ICCAD CAD Contest 2022

– "Microarchitecture Design Space Exploration"

Sicheng Li, Chen Bai, Xuechao Wei, Bizhao Shi,
Yen-Kuang Chen, Yuan Xie

Alibaba Group Inc.

Aug. 19, 2022

1 Revision History

- 2022/02/18 First version
- 2022/07/11 Second version: update the evaluation indicator, highlighted in yellow.
- 2022/07/30 Third version: update the score function, highlighted in green.

2 Introduction

Modern chip development requires high cost in design time and workforce. The reason for the expensive chip development cycle lies in two folds. On the one hand, the pre-defined performance, power, and area (PPA) targets of the chip are set aggressively, hoping to deliver the following generation product comparable and superior to market competitors. On the other hand, the design complexity of the chip ($\frac{\text{number of gates}}{\text{chip area}}$) is continuously increasing, leaving the improvement in design capability ($\frac{\text{number of gates}}{\text{staff} \times \text{month}}$) behind a considerable margin, causing an imbalance between the required design efforts and the cost of input.

Thanks to the agile design paradigm provided by advanced hardware description languages, *e.g.*, Chisel [1], and flexible and parameterized hardware generators, chip architects and engineers possess the capability to deliver a high-quality chip within a limited time budget.

To further enhance the chip design ability built on top of the agile development paradigm for industry, exploring a series of chips in a given design space to achieve different degrees of trade-offs *w.r.t.* performance, power, and area in a short time is necessary.

We look for effective and practical design space exploration algorithms to solve the problem. Specifically, in this contest problem, we focus on microarchitecture exploration of processors, *i.e.*, central processing units (CPU).

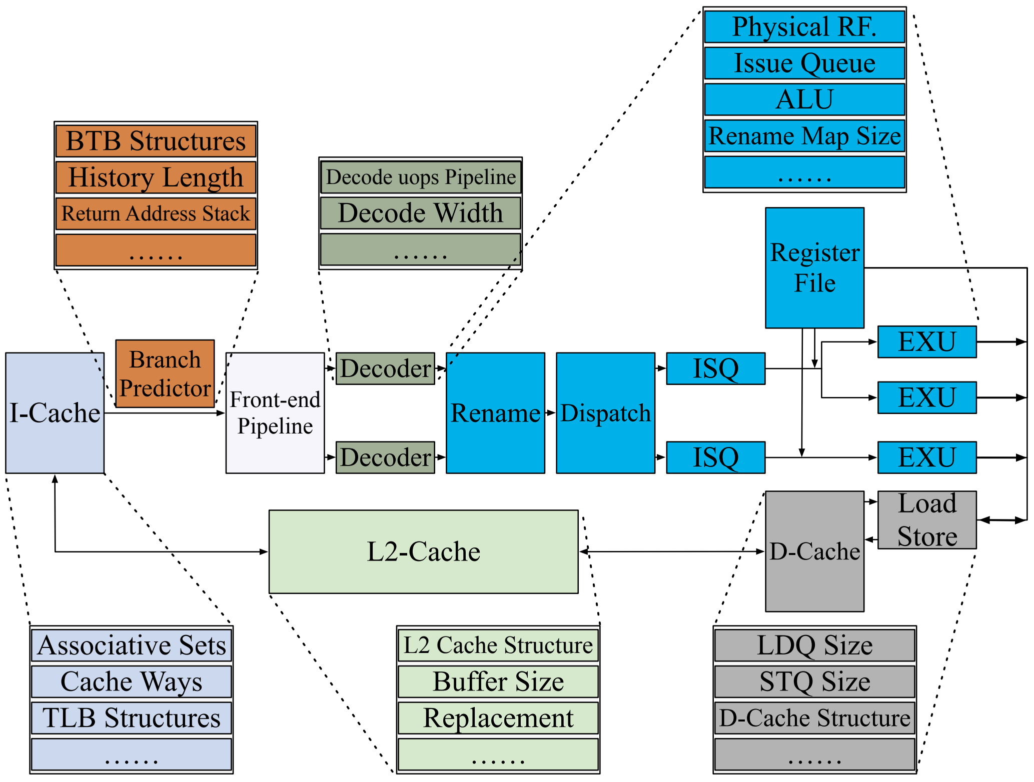


Figure 1: An example of a microarchitecture and its critical parameters.

