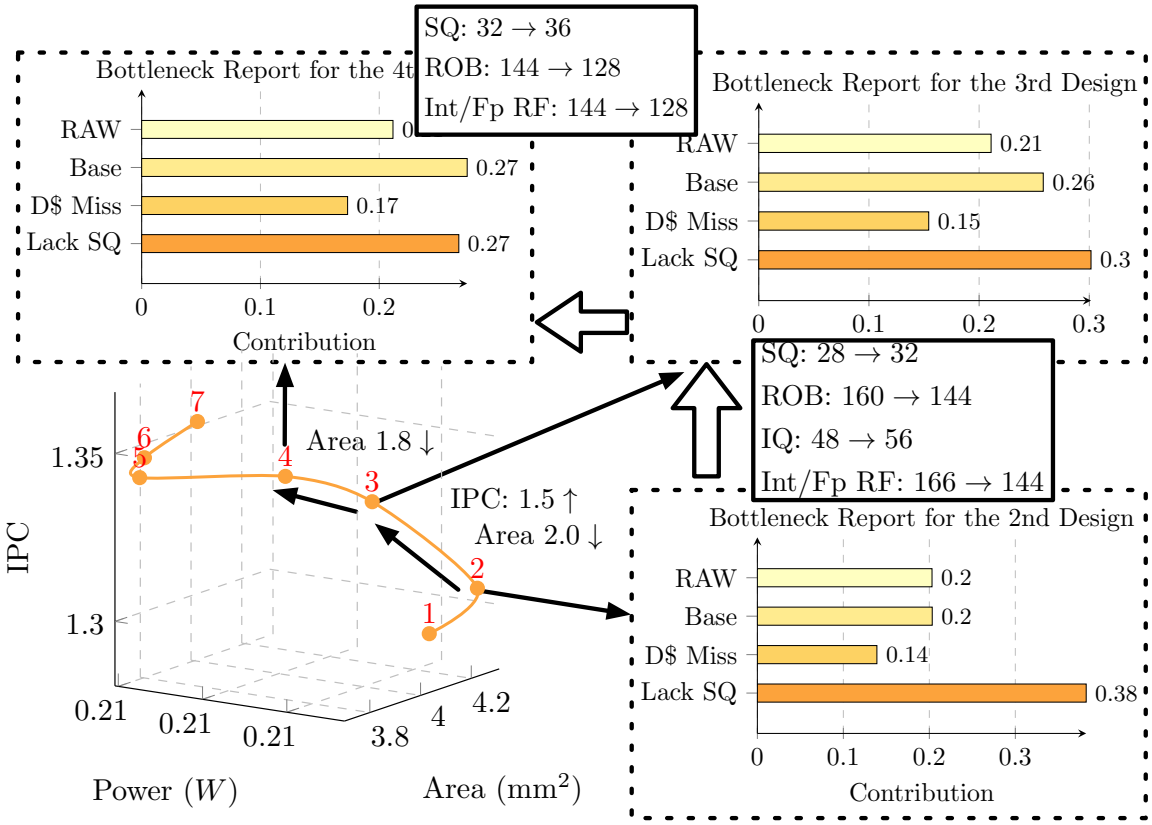


Figure 6.10: An overview of a search path.

same merit.

6.5 Experimental Setup & Evaluation Metrics

In this section, we illustrate our experimental setup and evaluation metrics.

6.5.1 Simulation Environment

We use GEM5 [36, 123], as the timing-accurate simulator and McPAT [119] as the power and area modeling tool. We modify GEM5 to generate dynamic timing information and implement ArchExplorer with C++ and Python. SPEC CPU2006 [3, 83]

(SPEC06) and SPEC CPU2017 [7] (SPEC17) are utilized in benchmark evaluations, as listed in Table 6.3. We use Simpoints [151] of each workload to evaluate. Each Simpoint includes a hundred million instructions, warming up using ten million instructions. Since identifying the resource utilization status does not require the entire workload, we use the first hundred thousand instructions of each Simpoint to calculate the critical path with ArchExploer. After the DSE, the explored Pareto solutions are re-evaluated with the full Simpoints.

We implement ArchRanker [48], AdaBoost [118], and BOOM-Explorer [28] as baselines. ArchRanker [48] leverages a black-box model for ranking. AdaBoost [118] also adopts machine-learning techniques [65, 165, 179]. BOOM-Explorer [28] uses Bayesian optimization [54] with active-learning [209]. The baselines represent recent solutions using advanced machine-learning techniques. We also compare with Calipers [75] since it uses the previous DEG formulation to enhance fast DSE. Although some baselines, like ArchRanker [48] and BOOM-Explorer [28], target different out-of-order (OoO) processors, their methods are transferable.

The design space of an OoO RISC-V processor similar to Alpha 21264 [104] is listed in Table 6.4. A two-level cache hierarchy with a 16GB DRAM main memory is applied. The first level cache is for optimization, while the L2 cache is 8-way associative with 2MB. Diverse components like branch prediction units, functional units, and L1 cache structures, *etc.*, are included. The design space size is more than 8.9×10^{14}. For ArchExplorer, we stop the optimization until we find that the PPA trade-off curve starts to plateau. We generate the initial design randomly. Baseline microarchitectures suggested by architects can also be leveraged in the initialization stage to achieve better results. All weighted coefficients w_i in Equation (6.2) are

Table 6.3: Workloads used for evaluation

Benchmark suite	# of Workloads	Example Workloads
SPEC06	12	bzip2, namd, dealII, h264ref, soplex, povray, ...
SPEC17	14	perlbench_s, gcc_s, cactuBSSN_s, nab_s, ...

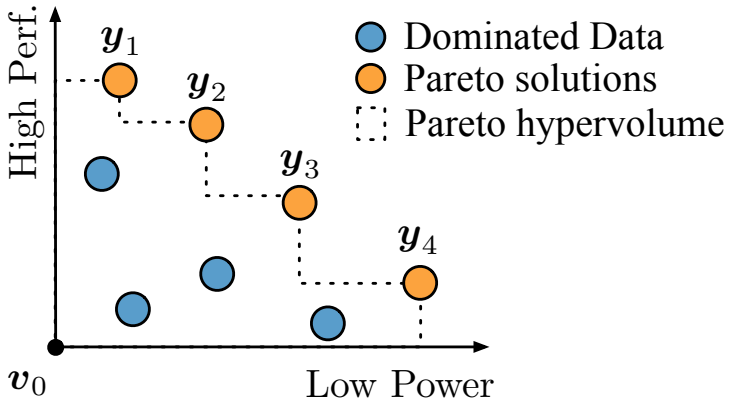


Figure 6.11: The visualization of Pareto hypervolume in Perf-Power space. Pareto hypervolume is the area bounded by $\mathcal{P}(\mathcal{Y}) = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3, \boldsymbol{y}_4\}$ and the reference point $\boldsymbol{v}_0$.

$1/|B|$ since we target average cases and assume no preference for workloads used in experiments.

6.5.2 Evaluation Metrics

Since we target multiple objectives involving higher performance, power efficiency, and area efficiency, we use *Pareto hypervolume* as the evaluation metric to compare different DSE algorithms fairly. Pareto hypervolume is a widely-applied metric to measure how good the explored Pareto frontier is [174]. Denote a Pareto frontier as $\mathcal{P}(\mathcal{Y}) = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_n\} \subseteq$ Y, with $\boldsymbol{y}_i$ representing PPA values of i-th design, and Y is the objective space. A reference point $\boldsymbol{v}_0$ is set and dominated by $\mathcal{P}(\mathcal{Y})$, *i.e.*, $\forall \boldsymbol{y}_i \in \mathcal{P}(\mathcal{Y})$, the PPA values are all better than $\boldsymbol{v}_0$, and we denote $\boldsymbol{y}_i \succeq \boldsymbol{v}_0$.

Table 6.4: Microarchitecture design space specification

Components	Description	Hardware Resource	#
Pipeline width	fetch/decode/rename/dispatch/ issue/writeback commit width	1:8:1 [1]	8
Fetch buffer	fetch buffer size in bytes	16, 32, 64	3
Fetch queue	fetch queue size in μ-ops	8:48:4	11
Local predictor	local predictor size of the Tournament BP	512, 1024, 2048	3
Global/Choice predictor	global predictor size of the Tournament BP	2048, 4096, 8192	3
RAS	return address stack size	16:40:2	13
BTB	branch target buffer size	1024, 2048, 4096	3
ROB	reorder buffer entries	32:256:16	15
Int RF	number of physical integer registers	40:304:8	18
Fp RF	number of physical floating-point registers	40:304:8	18
IQ	number of instruction queue entries	16:80:8	9
LQ	number of load queue entries	20:48:4	8
SQ	number of store queue entries	20:48:4	8
IntALU	number of integer ALUs	3:6:1	4
IntMultDiv	number of integer multipliers and dividers	1, 2	2
FpALU	number of floating-point ALUs	1, 2	2
FpMultDiv	number of floating-point multipliers and dividers	1, 2	2
I$ size	the size of I$ in KB	16, 32, 64	3
I$ assoc.	associative sets of I$	2, 4	2
D$ size	the size of D$ in KB	16, 32, 64	3
D$ assoc.	associative sets of D$	2, 4	2
Total size	8.9649×10^{14}		

[1] The values are start number:end number:stride

Pareto hypervolume is defined as

$$\mathrm{PV}_{\boldsymbol{v}_0}(\mathcal{P}(\mathcal{Y})) = \int_{\mathrm{Y}} \mathbb{1}[\boldsymbol{y} \succeq \boldsymbol{v}_0][1 - \prod_{\boldsymbol{y}_* \in \mathcal{P}(\mathcal{Y})} \mathbb{1}[\boldsymbol{y}_* \not\succeq \boldsymbol{y}]]\mathrm{d}\boldsymbol{y}, \tag{6.3}$$

where the integral sums the space bounded from $\boldsymbol{v}_0$ to $\mathcal{P}(\mathcal{Y})$ [174], as shown in Figure 6.11. We also include the number of simulations spent as another evaluation metric. The higher the Pareto hypervolume and the fewer the simulations, the better the DSE algorithm. We measure the PPA trade-off as $\mathrm{Perf}^2/(\mathrm{Power} \times \mathrm{Area})$.

6.6 Results

In this section, we present our results with ArchExplorer.

