MICRO 2023
56th IEEE/ACM International Symposium on Microarchitecture

ArchExplorer: Microarchitecture Exploration Via Bottleneck Analysis

Chen Bai[1,4] Jiayi Huang[2] Xuechao Wei[4] Yuzhe Ma[2]
Sicheng Li[4] Hongzhong Zheng[4] Bei Yu[1] Yuan Xie[3,4]

[1] The Chinese University of Hong Kong
[2] The Hong Kong University of Science and Technology (Guangzhou)
[3] Hong Kong University of Science and Technology
[4] DAMO Academy, Alibaba Group

Oct. 30, 2023

Outline

Introduction

Introduction: Problem Formulation

Microprocessor Microarchitecture Design Space Exploration (DSE)

Given benchmark suites and microprocessor microarchitecture design space, find optimal microarchitecture parameters that can achieve good trade-offs between performance, power, and area (PPA).

Introduction: Previous Methodologies

- Industry:
  - Expertise of computer architects.
- Academia:
  - Analytical methodologies: based on mechanistic models with intepretable equations.[1][2][3][4]
  - Black-box methodologies: based on machine-learning techniques.

---

[1] Mark D Hill and Alan Jay Smith (1989). "Evaluating Associativity in CPU Caches". In: *IEEE Transactions on Computers* 38.12, pp. 1612–1630; MS Hrishikesh et al. (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, pp. 14–24.

[2] Guangyu Sun et al. (2011). "Moguls: A Model to Explore the Memory Hierarchy for Bandwidth Improvements". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 377–388.

[3] 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)*, pp. 402–411.

[4] Stijn Eyerman et al. (2009). "A Mechanistic Performance Model for Superscalar Out-of-order Processors". In: *ACM Transactions on Computer Systems (TOCS)* 27.2, pp. 1–37.

Introduction: Previous Methodologies

- Black-box methodologies: based on machine-learning techniques.
  Some representative approaches:
  - Regression[5][6][7]
  - Ranking[8]
  - Bayesian optimization[9]
  - *etc.*

---

[5] Engin Ïpek et al. (2006). "Efficiently Exploring Architectural Design Spaces Via Predictive Modeling". In: *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* 40.5, pp. 195–206.

[6] 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, pp. 340–351.

[7] Dandan Li et al. (2016). "Efficient Design Space Exploration Via Statistical Sampling and AdaBoost Learning". In: *ACM/IEEE Design Automation Conference (DAC)*. IEEE, pp. 1–6.

[8] Tianshi Chen et al. (2014). "ArchRanker: A Ranking Approach to Design Space Exploration". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE.

[9] Chen Bai et al. (2021). "BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, pp. 1–9.

Introduction: Limitations of Previous Methodologies

- Industry solutions $\rightarrow$ architects' bias: whether the solution given by architects is optimal or how many benefits we can gain based on a sub-optimal solution?
- Analytical methodologies $\rightarrow$ require immense domain knowledge.
- Black-box methodologies $\rightarrow$ require high computing resources.

Introduction: Goal & Approach

Goal

Solve the problem by removing limitations of previous methodologies: remove massive domain knowledge requirements & mitigate the high computing demands.

Approach

DSE via automated bottleneck analysis.

Introduction: Rationales (Thought Experiment)

Perfect machine: unlimited hardware resources.

- Performance is constrained only by program's true data dependencies.

Real machine: limited hardware resources.

- Performance is constrained by program's true data dependencies and **resource constraints**.
- Two distinct types of resources:
  - deficient and exhausted.
  - abundant and idle.

Resource constraints $\rightarrow$ the usage dependencies of deficient resources that blocks instructions from progressing.

Balanced Microarchitecture

A balanced microarchitecture can simultaneously maximize the utilization of each hardware resource.

Bottleneck

We refer to a bottleneck as insufficient hardware resource that is exhausted by instructions and results in high program runtime.

The Key to Success

Accurate and efficient identification of types of hardware resources is the first principle to finding a balanced microarchitecture.