2.1 Microarchitecture

Microarchitecture is an implementation of a given instruction set architecture (ISA). For example, it decides the detailed implementation of different boxes (modules) in processors, such as instruction fetch unit, decoding unit, scheduling and issuing unit, execution unit, load and store unit, *etc*. Inside each box, queues, buffers, stacks, caches, *etc*., corporate within different logics to perform pre-determined logic functions. Out-of-order processors mainly leverage a re-order buffer to trace every in-flight instruction in the pipeline. The entries of the re-order buffer needed are relevant to the resources of instruction issuing queues, load and store queues, *etc*. When a processor is assigned with abundant issuing resources and the load-store queues are highly optimized, we can enlarge entries of the re-order buffer to accommodate higher instruction throughputs. Hence, we can achieve different compromises in adjusting the resources and capacities of critical components to attain various balances between PPA design targets.

Figure 1 shows an example model of a processor. Instructions are fetched from I-Cache and packed and stored in a fetch queue. By recording and learning from the history, branch predictors predict the following instruction address given conditional jump instructions. Fetch queue decouples between the front-end and decoders. Micro-ops (Uops) are generated according to instruction. Conflicts in instructions are alleviated by register renaming, bypassing network, *etc*. Micro-ops trigger execution units to calculate with operands after being issued from issue queues (ISQ). Some memory-related instructions interact with D-Cache via the load and store unit. Across different boxes, parameters *w.r.t.* components of each box are extracted based on architects' prior knowledge. Some of the example parameters are also listed in Figure 1. With different combinations of each parameter, a new microarchitecture can be formulated.

A very-large-scale integration (VLSI) verification flow is required to estimate the microarchitecture's performance, power, and area under a technology with an accept fidelity. The VLSI verification flow incorporates different electronic design automation tools, *e.g.*, logic synthesis, physical design, simulation, power estimation, *etc.*, to give PPA values.

2.2 Design Space Exploration

In the early stage of the chip development cycle, deciding hardware resources and structures for these components is requisite due to their variate impacts on the performance, power, and area. In previous times, the method to select appropriate logic and hardware resources for these boxes or components mainly relies on the engineering experiences of architects. Nevertheless, it becomes increasingly difficult when processors integrate with more components, causing little prior knowledge transferred to solve it. Therefore, researchers tried different models, analytical or data-driven black-box models, to characterize a suitable parameter combinations [2–7]. As analytical methods become difficult to establish and it is failed to be promising in the compromise between workforce input and accuracy feedback, current solutions rely on machine-learning-driven design space exploration methodologies. Researchers have designed various sampling algorithms and models, *e.g.*, transductive experimental design, orthogonal array sampling, *etc.*, and adopted diverse black-box models, *e.g.*, ,linear regression with regularization, AdaBoost, Gaussian process, *etc.* A meaningful question is emerged, *i.e.*, which design space exploration algorithm is more practical and effective in solving the problem? Can we design a more robust and efficient space exploration algorithm that approaches the optimal microarchitecture or predicts the Pareto frontier with higher accuracy in the large design space?

2.3 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.