6.6.1 Comparison w. DSE Methodologies

Figure 6.12 visualizes the Pareto hypervolume curves in terms of simulations for each algorithm. The average PPA values among workloads are used in Pareto hypervolume computing. The Pareto hypervolume is non-decreasing since more Pareto designs are explored as simulations continue. It is worth noting that ArchExplorer achieves higher Pareto hypervolume very early, and dominates other methods at different simulation budgets. Two cases are selected for comparison to study the improved benefits. First, how many simulations [5] are needed when achieving a target Pareto hypervolume? Second, how much Pareto hypervolume can be attained when all methods use the same simulation budgets? We choose $y = 15.80$ and $x = 3000$ for SPEC06 and

[5]Compared to Calipers, the induced DEG has an average of 39.59% more vertices and 51.72% fewer edges with SPEC17 Simpoints. In our experiments, the longest path evaluation in ArchExplorer incurs 2.24% of the simulation runtime, which can be negligible.

$y = 15.60$ and $x = 2400$ for SPEC17 as two cases, as shown in Figure 6.12. We chose these two cases for SPEC06 and SPEC17 because, at this moment, the performance curves tend to converge. The corresponding results are shown in Table 6.5. In SPEC06, compared to ArchRanker [48], AdaBoost [118], and BOOM-Explorer [28], ArchExplorer users 14.47% more, 24.56% fewer, and 74.12% fewer simulations when $y = 15.80$, respectively. For $x = 3000$, the gained Pareto hypervolume of ArchExplorer surpasses BOOM-Explorer [28], AdaBoost [118], and ArchRanker [48] by 1.58%, 4.20%, and 3.32%, respectively. In SPEC17, ArchExplorer can save 74.63% of simulation budgets at most and achieve 6.80% higher Pareto hypervolume. Particularly, the relatively modest increases in Pareto hypervolume translate into significant performance improvements. For example, when the simulation budget equals 480 in SPEC17, the Pareto hypervolume of ArchExplorer is improved with an average of 5.40% than all AdaBoost [118]. However, the performance of Pareto designs is higher up to 16.20%. The results demonstrate that ArchExplorer can achieve comparable solution qualities by removing large simulation budgets. ArchRanker [48] compares pairs of designs to determine which is better and conducts the constrained DSE with a binary search. Due to the complicated design space introduced in Section 1.2, the trained ranking model is hard to give accurate rankings when confronting a large design space.

To figure out how ArchExplorer performs better than black-box methods, we plot the Pareto frontiers of all methods with 3600 simulations in SPEC06, as shown in Figure 6.13. In IPC-1/Power space, the Pareto frontiers of each method are relatively close. However, in IPC-1/Area and Area/Power space, ArchExplorer's Pareto frontier dominates other methods in several regions, particularly close to the origin of

Table 6.5: Comparison under two cases.

Methods	SPEC CPU2006				SPEC CPU2017			
	Pareto hypervolume at $y = 15.80$		# of Simulations at $x = 3000$		Pareto hypervolume at $y = 15.60$		# of Simulations at $x = 2400$	
	# of Simulations	Ratio	Pareto hypervolume	Ratio	# of Simulations	Ratio	Pareto hypervolume	Ratio
ArchRanker [48]	2736	1	15.9185	1	1296	1	16.4542	1
AdaBoost [118]	3132	1.1447	15.6785	0.9849	2208	0.7037	15.9359	0.9685
BOOM-Explorer [28]	2064	0.7544	16.0854	0.0105	1120	0.8642	16.7416	1.0175
ArchExplorer	708	0.2588	16.3473	1.0269	560	0.4321	17.0198	1.0344

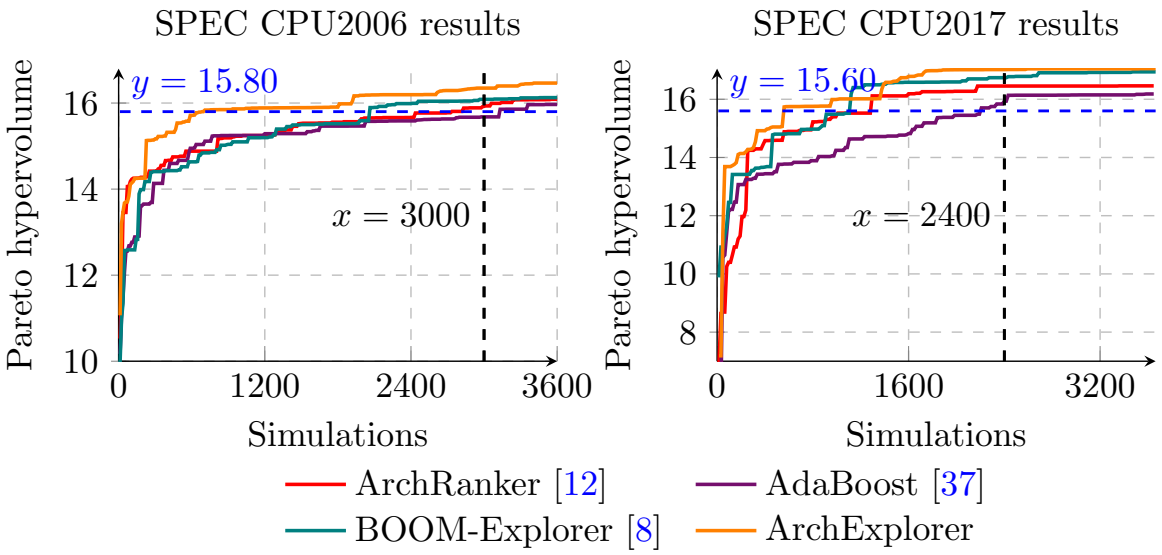
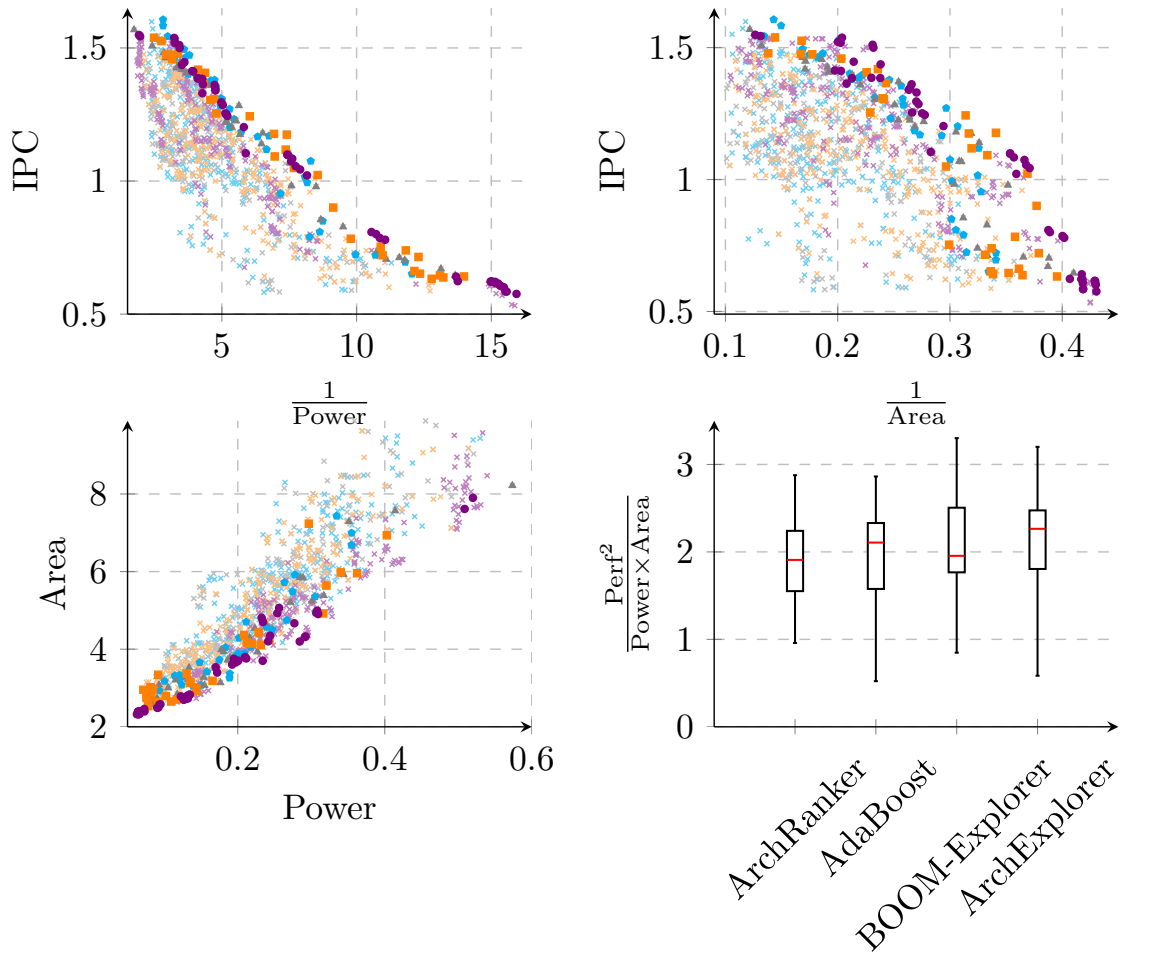


Figure 6.12: The visualization of Pareto hypervolume curves in terms of the number of simulations.

Area/Power space. The visualization suggests that ArchExplorer outperforms other methods not by exploring more higher-performance microarchitectures but higher power and area efficiency designs. Figure 6.13 also shows the PPA trade-off distributions of Pareto designs for each method. ArchExplorer's Pareto designs achieve an average of 2.26 in the trade-off, surpassing BOOM-Explorer [28], AdaBoost [118], and ArchRanker [48] by 15.81%, 7.47%, and 18.63%, respectively. These results demonstrate that we can achieve a better PPA trade-off by assigning and removing indispensable hardware resources based on their performance contribution.

Why can ArchExplorer surpass baselines? ArchExplorer can surpass DSE methods with black-box models because previous methods [48, 118, 28] rely on AdaBoost.RT [118], Gaussian process [28], *etc.*, to learn relationships between microarchitecture features and PPA values. These models can be trained before or during DSE, but they are purely algorithm-driven and do not consider microarchitecture analysis in



- ArchRanker's [12] Explorations
- AdaBoost's [37] Explorations
- BOOM-Explorer's [8] Explorations
- ArchExplorer's Explorations
- ArchRanker's [12] Pareto Frontier
- AdaBoost's [37] Pareto Frontier
- BOOM-Explorer's [8] Pareto Frontier
- ArchExplorer' Pareto Frontier

Figure 6.13: The visualization of Pareto frontiers and the distributions of PPA trade-offs for all methods.

depth. As a result, more simulations are required in the DSE. In contrast, ArchExplorer applies domain knowledge in DSE directly by identifying bottlenecks and adjusting hardware resources to eliminate them, resulting in a better PPA balance immediately. In summary, ArchExplorer's explainable DSE process can provide insights for architects that may not be available through an AI model.