Findings

We find that the relations between resource constraints and machine parallelism are similar to the cask effect.[10][11]

How to identify the type of resources?

- The utilization status of each resource in the microexecution should be captured.
- Whether the overlapping events matter for the execution time should be considered.
  → Call for a global view of the entire microexecution, which the critical path analysis can help[12].

[10] Norman P. Jouppi (1989). "The Nonuniform Distribution of Instruction-level and Machine Parallelism and Its Effect on Performance". In: *IEEE Transactions on Computers* 38.12, pp. 1645–1658.

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

[12] 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, pp. 74–85.

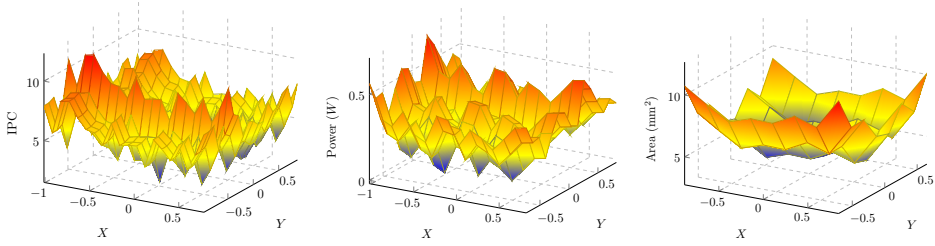
Design Principles for DSE via Bottleneck Analysis

- The dependencies contributing to execution time should be captured as much as possible.
- Concurrent events should be distinguishable.

ArchExplorer is the implementation of the design principles.

Background & Motivation

Background & Motivation: Challenges



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

A visualization of the design space for 458.sjeng.

Challenges in microprocessor microarchitecture DSE:

- Complicated design space.
- High simulation runtime.

Background & Motivation: Findings

Bottleneck Analysis Matters in DSE

Removing microarchitecture bottlenecks can significantly enhance the PPA trade-off.

A Straightforward Heuristic

In the DSE, assigning necessary hardware resources and reducing redundant ones.

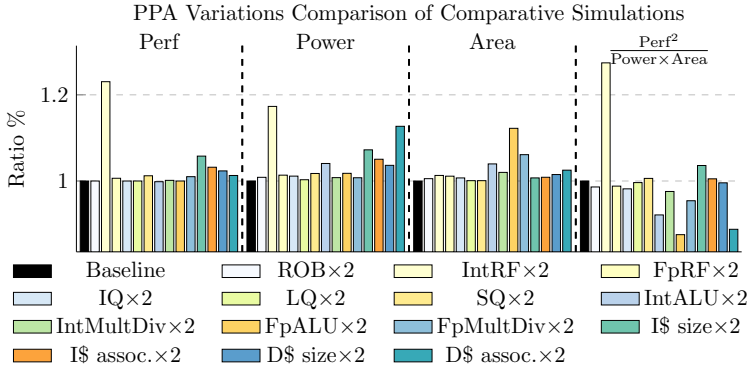
- DSE starts from a baseline microarchitecture.
- Adjust resources based on the degree of necessity manually.
  - The ratio of delayed instructions due to the resource's insufficiency.

Background & Motivation: Findings

Table: 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

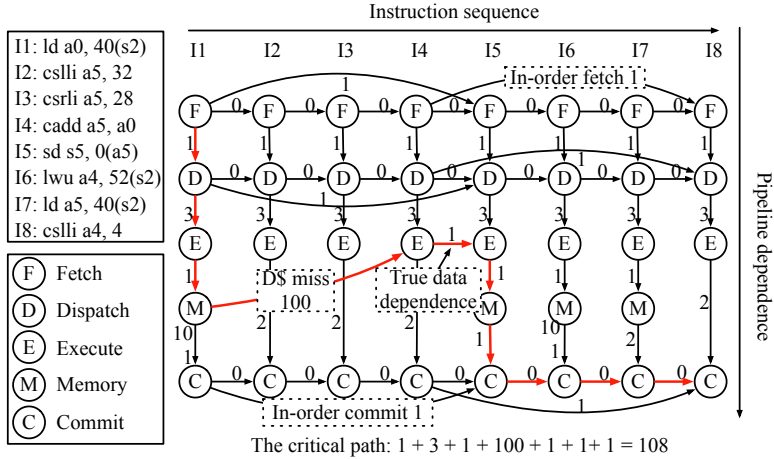
Background & Motivation: Findings



The bar, *e.g.*, "ROB ×2", indicates the microarchitecture is the same as the baseline except that it doubles ROB.