3 Problem Formulation

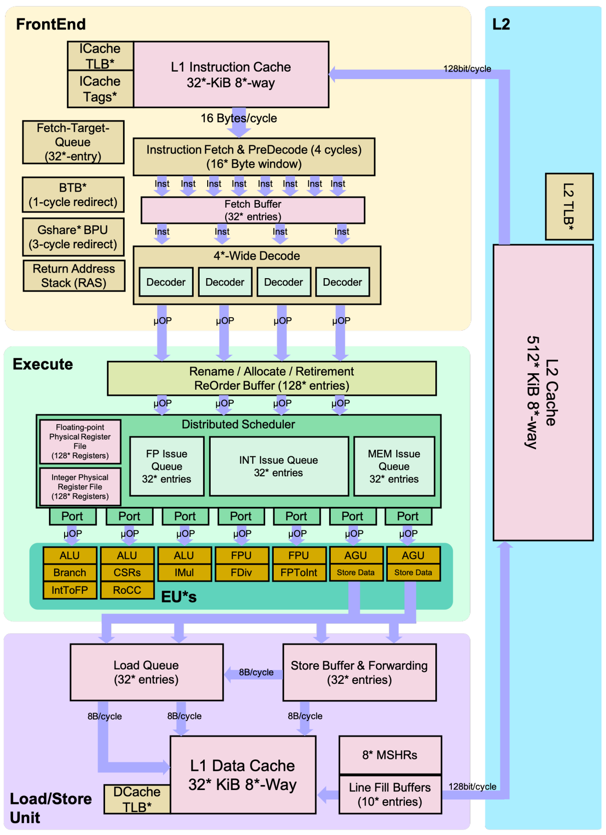
The contest problem is formulated as a microarchitecture design space exploration. Given a fixed design space, find the Pareto optimal set within a short time. The Pareto optimal set is mapped to the Pareto frontier in the PPA objective space.

3.1 Problem Description

For better understanding, we utilize an open-source RISC-V out-of-order core, Berkeley-Out-of-Order Machine (BOOM) [8–10], as an example.

Figure 2 shows an overview of the BOOM pipeline. It is a ten-stage pipeline design, fully compliant with RV64GC ISA. Following Section 2.1, we can extract a microarchitecture design space for the core, listed in Table 1.

A combination of parameters determines a microarchitecture implementation. Hence, we use a feature vector to define a microarchitecture embedding (the combination of parameters). Each dimension and element is the component, and selected candidate value, respectively. The encoding is a one-to-one mapping, *e.g.*, a possible microarchitecture embedding



FrontEnd
ICache TLB*
ICache Tags*
L1 Instruction Cache 32*-KiB 8*-way
128bit/cycle
16 Bytes/cycle
Fetch-Target-Queue (32*-entry)
Instruction Fetch & PreDecode (4 cycles) (16* Byte window)
BTB* (1-cycle redirect)
Inst
Fetch Buffer (32* entries)
Gshare* BPU (3-cycle redirect)
Return Address Stack (RAS)
4*-Wide Decode
Decoder
µOP
L2
L2 TLB*
L2 Cache 512* KiB 8*-way
Execute
Rename / Allocate / Retirement ReOrder Buffer (128* entries)
Distributed Scheduler
Floating-point Physical Register File (128* Registers)
Integer Physical Register File (128* Registers)
FP Issue Queue 32* entries
INT Issue Queue 32* entries
MEM Issue Queue 32* entries
Port
ALU
Branch
IntToFP
CSRs
RoCC
IMul
FPU
FDiv
FPToInt
AGU
Store Data
EU*s
Load Queue (32* entries)
8B/cycle
Store Buffer & Forwarding (32* entries)
L1 Data Cache 32* KiB 8*-Way
8* MSHRs
Line Fill Buffers (10* entries)
128bit/cycle
DCache TLB*
Load/Store Unit