6.6.2 Comparison w. Best Balanced Designs

To study how high-performance design balances PPA well, we rank the designs explored by each method with $\text{Perf}^2/(\text{Power} \times \text{Area})$ and select the ones with the highest performance for comparison. Figure 6.14 lists the results for SPEC06 and SPEC17. Table 6.6 lists key parameters of each Pareto design. For SPEC06, on average, ArchExplorer's best balanced design achieves 1.53%, 16.65%, and 19.81% higher performance than ArchRanker [48], AdaBoost [118], and BOOM-Explorer [28]. It also saves power by 2.15% compared to AdaBoost's [118]. ArchExplorer improves the area by an average of 11.79%. Namely, ArchExplorer's Pareto design is better than other methods by an average of 56.05% and, at most, 64.29% in the PPA trade-off. In SPEC17, ArchExplorer's solution achieves an average of 9.46% higher performance. While the solution sacrifices 5.54% more power compared to baselines, it attains a 20.07% smaller area and 49.53% higher PPA trade-off. Although the designs in Table 6.6 look similar, big differences in the PPA trade-off are observed. The results illustrate that better solutions are achieved by balancing resources. For example, the ROB should not be allocated many entries compared to ArchRanker [48]. Otherwise, it can degrade the overall performance by long squashing due to misprediction events. Redundant resources like floating-point physical registers can worsen power

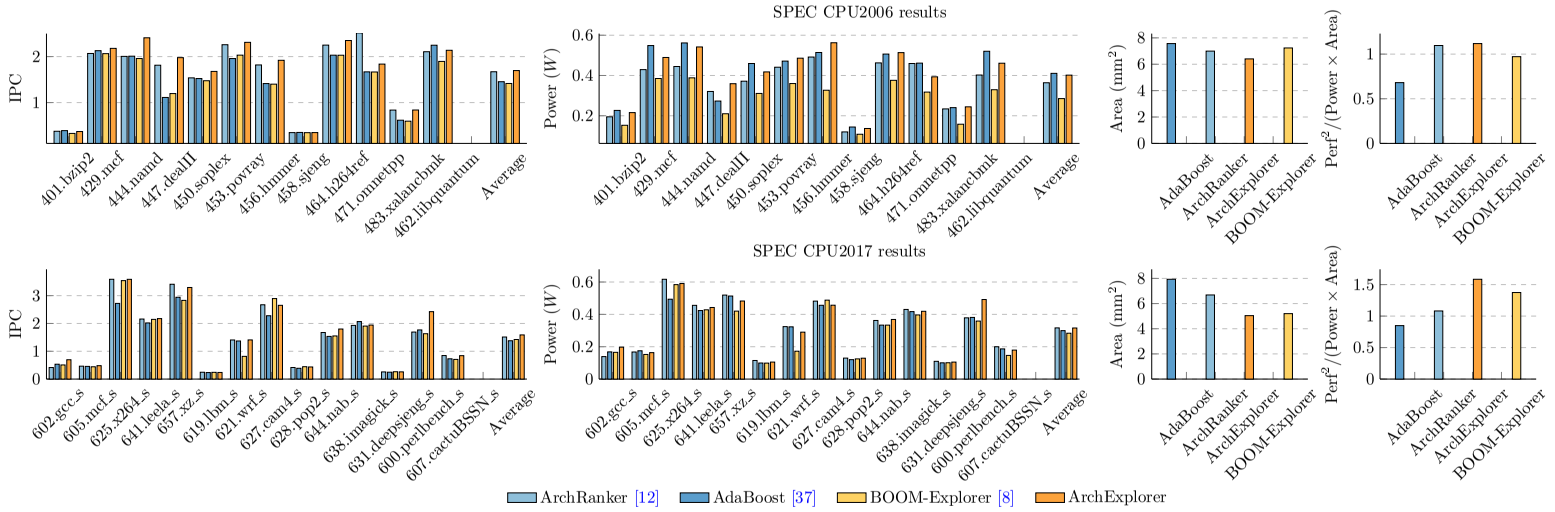


Figure 6.14: Comparisons between the Pareto designs in performance and power.

dissipation compared to AdaBoost [118].

6.6.3 Comparison w. Calipers

Calipers [75] represent the latest critical path analysis method to enable fast DSE. By scanning the design space, Calipers [75] assists in exploring high-performance microarchitectures. We use a different experimental setup to demonstrate the superiority in identifying the type of resources with the new DEG formulation over Calipers [75]. Calipers [75] only targets performance and neglects the consideration for power and area optimization. And it does not provide how to search except for applying the previous DEG formulation in performance modeling with microexecution. On the other hand, Calipers [75] can model more accurately with information like branch prediction results and cache access penalties from the simulation. Hence, we extract very similar 1296 designs from Table 6.4 and use Calipers [75] to sweep the design space and retrieve the highest-performance solution. The reason are two folds. First, it helps Calipers to obtain what it considers the global optimal solution for a fair comparison. Second, very similar designs can expose modeling differences between two DEG formulations in deep. We apply ArchExplorer to the same sub-design space and compare the solutions in performance and power. Different from Calipers [75], ArchExplorer does not scan the entire design space.

Results are shown in Figure 6.15. The solution found by ArchExplorer outperforms Caliper's [75] by 2.11% in performance on average in SPEC06. And ArchExplorer' solution achieves 4.36% lower power and 2.38% lower area. In SPEC17, we receive a 1.88% higher performance compared to Calipers [75]. The previous DEG formulation incorrectly model the microexecution, leading to sub-optimal solutions.

Table 6.6: Key parameters of Pareto designs

Components	Method			
	AdaBoost [118]	ArchRanker [48]	BOOM-Explorer [28]	ArchExplorer
Pipeline width	8	8	8	8
ROB/IQ	176/56	192/72	112/44	176/72
Int/Fp RF	192/208	256/96	240/192	240/80
LQ/SQ	44/48	48/44	24/40	48/48
IntALU IntMultDiv	5/1	4/2	4/2	4/2
FpALU FpMultDiv	2/1	2/1	2/2	1/1
I$	4-way 64K	2-way 64K	2-way 64K	4-way 64K
D$	4-way 64K	2-way 64K	2-way 32K	4-way 64K

Calipers cannot identify which microarchitecture achieves higher performance with a very small ROB resource difference. Conversely, ArchExplorer adopts the new DEG formulation and attains better results by identifying the deficient resources more accurately. Notably, these improved performance benefits are gained by only using 48 simulations in ArchExplorer for SPEC06 and SPEC17, respectively. In summary, due to the more accurate and practical new DEG formulation, ArchExplorer performs better than Calipers [75].

6.7 Discussions

The new DEG formulation can set up many research opportunities for microarchitecture research.

Combine with machine learning (ML): The new DEG formulation provides richer features for recent powerful deep learning models such as graph neural networks [164, 198]. Performance modeling is possible by combining features extracted from vertices and edges. Combining ML techniques and the new DEG formulation can

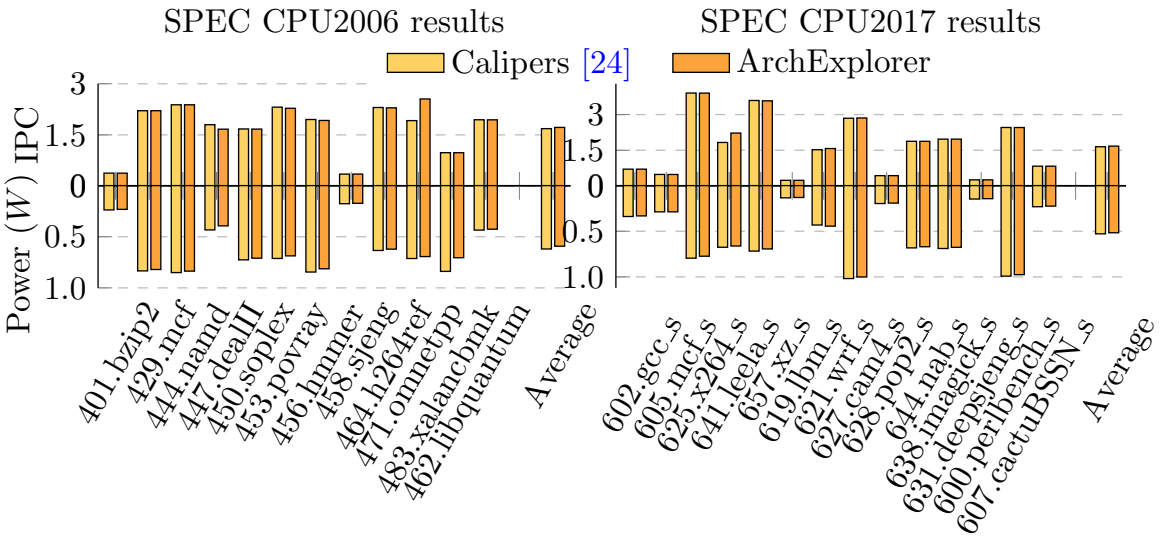


Figure 6.15: Comparisons w. Calipers [75].

also improve the DSE procedure.

Instruction scheduling: The instruction-level parallelism can be modeled by projecting "skewed" edges to the Y-axis. The projection length equals the degree of parallelism. The information helps design new criticality-driven instruction scheduling algorithms for new emerging workloads.

Multi-core formulation: Commodity microprocessors continue to scale in a number of cores (96 cores in AMD EPYC Genoa [136], 64 Intel Raptor Cove in Emerald Rapids [9]), given the technology scaling. Hence, the DEG formulation for multi-core deserves exploitation. It is beneficial to find bottlenecks for a multi-core system.

6.8 Additional Related Work

Critical Path Analysis. Fields *et al.* are the first to propose the critical path analysis with DEG to unveil bottlenecks in microexecution [68]. This approach has

been broadly used [67, 69, 137, 156, 116, 143, 144, 186, 75]. Behnam *et al.* adopt it to remove bottlenecks in uniprocessors [156]. Nowatzki *et al.* extend Fields' model and propose the transformable dependence graph (TDG) [143]. The TDG is further leveraged in modeling heterogeneous accelerators [144]. However, assigning dependencies and weights without adhering to actual microexecution in TDG leads to inaccurate resource contention modeling, as also mentioned in their work [144]. Lee *et al.* [116] and Calipers [75] enable fast DSE for microarchitecture parameters also based on Fields' model. Compared to the prior works [116, 75], ArchExplorer points out the source of error for previous DEG formulations and proposes the design principles and implementations accordingly. The new DEG formulation eliminates the limitations of Fields' model.