- Doubling the number of physical integer registers improves performance by 23.05% and enhances the PPA trade-off by 27.42%.
- Most instructions are stalled due to insufficient physical integer registers, which results in 25.71% of instructions in 657.xz_s and 18.94% for 625.x264_s getting stalled during renaming.

Background & Motivation: Critical Path Analysis



An overview of the dynamic event-dependence graph.

- Critical path analysis[13]

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

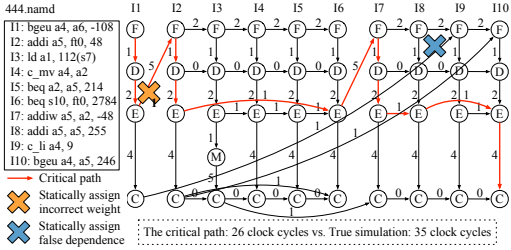
Background & Motivation: Critical Path Analysis

The former dynamic event dependence graph is inaccurate:

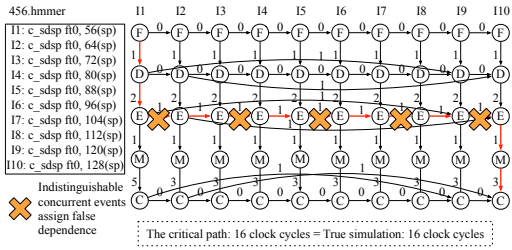
- The dependence and weights assignment are static without adhering to actual microexecution.
- The critical path cannot accurately characterize the bottlenecks' contributions to the overall runtime, even if the modeled critical path length is strictly identical to the simulation runtime.

Lessons Learned & Design Principles

Lessons Learned & Design Principles



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



(b) Previous DEG formulation cannot distinguish overlapped events.

(a) and (b) uses Calipers[a] to demonstrate three kinds of error sources.

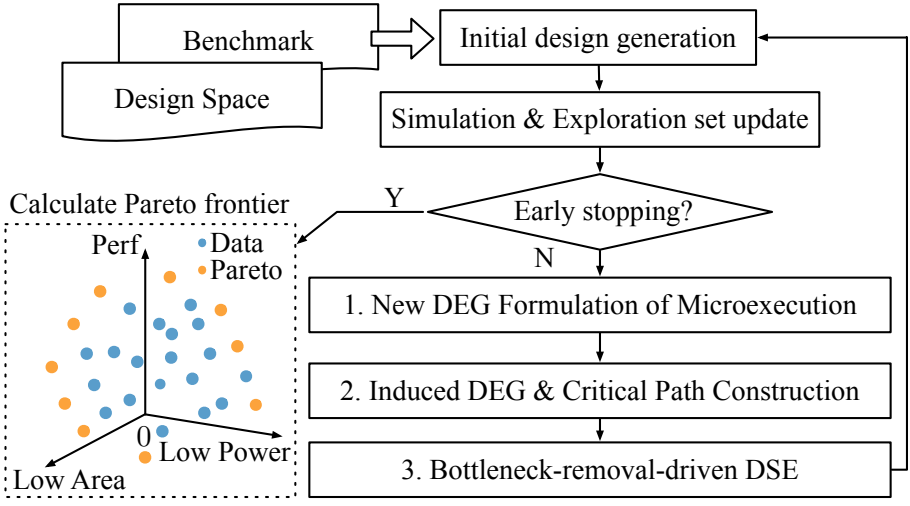
[a] Hossein Golestani et al. (2022). "Calipers: A Criticality-aware Framework for Modeling Processor Performance". In: *ACM International Conference on Supercomputing (ICS)*.