Figure 2: An overview of BOOM pipeline

Table 1: Microarchitecture Design Space of BOOM

Module	Component	Descriptions	Candidate values
FrontEnd	FetchWidth	Number of instructions the fetch unit can retrieve once	4, 8
	FetchBufferEntry	Entries of the fetch buffer register	8, 16, 24, 32, 35, 40
	RasEntry	Entries of the Return Address Stack (RAS)	16, 24, 32
	BranchCount	Entries of the Branch Target Buffer (BTB)	8, 12, 16, 20
	ICacheWay	Associate sets of L1 I-Cache	2, 4, 8
	ICacheTLB	Entries of Table Look-aside Buffer (TLB) in L1 I-Cache	8, 16, 32
	ICacheFetchBytes	Unit of line capacity that L1 I-Cache supports	2, 4
IDU	DecodeWidth	Number of instructions the decoding unit can decode once	1, 2, 3, 4, 5
	RobEntry	Entries of the reorder buffer	32, 64, 96, 128, 130
	IntPhyRegister	Number of physical integer registers	48, 64, 80, 96, 112
	FpPhyRegister	Number of physical floating-point registers	48, 64, 80, 96, 112
EU	MemIssueWidth	Number of memory-related instructions that can issue once	1, 2
	IntIssueWidth	Number of integer-related instructions that can issue once	1, 2, 3, 4, 5
	FpIssueWidth	Number of floating-point-related instructions that can issue once	1, 2
LSU	LDQEntry	Entries of the Loading Queue (LDQ)	8, 16, 24, 32
	STQEntry	Entries of the Store Queue (STQ)	8, 16, 24, 32
	DCacheWay	Associate sets of L1 D-Cache	2, 4, 8
	DCacheMSHR	Entries of Miss Status Handling Register (MSHR)	2, 4, 8
	DCacheTLB	Entries of Table Look-aside Buffer (TLB) in L1 D-Cache	8, 16, 32

$(4, 16, 32, 12, 4, 8, 2, 2, 64, 80, 64, 1, 2, 1, 16, 16, 4, 2, 8)$ denotes the microarchitecture is a two-wide out-of-order core with 4-issue slots, 32-byte fetch width, 4-way I-Cache, 4-way D-Cache, *etc.* According to Table 1 and the design specifications of BOOM, the design space can be approximately 1.6×10^8. Each microarchitecture is evaluated with the same benchmark suite via the same VLSI verification flow to get golden performance, power, area values, and overall running time of the VLSI verification flow. Contestants need to develop a practical, efficient, and accurate microarchitecture design space exploration algorithm for the proposed microarchitecture design space. The algorithm should predict the Pareto optimal set, *i.e.*, a set of microarchitectures formulate the Pareto frontier in the PPA objective space.

3.2 Input and Output Format

We will provide a design space exploration algorithm benchmarking platform in this contest problem. The design space exploration algorithm benchmarking platform accepts contestants' submitted solutions and returns their scores according to our pre-defined score functions.

3.2.1 Input Format

We only release a subset of the design space to contestants, *e.g.*, one possible subset of Table 1. As Section 3.1 introduces, each microarchitecture embedding is a high-dimensional feature vector. Thus, the candidate values are known to contestants. Contestants then implement their design space exploration algorithms to predict the Pareto optimal set. Contestants can only access the PPA values of the subset of the design space and are ignorant of other combinations of parameters outside the design space. Moreover, they are restricted from implementing their algorithms within our provided application program interfaces (APIs).

The design space exploration algorithm benchmarking platform is based on Bayesmark [11], and we will provide an appropriate version for the contest.

Most black-box design space exploration algorithms include two critical steps, sample, and update. "Sample" is the step of selecting microarchitectures that need to be evaluated. "Update" is the step to update the black-box model with the already sampled dataset. Design space exploration algorithms iterate over these two steps from time to time and gradually find the Pareto optimal set with the explored dataset, *e.g.*, Bayesian optimization, simulated annealing, genetic algorithm, *etc.* Our provided benchmarking platform supports these two steps.

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

class YourDesignSpaceExplorationAlgorithm(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.

            returns
            -------
            next_guess: <list> of <list>
                list of `self.n_suggestions` suggestion(s).
                each suggestion is a microarchitecture embedding.
        """
        # do whatever is needed to get the parallel guesses
        # ...
        return x_guess

    def observe(self, X, y):
        """
            send an observation of a suggestion back to the optimizer.

            parameters
            ----------
            x: <list> of <list>
                the output of `suggest`.
            y: <list> of <list>
                corresponding values where each `x` is mapped to.
        """
        # update the model with new objective function observations
        # ...
        # no return statement needed