Microarchitecture Design Space Exploration. Many works [133, 14, 39, 91, 115, 101, 59, 146, 116, 48, 118, 28, 75] proposed solutions to DSE for microarchitecture parameters. They either leverage massive access to expert knowledge [100, 101, 63] or require high computing resources to train a black-box model [48, 118, 28] to conduct the DSE. Compared to Karkhanis and Smith [101], ArchExplorer circumvents the experts' efforts by adopting an automated bottleneck analysis. Compared to prior black-box methodologies [91, 115, 48, 118, 28], ArchExplorer conducts DSE by uncovering the bottlenecks rather than relying on black-box models.

6.9 Summary

In this chapter, we discuss the problem of DSE for microarchitecture parameters. To alleviate massive domain knowledge requirements for mechanistic models and to re-

duce high computing demands for black-box methods, we propose ArchExplorer, an automated bottleneck analysis-driven DSE approach implementing two design principles. Specifically, we propose a new DEG formulation for microexecution. An optimal critical path construction algorithm is also proposed to capture hardware resource utilization status. Several resource reassignment strategies are leveraged to remove bottlenecks and trade-off PPA values. ArchExplorer achieves an average of 6.80% Pareto hypervolume improvement and reduces more than 74.63% simulation overheads compared to previous state-of-the-art solutions. The Pareto solutions outperform previous approaches in performance by an average of 29.38% and PPA trade-off improvement by at most 64.29%.

6.10 Artifact Appendix

6.10.1 Abstract

The artifact contains ArchExplorer's codes and its setup and running descriptions. We provide instructions and click-to-run scripts for reproducing the main results in this chapter. Specifically, we reproduce the results of Figure 6.2, Figure 6.3, Figure 6.12, Figure 6.13, Figure 6.14, and Figure 6.15.

6.10.2 Artifact check-list (meta-information)

- **Code base:** The code base of the artifact involves the entire implementation of ArchExplorer. For example, the artifact contains the new DEG modeling, critical path construction, bottleneck-removal-driven DSE, and automated simulator parallel compilation and simulations.

- **Run-time environment:** We provided a Docker run-time environment for users, which removes large burdens in setting up the environment.

- **Output:** The outputs of the artifact are figures in `PDF` format to reproduce the main results in this chapter.

- **Compilation, simulation & modeling:** The artifact includes the automated compilation of GEM5 simulators [36, 123] and the incorporation of power and area modeling using McPAT [119].

- **Benchmarking scripts:** We provide click-to-run bench-marking scripts to evaluate ArchExplorer's performance. Regarding users' different time budgets to run the artifact, we provide three modes to reproduce results. The script `exp_full_mode.sh` can reproduce the "full" results for this chapter (we term this reproduction as the full mode), but it costs high runtime and machine resources. The script `exp_partial_mode.sh` demonstrates experimental results with partial benchmarks in a relatively more efficient way compared to the full mode. So, we denote it as the partial mode. The script `exp_demo_mode.sh` is also provided for users to fast experience the functions provided by our artifacts (demo mode). We leverage RISC-V bare model benchmarks [1] in the demo mode.

- **Documents:** We provide detailed `README.md` documents to guide the Docker environment setup, evaluation steps, and results demonstrations.

- **How much disk space required (approximately)?:** The disk space should be larger than 2 TB.

- **How much time is needed to prepare workflow (approximately)?:** It takes several minutes to prepare the workflow if users have SPEC benchmarks [3, 7] and corresponding Simpoint checkpoints [151]. The preparation includes the SPEC benchmarks set up and configurations of some benchmarks. Due to the SPEC license restrictions, we are unable to publicly distribute the SPEC benchmarks. For SPEC CPU2017, users are also required to prepare Simpoints checkpoints. The necessary directory trees for both benchmarks are listed in the `README.md` file, along with detailed instructions on how to map users' provided benchmarks to the pulled Docker environment that we have prepared. In the Docker environment, certain benchmarks need to be reconfigured to support the simulations. Otherwise, the simulation would get stuck and restrict to reproduction of results. The methods to reconfigure these benchmarks are also listed in `README.md`. However, we provide scripts that allow the reproduction of results without reconfiguration of certain benchmarks, *i.e.*, partial mode. As a result, the expected outcomes and results may differ between the partial mode and full mode. However, the generated figures using the partial mode do not affect the conclusions claimed in this chapter. Furthermore, results could be different if distinct Simpoints checkpoints are leveraged in the experiments. The Simpoints checkpoints settings are also mentioned in `README.md`.

- **How much time is needed to complete experiments (approximately)?:** For high-end Linux machines, such as 80 cores of Intel(R) Xeon(R) CPU E7-4820 v3 @ 1.90GHz with 1 TB of main memory, the full mode takes approximately 15 days. The partial mode costs about 9 days. The demo mode consumes around 5 hours, but it only reproduces Figure 6.2 and Figure 6.3.

6.10.3 Description

How to access

The artifact is archived in Zenodo [6].

Hardware dependencies

The artifact requires a high-end Linux machine with at least 2 TB of disk space. The main memory should be at least 64 GB to support parallel compilation and simulations.

For reference, we list our system configurations here:

- OS: Ubuntu 18.04
- CPU: Intel Xeon Platinum 8163 CPU @ 2.50GHz (96 cores)
- DRAM: 400 GB
- 9.6 TB

Software dependencies

The software dependencies are resolved by our provided Docker environment. Users are required to support Docker commands in the machines if using our provided Docker environment.

6.10.4 Installation

The installation requires two steps, as listed below.

[6] https://doi.org/10.5281/zenodo.8353864

SPEC CPU benchmarks installation

This step may take some time since it is necessary to prepare for the workflow. Due to the SPEC CPU benchmarks license, we cannot release benchmarks to the public. Users need to prepare SPEC CPU benchmarks and install them in a manner *w.r.t.* our accepted directories trees. More detailed information about Simpoints settings and accepted directories trees are instructed in the `README.md` file. Assume the SPEC CPU benchmarks have been installed in `/path/to/benchmarks`.

Code installation

Download and decompress the artifact, and pull and install the Docker image.

```
$ unzip arch-explorer.zip
$ cd arch-explorer-main
$ docker run -it -d \
--name micro23 \
--hostname micro23 \
--network=host \
-v $(pwd):/root/workspace/arch-explorer \
-v /path/to/benchmarks: \
  /root/workspace/benchmarks \
-w /root/workspace \
docker.io/troore/arch-explorer:2.0
$ docker exec -it micro23 /bin/bash
```

Since users have already set the directories mapping strategy by executing the

Docker "run" command, codes should be placed in `/root/workspace/arch-explorer`, and SPEC CPU benchmarks should be placed in `/root/workspace/benchmarks` when users enter the Docker environment using the "exec" command.

6.10.5 Experiment workflow

Basic Setup

After users enter the docker image using the `exec` command, execute the following commands to build configurable infrastructures that ArchExplorer depends on.

```
$ cd /path/to/arch-explorer
$ ./tools/settings.sh
$ export PYTHONPATH=`pwd`
```

Run experiments with three modes

There are three possible factors that will probably prevent users reproducing our results:

- benchmark availability
- hard-coded workload absolute paths
- long runtime

We assume that SPEC CPU benchmarks are available for users. However, if it is not true, we provide a demo mode for users successfully run through some of

our experiments. Under the demo mode, we use open source RISC-V bare model benchmarks [1] rather than SPEC CPU benchmarks.

Moreover, some SPEC benchmarks MUST contain hard-coded workloads absolute paths, *e.g.*, 464.h264ref from SPEC CPU2006 benchmark. These hard-coded absolute paths would prevent users reproducing results in a push-button way. Human efforts are required to configure the hard-coded absolute paths before results related to these benchmarks are expected.

Last, the whole process for reproducing all results in this chapter will take approximately 15 days on our testing systems (for high-end Linux server machines, *e.g.*, 96 cores of Intel Xeon Platinum 8163 CPU @ 2.50GHz with 400 GB main memory).

Therefore, if users have SPEC CPU benchmarks, and do not want to manually resolve the hard-coded absolute paths, or cannot accept the long runtime, we set the partial mode in which only the benchmarks without hard-coded absolute paths will be run.

We also provide the full mode which can reproduce the experimental results with all utilized benchmarks as in this chapter.

Which mode to choose depends on your time budget.

Commands for three modes

In the Docker environment, enter the `artifacts` directory.

```
1 $ cd arch-explorer/artifacts/
```

The steps for running experiments with three modes are shown below.

```
$ ./exp_demo_mode.sh # demo mode: about 5 \
  # hours, but only reproduce \
  # Figure 2 and Figure 3.
$ ./exp_partial_mode.sh # partial mode: about \
  # 9 days, and can reproduce all figures.
$ ./exp_full_mode.sh # full mode: about \
  # 15 days, and can reproduce all figures.
```

6.10.6 Evaluation and expected results

Results are figures in `PDF` format. The demo and partial mode have some distinctions from the figures in the chapter due to the different workloads utilized in the experiments. However, the generated figures do not affect the conclusions claimed in this chapter. Results are stored in respective sub-directories in `artifacts`, and the paths of these results are demonstrated in the `README.md` file.

6.10.7 Notes

Benchmark reconfiguration for full mode

Before running in full mode, users are required to reconfigure SPEC CPU benchmarks. Otherwise, those benchmarks would fail in simulations and prevent the entire DSE process from continuing to run. These benchmarks are 464.h264ref, 600.perlbench_s, 623.xalancbmk_s, 625.x264_s, and 638.imagick_s, where 464.h264ref is from SPEC CPU2006 benchmarks, and others are from SPEC CPU2017 benchmarks.

For the detailed steps of the reconfiguration, please refer it to in the `README.md`.

About `README.md`

The `README.md` documents of the artifact provide additional information on the illustration of code organizations and detailed steps regarding running experiments.

Chapter 7

Conclusion and Future Work

In this thesis, we have presented a series of research outcomes to tackle microprocessor microarchitecture design space exploration. In this chapter, we summarize the proposed methodologies and then discuss the future work.

7.1 Summary

During years of recent research, we delve into the problem stepwise, proposing methodologies from black-box to white-box, from data-driven to interpretable, and from statistical techniques to formal analysis. [28, 29, 120, 31, 27]

- In Chapter 3, we present BOOM-Explorer, using a customized Bayesian optimization flow for the DSE of the RISC-V BOOM microprocessor. With a dedicated initialization algorithm (MicroAL), a powerful surrogate model (DKL-GP), a negatively correlated multi-objective exploration (EIPV), and a diversity-guided enhancement, we make the DSE of microprocessor microar-

chitectures at the RTL level for the first time. Experiments using commercial electronic design automation (EDA) tools at 7-nm technology demonstrate that BOOM-Explorer can outperform previous representative arts by an average of 18.75% higher Pareto hypervolume, 34.57% less average distance to reference set (ADRS), and 65.38% less overall running time (ORT).