Design Principles

- The dependencies contributing to execution time should be captured as much as possible.
  - Capturing more resource usages improves the utilization approximation.
- Concurrent events should be distinguishable.
  - The distinguishability unveils whether we matter a concurrent event for bottleneck contributions to the overall execution time.

The ArchExplorer Approach

The ArchExplorer Approach: Overview



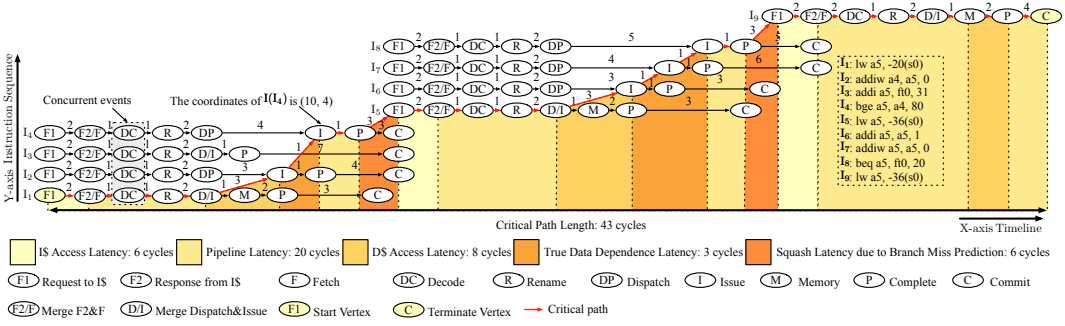
An overview of the ArchExplorer approach.

The ArchExplorer Approach: New DEG Formulation

Highlights of new DEG formulation:

- Nodes represent pipeline stages, and edges represent dependencies.
- Align instructions *w.r.t.* the time instead of pipeline stages.
- Dynamic DEG construction.
- Ascertain the overlapped events.

The ArchExplorer Approach: New DEG Formulation

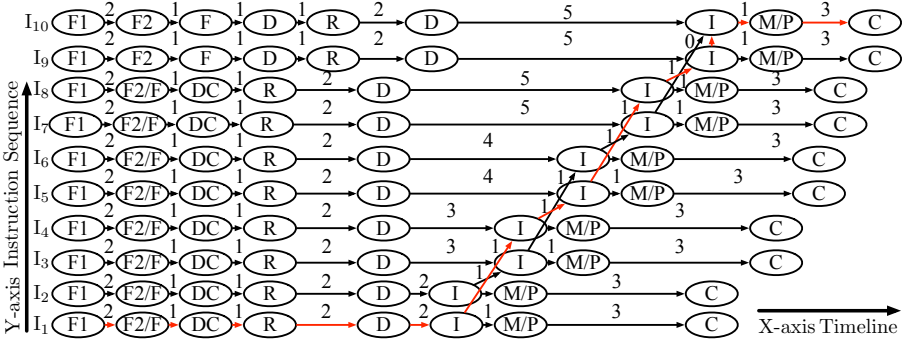


An overview of the new DEG formulation of microexecution. The critical path is highlighted in red.

Table: 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.

The ArchExplorer Approach: New DEG Formulation



The new DEG formulation is applied *w.r.t.* the code snippet as shown in Figure 4b. 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 $I(I_8) \rightarrow I(I_9)$.