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(YourDesignSpaceExplorationAlgorithm)
```

The code snippet listed above demonstrates the input sample of the contest problem. The programming language is restricted to Python. Contestants are required to implement "suggest" and "observe" functions. The function "suggest" is a wrapper for "Sample" and "observe" is a wrapper for "Update". The benchmarking platform also supports automatic scripts for contestants to familiarize the design space exploration flow and automatic solution codes packing and submission. Random search will serve as a baseline example for contestants to get started quickly. Third-party Python packages are allowed to implement the algorithm, *e.g.*, basic operations with tensors, *etc*. If contestants leverage third-party Python packages, they should strictly follow submission instructions to make their solutions executed correctly in our servers.

3.2.2 Output Format

The benchmarking platform only returns scores of their submissions. Contestants can only access to dataset released to them. We leverage hidden datasets to evaluate their submissions.

All evaluations are executed on the same independent servers, hidden from all contestants. The requirement is necessary to make the contest fairer. The scores of submissions are calculated based on the Euclidean distance of PPA values mapped from the predicted Pareto optimal set and the golden Pareto optimal set. We will detail the evaluation metrics in Section 4. Each team can submit to their solutions no more than a fixed number of times a day. The rankings regarding scores of their submissions are released to all contestants timely.

3.2.3 Notice

The design space is vast, but it does not mean each microarchitecture in the design space is valid. Invalid microarchitectures are failed to simulate. We do not know whether a microarchitecture is invalid until we acquire the simulation reports in the VLSI verification flow. Therefore, the benchmarking platform will return a specified and reasonable number to the function "observe" shown in Section 3.2.1 and increase the running time spent evaluating such an invalid microarchitecture. It incentivizes contestants to design a model in their solutions to learn and circumvent these invalid microarchitectures, saving running time by not pushing them to the VLSI verification flow.

4 Evaluation

In this contest problem, we rank contestants' submissions based on two metrics, Pareto hypervolume difference and overall running time (ORT).

Pareto hypervolume, as illustrated in Equation (1), is the *Lebesgue measure* of the space dominated by the Pareto froniter and bounded by a reference point $\boldsymbol{v}_{\text{ref}}$ [7].

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

where $\mathbb{1}(\cdot)$ is the indicator function, which outputs 1 if its argument is true and 0 otherwise, $\mathcal{P}(\mathcal{Y})$ is the Pareto frontier.

Figure 3(a) illustrates an overview of the Pareto hypervolume. A better point will be closer to the origin point in the two-dimensional space ($f_1(x)$ and $f_2(x)$). Given the point colored in orange as the reference point, the points colored in purple are not dominated by any other, *e.g.*, purple points are not dominated by green. The Pareto hypervolume in Figure 3(a) is the area of the region colored in gray, *i.e.*, a convex polygon bounded by the reference point and purple points. In the contest problem, we deal with three-dimensional space, *i.e.*, performance, power, and area, as Figure 3(b) illustrates. Hence, the Pareto hypervolume is the volume bounded by the reference point (*e.g.*, the original point) and explored objective values, *i.e.*, $\boldsymbol{y}_1$, $\boldsymbol{y}_2$, and so on. The Pareto hypervolume difference is a difference between the golden Pareto hypervolume, formulated from the actual Pareto frontier in the design space, and the predicted Pareto hypervolume, formulated from the predicted Pareto frontier, as shown in Equation (2).

$$d_{\text{PVol}} = \text{PVol}_{\boldsymbol{v}_{\text{ref}}}(\mathcal{P}(\mathcal{Y}^*)) - \text{PVol}_{\text{v}_{\text{ref}}}(\mathcal{P}(\mathcal{Y})), \tag{2}$$

where $\mathcal{Y}^*$ is the set of golden objective values, and $\mathcal{Y}$ is the predicted objective values by the optimizer. If d_{PVol} is lower, the predicted Pareto hypervolume is larger, *i.e.*, the optimizer's performance is higher.

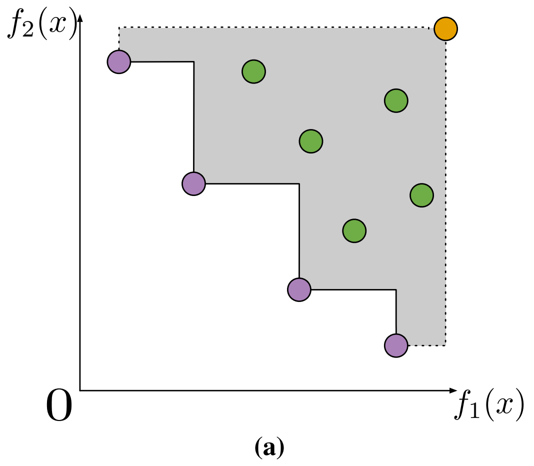