- In Chapter 4, we propose a reinforcement learning pathway, pointing out the mathematical limitations of previous data-driven methodologies, *e.g.*, using the Gaussian process. The formulated Markov decision process is tightly coupled with the microarchitecture scaling graph, and a single agent trained using envelope learning can explore different designs with diverse PPA preferences. The solution achieves an average PPA trade-off improvement of 16.03% compared to previous works, with 4.07× higher efficiency.

- In Chapter 5, we provide a microarchitecture DSE open benchmarking platform, which has been used in the CAD Contest of ICCAD in 2022. We release the benchmarking tools, associated data set, and online team ranking platform, expecting researchers worldwide to propose better algorithms than the baseline.

- In Chapter 6, we perform massive simulation analysis for microexecutions with different microarchitectures and workloads. We demonstrate the error sources of previous dynamic event-dependence graph formulation in the critical path analysis. We summarize the learned lessons from error sources and propose a new graph representation of microexecutions based on two design principles for an explainable DSE process. ArchExplorer, a DSE algorithm via bottleneck analysis, can search for Pareto-optimal solutions and elucidate how to find them

stepwise. Experiments show that ArchExplorer can find better Pareto PPA microarchitectures with an average of 6.80% higher Pareto hypervolume using at most 74.63% fewer simulations than state-of-the-art approaches.

It is noted that all research outcomes mentioned above are open-sourced for the community. We help researchers worldwide to reproduce our published experimental results and expect innovations and better achievements than our methodologies. The links to the codes are shown below.

- BOOM-Explorer: https://github.com/baichen318/boom-explorer-public.
- Reinforcement learning pathway: https://github.com/baichen318/rl-explorer.
- Microarchitecture DSE open benchmarking platform: https://github.com/iccad-contest.
- ArchExplorer: https://github.com/baichen318/arch-explorer.

7.2 Future Work

Although we have presented methodologies proposed in these years of research, many problems remain to be answered, and we leave these problems as future work, as summarized below.

First, we have provided design space exploration at the microarchitectural level but omit characteristics of circuit implementations. A design space exploration that considers architecture and circuit implementation for design technology co-optimization (DTCO) remains to be discussed. For example, the interaction between microarchitectural parameters and the robustness of voltage change or dynamic

voltage and frequency scaling (DVFS) capability for different workloads is unknown. What kind of microarchitecture parameters are sensitive when we dynamically change the frequency or voltage for each functional block, *e.g.*, a wide issue width to compensate for the performance loss if a base clock speed is enabled for handling light workloads [201, 202]? The research direction requires domain knowledge combining computer architecture, electronic design automation, and microelectronics.

Second, many domain-specific accelerators have emerged except for microprocessors, especially machine learning accelerators [58, 50, 51, 82, 105, 71, 193, 97, 176, 72, 73, 15, 96]. How to design an appropriate DSE algorithm for domain-specific accelerators is a hot research topic and valuable future work [212]. The PPA models for deep learning accelerators have been proposed, and many DSE algorithms have been published based on these tools [148, 113]. Similarly, the DSE algorithms can be categorized as white-box and black-box methodologies [87, 99, 110]. The search objectives can be PPA values or other metrics related to economic costs, such as total cost of ownership (TCO) [212]. Different from the DSE of microprocessor microarchitecture, the DSE algorithms for deep learning accelerators can focus on the hardware and software levels due to the characteristics of domain-specific applications [203, 12].

Third, design space exploration is only one small problem in computer architecture research. With the fast-changing nature of our application, a higher-level problem is how to build a better computer architecture or microarchitecture to execute newly-emerged applications efficiently due to the limitations of conventional microprocessors [17]. The key point is to analyze the characteristics of applications and extract more parallelism, including instruction level, memory level, *etc.*. Regarding emerging massive uncertainty in AI workloads nowadays, dataflow architec-

ture, which contrasts with the traditional von Neumann architecture, may be revived [175, 191, 142, 32]. A fundamental distinction from traditional control-flow-based microprocessors is the execution model. Namely, in dataflow architecture, the executability and execution of instructions are solely determined based on the availability of input operands to the instructions [55, 192, 88]. Many dataflow computers have been proposed in history [79, 149, 89, 177, 147, 77, 21, 140, 138, 160]. Nevertheless, these dataflow computers failed in the commercialization. One significant factor is the high difficulty of developing the code generation framework for dataflow architecture [57, 139, 56]. Nowadays, AI algorithm innovation promotes new applications and rekindles the hope for dataflow architecture. High investment is made, and new prototypes emerge [161, 90, 74]. We also propose Klotski, the latest research prototype of dataflow architecture accelerators for deep neural network applications [30]. Future work could focus on code generation for dataflow computers and the application of dataflow computers to new workloads like robotics. Another promising research direction could focus on three-dimensional computer architecture and implementation [122]. Although the research topic has been discussed for many years [37], we have witnessed the emergence of prototypes and commercial products [199, 76]. For example, TSMC has announced 3D Fabric technologies to build 3D silicon stacking chips [10]. However, many problems regarding the architecture technology co-design need to be sufficiently discussed in the future.

References

[1] 2013. Official RISC-V Benchmark Suites. https://github.com/riscv-software-src/riscv-tests.

[2] 2018. SEMICO Research & Consulting Group. SoC Silicon and Software 2018 Design Cost Analysis: How Rising Costs Impact SoC Design Starts. https://semico.com/sites/default/files/TOC_SC103-18_1.pdf.

[3] 2018. SPEC CPU 2006. https://www.spec.org/cpu2006/.

[4] 2019. RISC-V Specifications Volume 1, Unprivileged Specification version 20191213. https://drive.google.com/file/d/1s0lZxUZaa7eV_O0_WsZzaurFLLww7ou5/view?usp=drive_link.

[5] 2021. RISC-V Specifications Volume 2, Privileged Specification version 20211203. https://drive.google.com/file/d/1EMip5dZlnypTk7pt4WWUKmtjUKTOkBqh/view?usp=drive_link.

[6] 2022. CAD Contest at ICCAD. https://iccad-contest.org/.

[7] 2022. SPEC CPU 2017. https://www.spec.org/cpu2017/.

[8] 2023. Tenstorrent RISC-V OoO Superscalar Processor Family. https://tenstorrent.com/risc-v/.

[9] 2024. Intel Emerald Rapids. https://en.wikipedia.org/wiki/Emerald_Rapids.

[10] 2024. TSMC 3DFabric Technology. https://3dfabric.tsmc.com/english/dedicatedFoundry/technology/3DFabric.htm.

[11] 2024. What is Functional ECO? https://www.synopsys.com/glossary/what-is-functional-eco.html.

[12] Mohamed S Abdelfattah, Łukasz Dudziak, Thomas Chau, Royson Lee, Hyeji Kim, and Nicholas D Lane. 2020. Best of Both Worlds: AutoML Codesign of a CNN and its Hardware Accelerator. In *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[13] Axel Abels, Diederik Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. 2019. Dynamic Weights In Multi-objective Deep Reinforcement Learning. In *International Conference on Machine Learning (ICML)*. PMLR, 11–20.

[14] Santosh G Abraham and B Ramakrishna Rau. 2000. Efficient Design Space Exploration in PICO. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*. 71–79.

[15] Dennis Abts, Garrin Kimmell, Andrew Ling, John Kim, Matt Boyd, Andrew Bitar, Sahil Parmar, Ibrahim Ahmed, Roberto DiCecco, David Han, et al. 2022. A Software-defined Tensor Streaming Multiprocessor for Large-scale Machine Learning. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 567–580.

[16] Abien Fred Agarap. 2018. Deep Learning using Rectified Linear Units (ReLU). *arXiv preprint arXiv:1803.08375* (2018).

[17] Vikas Agarwal, MS Hrishikesh, Stephen W Keckler, and Doug Burger. 2000. Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 248–259.

[18] Ayaz Akram and Lina Sawalha. 2019. Validation of the GEM5 Simulator for x86 Architectures. In *IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 53–58.

[19] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, and B. Nikolić. 2020. Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs. *IEEE Micro* 40, 4 (2020), 10–21.

[20] Jennifer M Anderson, Lance M Berc, Jeffrey Dean, Sanjay Ghemawat, Monika R Henzinger, Shun-Tak A Leung, Richard L Sites, Mark T Vandevoorde, Carl A Waldspurger, and William E Weihl. 1997. Continuous Profiling: Where Have All The Cycles Gone? *ACM Transactions on Computer Systems (TOCS)* 15, 4 (1997), 357–390.

[21] K Arvind and Rishiyur S Nikhil. 1990. Executing a Program on the MIT Tagged-Token Dataflow Architecture. *IEEE Trans. Comput.* 39, 3 (1990), 300–318.

[22] Krste Asanovic, David A Patterson, and Christopher Celio. 2015. The Berkeley Out-of-order Machine (BOOM): An Industry-competitive, Synthesizable, Parameterized RISC-V Processor. (2015).

[23] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. 2016. The Rocket Chip Generator. UCB/EECS-2016-17 (Apr 2016). http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[24] Todd Austin, Eric Larson, and Dan Ernst. 2002. SimpleScalar: An Infrastructure for Computer System Modeling. *Computer* 35, 2 (2002), 59–67.

[25] Omid Azizi, Aqeel Mahesri, Benjamin C Lee, Sanjay Jeram Patel, and Mark Horowitz. 2010. Energy-performance Tradeoffs in Processor Architecture and Circuit Design: a Marginal Cost Analysis. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 26–36.

[26] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: Constructing Hardware in a Scala Embedded Language. In *ACM/IEEE Design Automation Conference (DAC)*. 1212–1221.

[27] Chen Bai, Jiayi Huang, Xuechao Wei, Yuzhe Ma, Sicheng Li, Hongzhong Zheng, Bei Yu, and Yuan Xie. 2023. ArchExplorer: Microarchitecture Exploration Via Bottleneck Analysis. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 268–282.

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

[29] Chen Bai, Qi Sun, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin DF Wong. 2023. BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Ex-

ploration. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* (2023).

[30] Chen Bai, Xuechao Wei, Youwei Zhuo, Yi Cai, Hongzhong Zheng, Bei Yu, and Yuan Xie. 2023. Klotski: DNN Model Orchestration Framework for Dataflow Architecture Accelerators. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–9.

[31] Chen Bai, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin D. F. Wong. 2024. Towards Automated RISC-V Microarchitecture Design with Reinforcement Learning. *AAAI Conference on Artificial Intelligence* 38, 1, 12–20. https://doi.org/10.1609/aaai.v38i1.27750

[32] Paul Barham, Aakanksha Chowdhery, Jeff Dean, Sanjay Ghemawat, Steven Hand, Daniel Hurt, Michael Isard, Hyeontaek Lim, Ruoming Pang, Sudip Roy, et al. 2022. Pathways: Asynchronous Distributed Dataflow for ML. *Machine Learning and Systems (MLSys)* 4 (2022), 430–449.