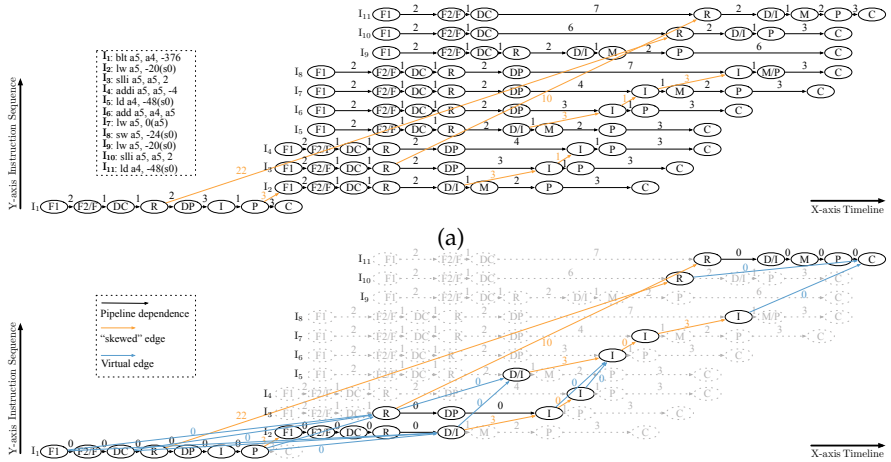
Induced DEG

A connected DEG consisting of horizontal, “skewed”, and virtual edges. The virtual edges are added to make the new DEG connected.

Rules for constructing Induced DEG:

- Connect nodes via time if the two nodes’ time are the closest.
- Connect nodes via instruction sequence if the two nodes’ instruction sequence are the closest.

The ArchExplorer Approach: Critical Path Construction



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

Algorithm Critical Path Construction

Require: G: The induced DEG with the edge cost;

node = `topological_sort`(G);
Initialize edge cost vector $\boldsymbol{d}$ with all zero;
Initialize the path vector $\boldsymbol{p}$ with all zero;
for $n \leftarrow$ node **do**
    if $\mathcal{N}_G(n) \neq \emptyset$ **then** ▷ $\mathcal{N}_G(n)$ are predecessors of n.
        $\boldsymbol{d}[n] = \arg \max_{v \in \mathcal{N}_G(n)} d[v] + \text{cost}$; assign $\boldsymbol{p}[n]$ with v;
    else
        $\boldsymbol{d}[n] = 0$; $\boldsymbol{p}[n] = n$;
    end if
end for
return `reverse`($\boldsymbol{p}$);

Computation of Bottleneck Contribution

For a critical path p with length L and containing N (non-overlapping) edges, a resource b's contribution $c(b)$:

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

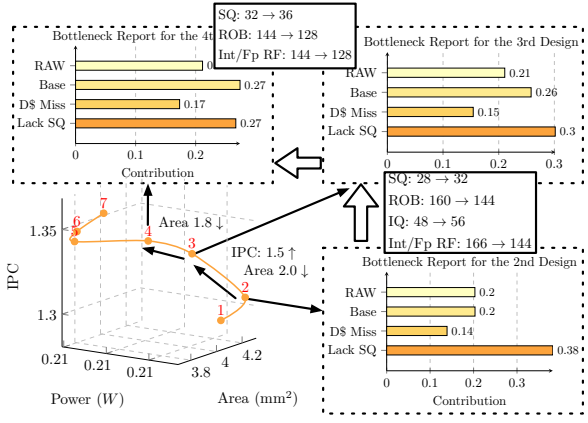
For multiple workloads evaluations:

$$\bar{c}(b) = \sum_{i=1}^{|B|} w_i \cdot c_i(b), \quad \sum_{i=1}^{|B|} w_i = 1, \tag{2}$$

The ArchExplorer Approach: Bottleneck-removal DSE

Resource reassignment:

- The reassigned parameter values are decided based on the design space specification.
- We select the next larger candidate value from the specification if we need to increase it.
- We decrease them to the next smaller candidate value if they do not have a contribution.



Experimental Setup & Evaluation Metrics

Experimental Setup: Simulation Environment

- GEM5: timing accurate simulator[14]
- McPAT: power & modeling tool[15]
- SPEC CPU2006 (SPEC06)[16] & SPEC CPU2017 (SPEC17)[17]
- Implement ArchExplorer w. Python & C++.

[14] Nathan Binkert et al. (2011). "The GEM5 Simulator". In: *ACM SIGARCH computer architecture news* 39.2, pp. 1–7; Jason Lowe-Power et al. (2020). "The GEM5 Simulator: Version 20.0+". In: *arXiv preprint arXiv:2007.03152*.

[15] Sheng Li et al. (2009). "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures". In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 469–480.