Figure 3: (a). An example overview of the Pareto hypervolume in the two dimensional space (b). An example overview of the Pareto hypervolume in the three dimensional space, *i.e.*, power, performance, and area.

Overall running time measures the total time of algorithms, including the submission of contestants' algorithms and the time spent on the VLSI verification flow. Since each microarchitecture has been pushed to the VLSI verification flow to get corresponding PPA values before evaluating contestants' submissions, we can access the time spent on the VLSI verification flow quickly with the dataset, *i.e.*, we do not push the predicted Pareto set to the VLSI verification flow.

The final scores are based on Pareto hypervolume and ORT, as Equation (3) shows.

$$\text{score} = \text{Pareto hypervolume} \cdot \begin{cases} \alpha - \dfrac{\text{ORT} - \theta}{\theta}, \text{ORT} \geq \theta \\ \alpha + |\dfrac{\text{ORT} - \theta}{\theta}|, \text{ORT} < \theta \end{cases}, \tag{3}$$

where α is an ORT score baseline, equal to 6, and θ is a pre-defined ORT budget, equivalent to 2625000. The constants are set to align with the de facto microarchitecture design space exploration flow. 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 has **lower Pareto hypervolume and ORT** as much as possible.

All evaluations are executed on the same independent servers, hidden from all contestants. Overall running times are calculated based on nearly the same environment, *e.g.*, servers' workload, *etc.*

5 Benchmark Suite

The benchmark suite includes the microarchitecture design space of several processors, corresponding PPA values, and time consumed in the VLSI verification flow. The processors are open-sourced RISC-V cores, *e.g.*, Rocket [12], BOOM [8–10] , *etc.*, hoping to strive for a research hotspot in the RISC-V community. Each microarchitecture is pushed to the same VLSI verification flow, *i.e.*, electronic design automation tools with the same version and a processing predictive kit (PDK), *e.g.* ASAP7 [13].

6 Discussion

Microarchitecture design space exploration is a potential method, highly coupled with the agile design paradigm. Due to the large design space and time-consuming VLSI verification flow, designing such an algorithm that can perform well on average for processors is difficult. For industry, it can save a considerable cost in the long-time chip development cycle with the help of microarchitecture design space exploration and deliver comparable products for customers. We expect a practical, efficient, and accurate microarchitecture design space exploration algorithm to further research in computer architecture and electronic design automation.

References

[1] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2012, pp. 1212–1221.

[2] E. İpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz, "Efficiently exploring architectural design spaces via predictive modeling," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 5, pp. 195–206, 2006.

[3] B. C. Lee and D. M. Brooks, "Illustrative design space studies with microarchitectural regression models," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2007, pp. 340–351.

[4] C. Dubach, T. Jones, and M. O'Boyle, "Microarchitectural design space exploration using an architecture-centric approach," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2007, pp. 262–271.

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

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

[7] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2021.

[8] K. Asanovic, D. A. Patterson, and C. Celio, "The berkeley out-of-order machine (BOOM): An industry-competitive, synthesizable, parameterized RISC-V processor," University of California at Berkeley, Tech. Rep., 2015.

[9] C. P. Celio, *A Highly Productive Implementation of an Out-of-Order Processor Generator*. eScholarship, University of California, 2017.

[10] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "SonicBOOM: The 3rd generation berkeley out-of-order machine," in *Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2020.

[11] "Bayes Opt Benchmark," https://bayesmark.readthedocs.io/en/latest/index.html.

[12] K. Asanović, R. Avizienis, J. Bachrach *et al.*, "The Rocket Chip Generator," University of California, Berkeley, Tech. Rep., 2016.

[13] V. Vashishtha, M. Vangala, and L. T. Clark, "ASAP7 predictive design kit development and cell design technology co-optimization," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 992–998.