[33] Scott Beamer and David Donofrio. 2020. Efficiently Exploiting Low Activity Factors to Accelerate RTL Simulation. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[34] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. 2019. Max-value Entropy Search for Multi-objective Bayesian Optimization. *Annual Conference on Neural Information Processing Systems (NIPS)* 32 (2019).

[35] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-parameter Optimization. *Annual Conference on Neural Information Processing Systems (NIPS)* 24 (2011).

[36] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The GEM5 Simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011), 1–7.

[37] Bryan Black, Murali Annavaram, Ned Brekelbaum, John DeVale, Lei Jiang, Gabriel H Loh, Don McCaule, Pat Morrow, Donald W Nelson, Daniel Pantuso, et al. 2006. Die Stacking (3D) Microarchitecture. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 469–479.

[38] Pradip Bose. 2021. The POWER Processor Family: A Historical Perspective From the Viewpoint of Presilicon Modeling. *IEEE Micro* 41, 6 (2021), 71–77.

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

[40] D. Brooks, V. Tiwari, and M. Martonosi. 2000. Wattch: A Framework for Architectural-level Power Analysis and Optimizations. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 83–94.

[41] J Adam Butts and Gurindar S Sohi. 2000. A Static Power Model for Architects. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 191–201.

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

[43] Trevor E Carlson, Wim Heirman, Stijn Eyerman, Ibrahim Hur, and Lieven Eeckhout. 2014. An Evaluation of High-level Mechanistic Core Models. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 3 (2014), 1–25.

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

[45] Po-Yung Chang, Marius Evers, and Yale N Patt. 1997. Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference. *Springer International Journal of Parallel Programming* 25 (1997), 339–362.

[46] Chen Chen, Xiaoyan Xiang, Chang Liu, Yunhai Shang, Ren Guo, Dongqi Liu, Yimin Lu, Ziyi Hao, Jiahui Luo, Zhijian Chen, 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)*. IEEE, 52–64.

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

[48] Tianshi Chen, Qi Guo, Ke Tang, Olivier Temam, Zhiwei Xu, Zhi-Hua Zhou, and Yunji Chen. 2014. ArchRanker: A Ranking Approach to Design Space Exploration. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE.

[49] Yu-Guang Chen, Chun-Yao Wang, Tsung-Wei Huang, and Takashi Sato. 2022. Overview of 2022 CAD Contest at ICCAD. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–3.

[50] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 367–379.

[51] Yu-Hsin Chen, Tushar Krishna, Joel Emer, and Vivienne Sze. 2016. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. In *IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 262–263.

[52] George Z Chrysos and Joel S Emer. 1998. Memory Dependence Prediction Using Store Sets. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 142–153.

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

[54] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. 2020. Differentiable Expected Hypervolume Improvement for Parallel Multi-objective Bayesian Optimization. *Annual Conference on Neural Information Processing Systems (NIPS)* 33 (2020), 9851–9864.

[55] Jack B Dennis. 1980. Data Flow Supercomputers. *Computer* 13, 11 (1980), 48–56.

[56] Jack B Dennis. 2005. First Version of a Data Flow Procedure Language. In *Programming Symposium: Proceedings, Colloque sur la Programmation Paris, April 9–11, 1974*. Springer, 362–376.

[57] Jack B Dennis and David P Misunas. 1974. A Preliminary Architecture for a Basic Data-Flow Processor. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 126–132.

[58] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting Vision Processing Closer to the Sensor. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 92–104.

[59] Christophe Dubach, Timothy Jones, and Michael O'Boyle. 2007. Microarchitectural Design Space Exploration Using an Architecture-centric Approach. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 262–271.

[60] Christophe Dubach, Timothy M Jones, and Michael FP O'Boyle. 2008. Exploring and predicting the architecture/optimising compiler co-design space. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*. 31–40.

[61] Philip G Emma. 1997. Understanding Some Simple Processor-performance Limits. *IBM Journal of Research and Development* 41, 3 (1997), 215–232.

[62] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E Smith. 2006. A Performance Counter Architecture for Computing Accurate CPI Components. *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* 41, 11 (2006), 175–184.

[63] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E Smith. 2009. A Mechanistic Performance Model for Superscalar Out-of-order Processors. *ACM Transactions on Computer Systems (TOCS)* 27, 2 (2009), 1–37.

[64] M. L. Fair, C. R. Conklin, S. B. Swaney, P. J. Meaney, W. J. Clarke, L. C. Alves, I. N. Modi, F. Freier, W. Fischer, and N. E. Weber. 2004. Reliability, Availability, and Serviceability (RAS) of the IBM eServer z990. *IBM Journal of Research and Development* 48, 3.4 (2004), 519–534. https://doi.org/10.1147/rd.483.0519

[65] Kai-Tai Fang and Yuan Wang. 1993. *Number-theoretic Methods In Statistics*. Vol. 51. CRC Press.

[66] Johannes Feldmann, Kira Kraft, Lukas Steiner, Norbert Wehn, and Matthias Jung. 2020. Fast and Accurate DRAM Simulation: Can We Further Accelerate It?. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 364–369.

[67] Brian Fields, Rastislav Bodik, and Mark D Hill. 2002. Slack: Maximizing Performance Under Technological Constraints. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 47–58.

[68] Brian Fields, Shai Rubin, and Rastislav Bodik. 2001. Focusing Processor Policies via Critical-path Prediction. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 74–85.

[69] Brian A Fields, Rastislav Bodik, Mark D Hill, and Chris J Newburn. 2003. Using Interaction Costs for Microarchitectural Bottleneck Analysis. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 228–239.

[70] Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. 2003. An Efficient Boosting Algorithm for Combining Preferences. *Journal of Machine Learning Research (JMLR)* 4, Nov (2003), 933–969.

[71] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. Tetris: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 751–764.

[72] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. 2019. Tangram: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 807–820.

[73] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, et al. 2021. Gemmini: Enabling Systematic Deep-learning Architecture Evaluation via Full-stack Integration. In *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 769–774.

[74] Graham Gobieski, Souradip Ghosh, Marijn Heule, Todd Mowry, Tony Nowatzki, Nathan Beckmann, and Brandon Lucia. 2022. RipTide: A Programmable, Energy-Minimal Dataflow Compiler and Architecture. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 546–564. https://doi.org/10.1109/MICRO56248.2022.00046

[75] Hossein Golestani, Rathijit Sen, Vinson Young, and Gagan Gupta. 2022. Calipers: A Criticality-aware Framework for Modeling Processor Performance. *ACM International Conference on Supercomputing (ICS)* (2022).

[76] Wilfred Gomes, Altug Koker, Pat Stover, Doug Ingerly, Scott Siers, Srikrishnan Venkataraman, Chris Pelto, Tejas Shah, Amreesh Rao, Frank O'Mahony, et al. 2022. Ponte Vecchio: A Multi-Tile 3D Stacked Processor for Exascale Computing. In *IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. IEEE, 42–44.

[77] V Gerald Grafe and Jamie E Hoch. 1990. The Epsilon-2 Multiprocessor System. *J. Parallel and Distrib. Comput.* 10, 4 (1990), 309–318.

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

[79] John R. Gurd, Chris C. Kirkham, and Ian Watson. 1985. The Manchester Prototype Dataflow Computer. *Commun. ACM* 28, 1 (1985), 34–52.

[80] Anthony Gutierrez, Joseph Pusdesris, Ronald G Dreslinski, Trevor Mudge, Chander Sudanthi, Christopher D Emmons, Mitchell Hayenga, and Nigel Paver. 2014. Sources of Error in Full-system Simulation. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 13–22.

[81] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning (ICML)*. PMLR, 1861–1870.

[82] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 243–254.

[83] John L Henning. 2006. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News* 34, 4 (2006), 1–17.

[84] Mark D Hill and Alan Jay Smith. 1989. Evaluating Associativity in CPU Caches. *IEEE Trans. Comput.* 38, 12 (1989), 1612–1630.

[85] MS Hrishikesh, Norman P Jouppi, Keith I Farkas, Doug Burger, Stephen W Keckler, and Premkishore Shivakumar. 2002. The Optimal Logic Depth per Pipeline Stage is 6 to 8 FO4 Inverter Delays. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 14–24.

[86] M. Y. Hsiao, W. C. Carter, J. W. Thomas, and W. R. Stringfellow. 1981. Reliability, Availability, and Serviceability of IBM Computer Systems: A Quarter Century of Progress. *IBM Journal of Research and Development* 25, 5 (1981), 453–468. https://doi.org/10.1147/rd.255.0453

[87] Qijing Huang, Minwoo Kang, Grace Dinh, Thomas Norell, Aravind Kalaiah, James Demmel, John Wawrzynek, and Yakun Sophia Shao. 2021.

CoSA: Scheduling by Constrained Optimization for Spatial Accelerators. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 554–566.

[88] Ali R Hurson and Krishna M Kavi. 2007. Dataflow Computers: Their History and Future. *Wiley Encyclopedia of Computer Science and Engineering* (2007).

[89] W. Hwu and Y. N. Patt. 1986. HPSm, A High Performance Restricted Data Flow Architecture Having Minimal Functionality. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)* (Tokyo, Japan). 297–306.

[90] Drago Ignjatović, Daniel W. Bailey, and Ljubisa Bajić. 2022. The Wormhole AI Training Processor. In *IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. 356–358. https://doi.org/10.1109/ISSCC42614.2022.9731633

[91] Engin İpek, Sally A McKee, Rich Caruana, Bronis R de Supinski, and Martin Schulz. 2006. Efficiently Exploring Architectural Design Spaces Via Predictive Modeling. *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* 40, 5 (2006), 195–206.

[92] Engin Ipek, Sally A McKee, Karan Singh, Rich Caruana, Bronis R de Supinski, and Martin Schulz. 2008. Efficient Architectural Design Space Exploration via Predictive Modeling. *ACM Transactions on Architecture and Code Optimization (TACO)* 4, 4 (2008), 1–34.

[93] Hanhwi Jang, Jae-Eon Jo, Jaewon Lee, and Jangwoo Kim. 2018. RpStacks-MT: A High-throughput Design Evaluation Methodology for Multi-core Processors. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 586–599.

[94] PJ Joseph, Kapil Vaswani, and Matthew J Thazhuthaveetil. 2006. Construction and Use of Linear Regression Models for Processor Performance Analysis. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 99–108.

[95] Norman P. Jouppi. 1989. The Nonuniform Distribution of Instruction-level and Machine Parallelism and Its Effect on Performance. *IEEE Trans. Comput.* 38, 12 (1989), 1645–1658.