[16] *SPEC CPU 2006* (2018). https://www.spec.org/cpu2006/.

[17] *SPEC CPU 2017* (2022). https://www.spec.org/cpu2017/.

Experimental Setup: Baselines

- ArchRanker[18]
- AdaBoost[19]
- BOOM-Explorer[20]
- Calipers[21]

[18] Tianshi Chen et al. (2014). "ArchRanker: A Ranking Approach to Design Space Exploration". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE.

[19] Dandan Li et al. (2016). "Efficient Design Space Exploration Via Statistical Sampling and AdaBoost Learning". In: *ACM/IEEE Design Automation Conference (DAC)*. IEEE, pp. 1–6.

[20] Chen Bai et al. (2021). "BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, pp. 1–9.

[21] Hossein Golestani et al. (2022). "Calipers: A Criticality-aware Framework for Modeling Processor Performance". In: *ACM International Conference on Supercomputing (ICS)*.

Experimental Setup: Design Space

Table: Design space of an OoO RISC-V processor

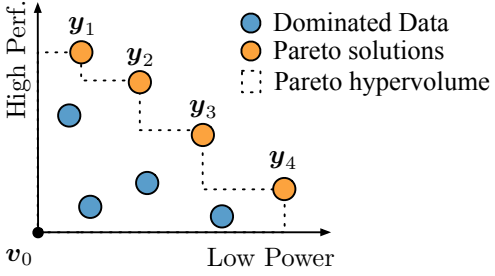
Components	Description	Hardware Resource	#
Pipeline width	fetch/decode/rename/dispatch/ issue/writeback commit width	1:8:1 [1]	8
Fetch buffer	fetch buffer size in bytes	16, 32, 64	3
Fetch queue	fetch queue size in μ-ops	8:48:4	11
Local predictor	local predictor size of the Tournament BP	512, 1024, 2048	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

Evaluation Metrics

Two main metrics:

- Pareto hypervolume.
- Number of simulations.

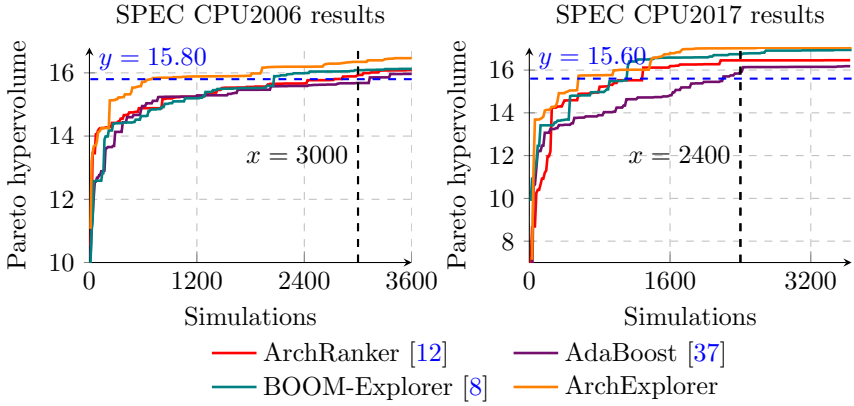


The visualization of Pareto hypervolume in Perf-Power space. Pareto hypervolume is the area bounded by $\mathcal{P}(\mathcal{Y}) = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3, \boldsymbol{y}_4\}$ and the reference point $\boldsymbol{v}_0$.

$$\mathrm{PV}_{\boldsymbol{v}_0}(\mathcal{P}(\mathcal{Y})) = \int_{\mathrm{Y}} \mathbb{1}[\boldsymbol{y} \succeq \boldsymbol{v}_0][1 - \prod_{\boldsymbol{y}_* \in \mathcal{P}(\mathcal{Y})} \mathbb{1}[\boldsymbol{y}_* \not\succeq \boldsymbol{y}]]\mathrm{d}\boldsymbol{y}, \tag{3}$$