[96] Norman P Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, et al. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine

Learning with Hardware Support for Embeddings. *IEEE/ACM International Symposium on Computer Architecture (ISCA)* (2023).

[97] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter Performance Analysis of a Tensor Processing Unit. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 1–12.

[98] Andrew B Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. 2009. ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-stage Design Space Exploration. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. IEEE, 423–428.

[99] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. 2020. Confuciux: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 622–636.

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

[101] Tejas S Karkhanis and James E Smith. 2007. Automated Design of Application Specific Superscalar Processors: An Analytical Approach. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 402–411.

[102] T Karn, Shishpal Rawat, Desmond Kirkpatrick, Rabindra Roy, Gregory S Spirakis, Naveed Sherwani, and Craig Peterson. 2000. EDA Challenges Facing Future Microprocessor Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 19, 12 (2000), 1498–1506.

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

[104] Richard E Kessler. 1999. The Alpha 21264 Microprocessor. *IEEE Micro* 19, 2 (1999), 24–36.

[105] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. 2016. Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 380–392. https://doi.org/10.1109/ISCA.2016.41

[106] Young-Il Kim and Chong-Min Kyung. 2004. Automatic Translation of Behavioral Testbench for Fully Accelerated Simulation. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 218–221.

[107] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[108] Toru Koizumi, Ryota Shioya, Shu Sugita, Taichi Amano, Yuya Degawa, Junichiro Kadomoto, Hidetsugu Irie, and Shuichi Sakai. 2023. Clockhands: Rename-free Instruction Set Architecture for Out-of-order Processors. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–16.

[109] Ronny KRASHINSKY, Christopher BATTEN, Mark HAMPTON, Steve GERDING, Brian PHARRIS, and Jared CASPER. 2004. The Vector-Thread Architecture. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 52–63.

[110] Srivatsan Krishnan, Amir Yazdanbakhsh, Shvetank Prakash, Jason Jabbour, Ikechukwu Uchendu, Susobhan Ghosh, Behzad Boroujerdian, Daniel Richins, Devashree Tripathy, Aleksandra Faust, et al. 2023. ArchGym: An Open-Source Gymnasium for Machine Learning Assisted Architecture Design. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 1–16.

[111] Balasubramanian Kumar and Edward S. Davidson. 1980. Computer System Design Using a Hierarchical Approach to Performance Evaluation. *Commun. ACM* 23, 9 (1980), 511–521.

[112] Metin KUZHAN and Veysel Harun ŞAHİN. 2020. MBBench: A WCET Benchmark Suite. *Sakarya University Journal of Computer and Information Sciences* 3, 1 (2020), 40–50.

[113] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. Understanding Reuse, Performance, and Hardware Cost of DNN Dataflows: A Data-Centric Approach. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 754–768.

[114] Benjamin C. Lee and David M. Brooks. 2006. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 185–194. https://doi.org/10.1145/1168857.1168881

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

[116] Jaewon Lee, Hanhwi Jang, and Jangwoo Kim. 2014. RpStacks: Fast and Accurate Processor Design Space Exploration Using Representative Stall-event Stacks. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 255–267.

[117] Yunsup Lee, Andrew Waterman, Henry Cook, Brian Zimmer, Ben Keller, Alberto Puggelli, Jaehwa Kwak, Ruzica Jevtic, Stevo Bailey, Milovan Blagojevic, et al. 2016. An Agile Approach to Building RISC-V Microprocessors. *IEEE Micro* 36, 2 (2016), 8–20.

[118] Dandan Li, Shuzhen Yao, Yu-Hang Liu, Senzhang Wang, and Xian-He Sun. 2016. Efficient Design Space Exploration Via Statistical Sampling and AdaBoost Learning. In *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[119] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. 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.

[120] Sicheng Li, Chen Bai, Xuechao Wei, Bizhao Shi, Yen-Kuang Chen, and Yuan Xie. 2022. 2022 ICCAD CAD Contest Problem C: Microarchitecture Design Space Exploration. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–7.

[121] Shuangnan Liu, Francis CM Lau, and Benjamin Carrion Schafer. 2019. Accelerating FPGA Prototyping Through Predictive Model-based HLS Design Space Exploration. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[122] Gabriel H Loh, Yuan Xie, and Bryan Black. 2007. Processor Design in 3D Die-Stacking Technologies. *IEEE Micro* 27, 3 (2007), 31–48.

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

[124] Yuzhe Ma, Subhendu Roy, Jin Miao, Jiamin Chen, and Bei Yu. 2018. Cross-layer Optimization for High Speed Adders: A Pareto Driven Machine Learning

Approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 38, 12 (2018), 2298–2311.

[125] Yuzhe Ma, Ziyang Yu, and Bei Yu. 2019. CAD Tool Design Space Exploration via Bayesian Optimization. In *ACM/IEEE Workshop on Machine Learning CAD (MLCAD)*. 1–6.

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

[127] Paolo Mantovani, Davide Giri, Giuseppe Di Guglielmo, Luca Piccolboni, Joseph Zuckerman, Emilio G Cota, Michele Petracca, Christian Pilato, and Luca P Carloni. 2020. Agile SoC Development with Open ESP. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–9.

[128] Francisco Javier Mesa-Martinez, Joseph Nayfach-Battilana, and Jose Renau. 2007. Power Model Validation Through Thermal Measurements. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 302–311.

[129] Pierre Michaud, André Seznec, and Stéphan Jourdan. 2001. An Exploration of Instruction Fetch Requirement in Out-of-Order Superscalar Processors. *Springer International Journal of Parallel Programming* 29 (2001), 35–58.

[130] Tom M Mitchell. 1999. Machine Learning and Data Mining. *Commun. ACM* 42, 11 (1999), 30–36.

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

[132] Mayan Moudgill, Pradip Bose, and Jaime H Moreno. 1999. Validation of Turandot, a Fast Processor Model for Microarchitecture Exploration. In *International Performance Computing and Communications Conference (IPCCC)*. 451–457.

[133] Mayan Moudgill, J-D Wellman, and Jaime H Moreno. 1999. Environment for PowerPC Microarchitecture Exploration. *IEEE/ACM International Symposium on Microarchitecture (MICRO)* 19, 3 (1999), 15–25.

[134] Trevor Mudge. 2001. Power: A First-Class Architectural Design Constraint. *Computer* 34, 4 (2001), 52–58.

[135] Johannes M Mulder, Nhon T Quach, and Michael J Flynn. 1991. An Area Model for On-Chip Memories and its Application. *IEEE Journal Solid-State Circuits* 26, 2 (1991), 98–106.

[136] Samuel Naffziger, Noah Beck, Thomas Burd, Kevin Lepak, Gabriel H Loh, Mahesh Subramony, and Sean White. 2021. Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families: Industrial Product. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 57–70.

[137] Ramadass Nagarajan, Xia Chen, Robert G McDonald, Doug Burger, and Stephen W Keckler. 2006. Critical Path Analysis of the TRIPS Architecture. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 37–47.

[138] Ramadass Nagarajan, Karthikeyan Sankaralingam, Doug Burger, and Stephen W Keckler. 2001. A Design Space Evaluation of Grid Processor Architectures. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 40–51.

[139] Rishiyur S Nikhil. 1989. Can Dataflow Subsume von Neumann Computing?. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 262–272.

[140] Rishiyur S Nikhil, Gregory M Papadopoulos, and Arvind. 1992. T: A Multithreaded Massively Parallel Architecture. *IEEE/ACM International Symposium on Computer Architecture (ISCA)* 20, 2, 156–167.

[141] Derek B Noonburg and John P Shen. 1994. Theoretical Modeling of Superscalar Processor Performance. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.

[142] Tony Nowatzki, Vinay Gangadhar, and Karthikeyan Sankaralingam. 2019. Heterogeneous Von Neumann/Dataflow Microprocessors. *Commun. ACM* 62, 6 (2019), 83–91.

[143] Tony Nowatzki, Venkatraman Govindaraju, and Karthikeyan Sankaralingam. 2015. A Graph-based Program Representation for Analyzing Hardware Specialization Approaches. *IEEE Computer Architecture Letters (CAL)* 14, 2 (2015), 94–98.

[144] Tony Nowatzki and Karthikeyan Sankaralingam. 2016. Analyzing Behavior Specialized Acceleration. *ACM International Conference on Architectural Support*

for Programming Languages and Operating Systems (ASPLOS) 51, 4 (2016), 697–711.

[145] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[146] Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. 2009. ReSPIR: A Response Surface-based Pareto Iterative Refinement For Application-specific Design Space Exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 28, 12 (2009), 1816–1829.

[147] GM Papadopoulos and DE Culler. 1990. Monsoon: an Explicit Token-Store Architecture. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, 82–91.

[148] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer. 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 304–315.

[149] Y. N. Patt, W. M. Hwu, and M. Shebanow. 1985. HPS, A New Microarchitecture: Rationale and Introduction. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 103–108. https://doi.org/10.1145/18927.18916

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

[151] Erez Perelman, Greg Hamerly, Michael Van Biesbrouck, Timothy Sherwood, and Brad Calder. 2003. Using SimPoint for Accurate and Efficient Simulation. *ACM SIGMETRICS Performance Evaluation Review* 31, 1 (2003), 318–319.

[152] Chris H Perleberg and Alan Jay Smith. 1993. Branch Target Buffer Design and Optimization. *IEEE Trans. Comput.* 42, 4 (1993), 396–412.

[153] Laurence J Peter, Raymond Hull, et al. 1969. *The Peter Principle*. Vol. 4. Souvenir Press London.

[154] Daniel Petrisko, Farzam Gilani, Mark Wyse, Dai Cheol Jung, Scott Davidson, Paul Gao, Chun Zhao, Zahra Azad, Sadullah Canakci, Bandhav Veluri, et al. 2020. BlackParrot: An Agile Open-source RISC-V Multicore for Accelerator SoCs. *IEEE Micro* 40, 4 (2020), 93–102.

[155] Ramakrishna B Rau and George E Rossmann. 1977. The Effect of Instruction Fetch Strategies upon the Performance of Pipelined Instruction Units. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 80–89.

[156] Behnam Robatmili, Sibi Govindan, Doug Burger, and Stephen W Keckler. 2011. Exploiting Criticality to Reduce Bottlenecks in Distributed Uniprocessors. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 431–442.

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

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

[159] Sami Salamin, Martin Rapp, Anuj Pathania, Arka Maity, Jörg Henkel, Tulika Mitra, and Hussam Amrouch. 2021. Power-Efficient Heterogeneous Many-Core Design With NCFET Technology. *IEEE Trans. Comput.* 70, 9 (2021), 1484–1497. https://doi.org/10.1109/TC.2020.3013567

[160] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Doug Burger, Stephen W Keckler, and Charles R Moore. 2003. Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 422–433.