Results

Results: Comparison w. DSE Methodologies



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

Results: Comparison w. DSE Methodologies

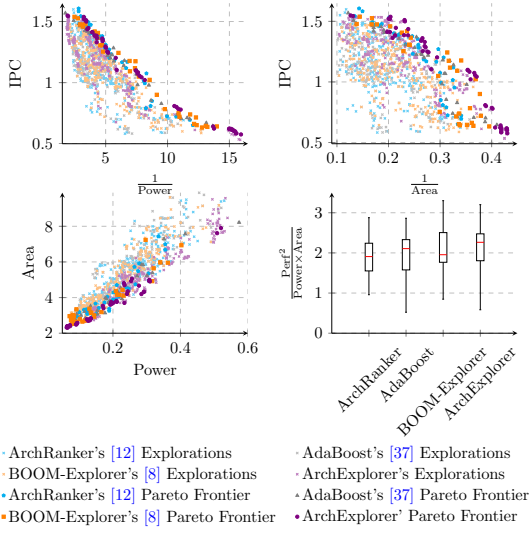
Table: Comparison under two cases.

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 [Chen et al. 2014]	2736	1	15.9185	1	1296	1	16.4542	1
AdaBoost [D. Li et al. 2016]	3132	1.1447	15.6785	0.9849	2208	0.7037	15.9359	0.9685
BOOM-Explorer [Bai et al. 2021]	2064	0.7544	16.0854	1.0104	1120	0.8642	16.7416	1.0175
ArchExplorer	708	0.2588	16.3473	1.0269	560	0.4321	17.0198	1.0344

Summary of comparison results w. DSE methodologies:

- In SPEC06, compared to ArchRanker, AdaBoost, and BOOM-Explorer, ArchExplorer uses 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, AdaBoost, and ArchRanker 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.

Results: Comparison w. DSE Methodologies

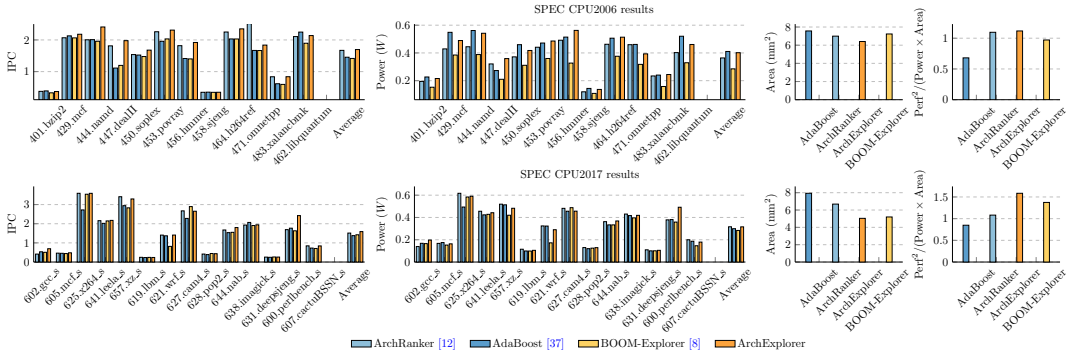


The visualization of Pareto frontiers and the distributions of PPA trade-offs for all methods.

Summary of visualization results:

- The visualization suggests that ArchExplorer outperforms other methods not by exploring more higher-performance microarchitectures but higher power and area efficiency designs.
- ArchExplorer's Pareto designs achieve an average of 2.26 in the trade-off, surpassing BOOM-Explorer, AdaBoost, and ArchRanker by 15.81%, 7.47%, and 18.63%, respectively.

Results: Comparison w. Best Balanced Designs



Comparisons between the Pareto designs in performance and power.

Summary of comparison results w. best balanced designs:

- 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 SPEC06.
- ArchExplorer's Pareto design is better than other methods by an average of 49.53% higher PPA trade-off in SPEC17.