[161] Karthikeyan Sankaralingam, Tony Nowatzki, Vinay Gangadhar, Preyas Shah, Michael Davies, William Galliher, Ziliang Guo, Jitu Khare, Deepak Vijay, Poly Palamuttam, Maghawan Punde, Alex Tan, Vijay Thiruvengadam, Rongyi Wang, and Shunmiao Xu. 2022. The Mozart Reuse Exposed Dataflow Processor for AI and Beyond: Industrial Product. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)* (New York, New York). 978–992. https://doi.org/10.1145/3470496.3533040

[162] Majid Sarrafzadeh and CK Wong. 1996. *An Introduction to VLSI Physical Design*. McGraw-Hill Higher Education.

[163] Tsutomu Sasao. 1993. *Logic Synthesis and Optimization*. Vol. 2. Springer.

[164] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The Graph Neural Network Model. *IEEE Transactions on Neural Networks (TNN)* 20, 1 (2008), 61–80.

[165] Robert E Schapire. 2003. The Boosting Approach to Machine Learning: An Overview. *Nonlinear estimation and classification* (2003), 149–171.

[166] Robert E Schapire and Yoav Freund. 2013. Boosting: Foundations and algorithms. *Kybernetes* 42, 1 (2013), 164–166.

[167] Robert Schreiber, Shail Aditya, Scott Mahlke, Vinod Kathail, B Ramakrishna Rau, Darren Cronquist, and Mukund Sivaraman. 2002. PICO-NPA: High-level synthesis of nonprogrammable hardware accelerators. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology* 31 (2002), 127–142.

[168] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2020. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature* 588, 7839 (2020), 604–609.

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

[170] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[171] André Seznec. 2011. A New Case for the Tage Branch Predictor. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 117–127.

[172] André Seznec and Pierre Michaud. 2006. A Case for (partially) TAgged GEometric History Length Branch Prediction. *The Journal of Instruction-Level Parallelism* 8 (2006), 23.

[173] Amar Shah and Zoubin Ghahramani. 2016. Pareto Frontier Learning with Expensive Correlated Objectives. In *International Conference on Machine Learning (ICML)*. 1919–1927.

[174] Amar Shah and Zoubin Ghahramani. 2016. Pareto Frontier Learning with Expensive Correlated Objectives. In *International Conference on Machine Learning (ICML)*. PMLR, 1919–1927.

[175] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=B1ckMDqlg

[176] Yongming Shen, Michael Ferdman, and Peter Milder. 2017. Maximizing CNN accelerator efficiency through resource partitioning. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, 535–547.

[177] T Shimada, K Hiraki, K Nishida, and S Sekiguchi. 1986. Evaluation of a Prototype Data Flow Processor of the SIGMA-1 for Scientific Computations. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 226–234.

[178] Premkishore Shivakumar and Norman P Jouppi. 2001. CACTI 3.0: An Integrated Cache Timing, Power, and Area Model. (2001).

[179] Durga L Shrestha and Dimitri P Solomatine. 2006. Experiments with AdaBoost. RT, An Improved Boosting Scheme for Regression. *Neural computation* 18, 7 (2006), 1678–1710.

[180] Kevin Skadron, Pritpal S Ahuja, Margaret Martonosi, and Douglas W Clark. 1998. Improving Prediction for Procedure Returns with Return-Address-Stack Repair Mechanisms. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 259–271.

[181] James E. Smith and Andrew R. Pleszkun. 1988. Implementing Precise Interrupts in Pipelined Processors. *IEEE Trans. Comput.* 37, 5 (1988), 562–573.

[182] Guangyu Sun, Christopher Hughes, Changkyu Kim, Jishen Zhao, Cong Xu, Yuan Xie, and Yen-Kuang Chen. 2011. Moguls: A Model to Explore the Memory Hierarchy for Bandwidth Improvements. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 377–388.

[183] Qi Sun, Tinghuan Chen, Siting Liu, Jin Miao, Jianli Chen, Hao Yu, and Bei Yu. 2021. Correlated Multi-objective Multi-fidelity Optimization for HLS Directives Design. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*.

[184] Richard S Sutton, Andrew G Barto, et al. 1998. *Introduction to Reinforcement Learning*. Vol. 135. MIT press Cambridge.

[185] Tarek M Taha and Scott Wills. 2008. An Instruction Throughput Model of Superscalar Processors. *IEEE Trans. Comput.* 57, 3 (2008), 389–403.

[186] Teruo Tanimoto, Takatsugu Ono, Koji Inoue, and Hiroshi Sasaki. 2017. Enhanced Dependence Graph Model for Critical Path Analysis on Modern Out-of-order Processors. *IEEE Computer Architecture Letters (CAL)* 16, 2 (2017), 111–114.

[187] Ben Tu, Axel Gandy, Nikolas Kantas, and Behrang Shafei. 2022. Joint Entropy Search for Multi-objective Bayesian Optimization. *Annual Conference on Neural Information Processing Systems (NIPS)* 35 (2022), 9922–9938.

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

[189] Peter JM Van Laarhoven and Emile HL Aarts. 1987. Simulated Annealing. In *Simulated annealing: Theory and applications*. Springer, 7–15.

[190] Vinay Vashishtha, Manoj Vangala, and Lawrence T Clark. 2017. ASAP7 Predictive Design Kit Development and Cell Design Technology Co-optimization. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 992–998.

[191] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *Annual Conference on Neural Information Processing Systems (NIPS)* 30.

[192] Arthur H Veen. 1986. Dataflow Machine Architecture. *Comput. Surveys* 18, 4 (1986), 365–396.

[193] Swagath Venkataramani, Ashish Ranjan, Subarno Banerjee, Dipankar Das, Sasikanth Avancha, Ashok Jagannathan, Ajaya Durg, Dheemanth Nagaraj, Bharat Kaul, Pradeep Dubey, et al. 2017. SCALEDEEP: A Scalable Compute Architecture for Learning and Evaluating Deep Networks. In *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 13–26.

[194] Chris Williams, Edwin V Bonilla, and Kian M Chai. 2007. Multi-task Gaussian Process Prediction. *Annual Conference on Neural Information Processing Systems (NIPS)*, 153–160.

[195] Christopher K Williams and Carl Edward Rasmussen. 2006. *Gaussian Processes for Machine Learning*. Vol. 2. MIT press Cambridge, MA.

[196] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. 2016. Deep Kernel Learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR, 370–378.

[197] Steven JE Wilton and Norman P Jouppi. 1996. CACTI: An Enhanced Cache Access and Cycle Time Model. *IEEE Journal Solid-State Circuits* 31, 5 (1996), 677–688.

[198] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2020), 4–24.

[199] John Wuu, Rahul Agarwal, Michael Ciraula, Carl Dietz, Brett Johnson, Dave Johnson, Russell Schreiber, Raja Swaminathan, Will Walker, and Samuel Naffziger. 2022. 3D V-Cache: the Implementation of a Hybrid-Bonded 64MB Stacked Cache for a 7nm x86-64 CPU. In *IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. IEEE, 428–429.

[200] Yinan Xu, Zihao Yu, Dan Tang, Guokai Chen, Lu Chen, Lingrui Gou, Yue Jin, Qianruo Li, Xin Li, Zuojun Li, et al. 2022. Towards Developing High Performance RISC-V Processors Using Agile Methodology. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1178–1199.

[201] Jawad Haj Yahya, Jeremie S Kim, A Giray Yağlıkçı, Jisung Park, Efraim Rotem, Yanos Sazeides, and Onur Mutlu. 2022. DarkGates: A Hybrid Power-Gating Architecture to Mitigate the Performance Impact of Dark-Silicon in High Performance Processors. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1170–1183.

[202] Shun Yamaguchi, Mahfuzul Islam, Takashi Hisakado, and Osami Wada. 2023. A Fully Synchronous Digital LDO with Built-in Adaptive Frequency Modulation and Implicit Dead-zone Control. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 186–187.

[203] Lei Yang, Zheyu Yan, Meng Li, Hyoukjun Kwon, Liangzhen Lai, Tushar Krishna, Vikas Chandra, Weiwen Jiang, and Yiyu Shi. 2020. Co-Exploration of Neural Architectures and Heterogeneous ASIC Accelerator Designs Targeting Multiple Tasks. In *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[204] Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. 2019. A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation.

In *Annual Conference on Neural Information Processing Systems (NIPS)*. Article 1311, 12 pages.

[205] Ahmad Yasin. 2014. A Top-down Method for Performance Analysis and Counters Architecture. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 35–44.

[206] Kenneth C Yeager. 1996. The MIPS R10000 Superscalar Microprocessor. *IEEE Micro* 16, 2 (1996), 28–41.

[207] Tse-Yu Yeh and Yale N Patt. 1991. Two-level Adaptive Training Branch Prediction. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 51–61.

[208] Joshua J Yi, David J Lilja, and Douglas M Hawkins. 2003. A Statistically Rigorous Approach for Improving Simulation Methodology. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 281–291.

[209] Kai Yu, Jinbo Bi, and Volker Tresp. 2006. Active Learning Via Transductive Experimental Design. In *International Conference on Machine Learning (ICML)*. 1081–1088.

[210] Ziyang Yu, Chen Bai, Shoubo Hu, Ran Chen, Taohai He, Mingxuan Yuan, Bei Yu, and Martin Wong. 2023. IT-DSE: Invariance Risk Minimized Transfer Microarchitecture Design Space Exploration. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–9.

[211] Jianwang Zhai, Chen Bai, Binwu Zhu, Yici Cai, Qiang Zhou, and Bei Yu. 2021. McPAT-Calib: A Microarchitecture Power Modeling Framework for Modern CPUs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–9.

[212] Dan Zhang, Safeen Huda, Ebrahim Songhori, Kartik Prabhu, Quoc Le, Anna Goldie, and Azalia Mirhoseini. 2022. A Full-stack Search Technique for Domain Optimized Deep Learning Accelerators. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 27–42.

[213] Shuhan Zhang, Fan Yang, Dian Zhou, and Xuan Zeng. 2020. An Efficient Asynchronous Batch Bayesian Optimization Approach for Analog Circuit Synthesis. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[214] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. 2020. SonicBOOM: The 3rd Generation Berkeley Out-of-order Machine. In *Workshop on Computer Architecture Research with RISC-V (CARRV)*, Vol. 5.

[215] Zhi-Hua Zhou. 2012. *Ensemble Methods: Foundations and Algorithms*. CRC press.

[216] V. Zivojnovic, J. Martinez, C. Schläger, and Heinrich Meyr. 1994. DSPstone: A DSP-Oriented Benchmarking Methodology. In *International Conference on Signal Processing Applications and Technology (ICSPAT)*.