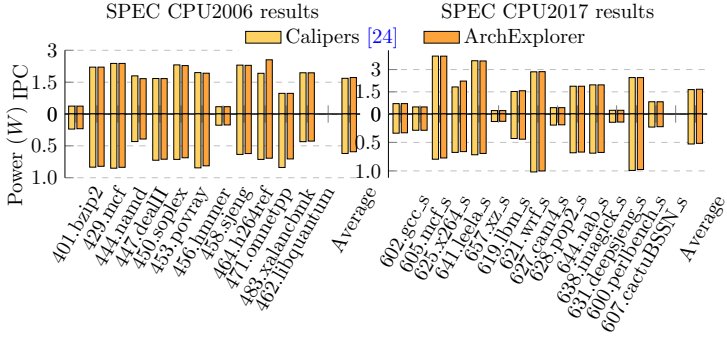
Results: Comparison w. Calipers

Experimental setup: A sub-design space including 1296 very similar designs.
Rationales:

- Calipers[22] only targets performance.
- Calipers does not provide how to search with the previous DEG formulation.

[22] Hossein Golestani et al. (2022). "Calipers: A Criticality-aware Framework for Modeling Processor Performance". In: *ACM International Conference on Supercomputing (ICS)*.

Results: Comparison w. Calipers



Comparisons w. Calipers **[D. Li et al. 2016]**.

Summary of comparison results w. Calipers:

- The solution found by ArchExplorer outperforms Caliper's by 2.11% in performance, 4.36% lower power and 2.38% lower area on average in SPEC06.
- In SPEC17, we receive a 1.88% higher performance compared to Calipers.
- These improved performance benefits are gained by only using 48 simulations in ArchExplorer.

Discussion

Discussion

The new DEG formulation can assist research in

- ML-assisted microprocessor performance modeling & enhanced DSE.
- Criticality-driven instruction scheduling.
- Program analysis & compiler research.

Multi-core formulation is also expected based on the new DEG formulation.
Codes repo: `https://github.com/baichen318/arch-explorer`

THANK YOU!

References I

Chen Bai et al. (2021). "BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, pp. 1–9.

Nathan Binkert et al. (2011). "The GEM5 Simulator". In: *ACM SIGARCH computer architecture news* 39.2, pp. 1–7.

Tianshi Chen et al. (2014). "ArchRanker: A Ranking Approach to Design Space Exploration". In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE.

Stijn Eyerman et al. (2009). "A Mechanistic Performance Model for Superscalar Out-of-order Processors". In: *ACM Transactions on Computer Systems (TOCS)* 27.2, pp. 1–37.

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, pp. 74–85.

Hossein Golestani et al. (2022). "Calipers: A Criticality-aware Framework for Modeling Processor Performance". In: *ACM International Conference on Supercomputing (ICS)*.

References II

Mark D Hill and Alan Jay Smith (1989). "Evaluating Associativity in CPU Caches". In: *IEEE Transactions on Computers* 38.12, pp. 1612–1630.

MS Hrishikesh et al. (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, pp. 14–24.

Engin Ïpek et al. (2006). "Efficiently Exploring Architectural Design Spaces Via Predictive Modeling". In: *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* 40.5, pp. 195–206.

Norman P. Jouppi (1989). "The Nonuniform Distribution of Instruction-level and Machine Parallelism and Its Effect on Performance". In: *IEEE Transactions on Computers* 38.12, pp. 1645–1658.

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)*, pp. 402–411.

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, pp. 340–351.

References III

Dandan Li et al. (2016). “Efficient Design Space Exploration Via Statistical Sampling and AdaBoost Learning”. In: *ACM/IEEE Design Automation Conference (DAC)*. IEEE, pp. 1–6.

Sheng Li et al. (2009). “McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures”. In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 469–480.

Jason Lowe-Power et al. (2020). “The GEM5 Simulator: Version 20.0+”. In: *arXiv preprint arXiv:2007.03152*.

Laurence J Peter, Raymond Hull, et al. (1969). *The Peter Principle*. Vol. 4. Souvenir Press London.

SPEC CPU 2006 (2018). https://www.spec.org/cpu2006/.

SPEC CPU 2017 (2022). https://www.spec.org/cpu2017/.

Guangyu Sun et al. (2011). “Moguls: A Model to Explore the Memory Hierarchy for Bandwidth Improvements”. In: *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 377–388.