ShanghaiTech University

EE215A VLSI Design Automation

Spring 2023

Project: Microarchitecture Design Space Exploration

Due date: 23:59, June 2, 2023

1 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 is continuously increasing, leaving the improvement in design capability 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, 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 problem, we focus on microarchitecture exploration of processors, i.e., central processing units (CPU).

1.1 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

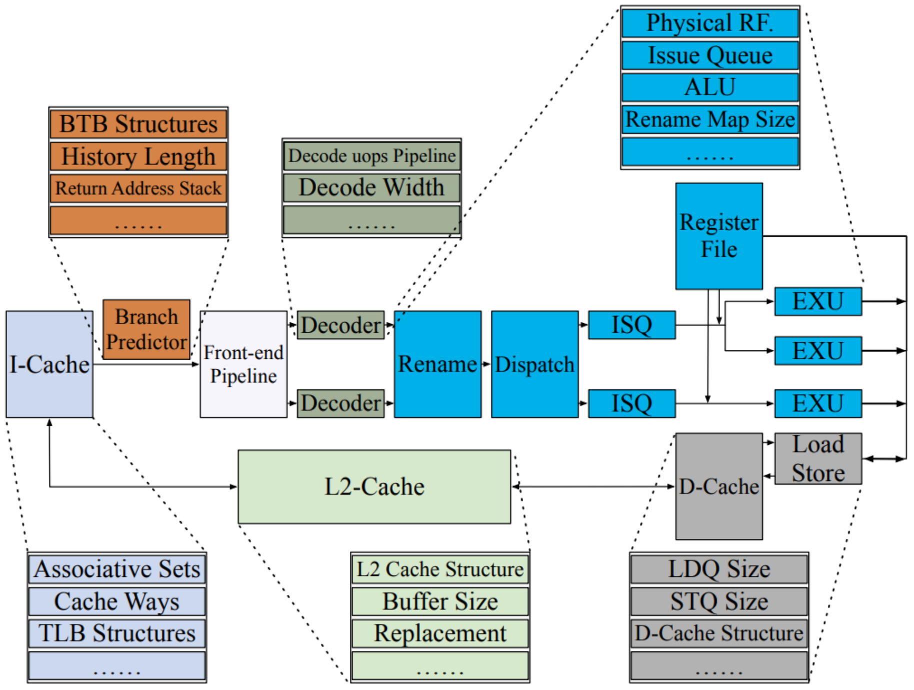


Figure 1: An example of a microarchitecture and its critical parameters.

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.

1.2 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. 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-learningdriven 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?

1.3 Project Objective

This problem aims to develop a practical, efficient, and accurate microarchitecture design space exploration algorithm. 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.

2 Problem Formulation

The 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.

2.1 Problem Description

For better understanding, we utilize an open-source RISC-V out-of-order core, Berkeley-Outof-Order Machine (BOOM), 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 1.1, 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 $(4, 16, 32, 12, 4, 8, 2, 2, 64, 80, 64, 1, 2, 1, 16, 16, 4, 2, 8)$ denotes the microarchitecture is a twowide 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. You 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.

2.2 Input and Output Format

We use, iccad-contest python package, a design space exploration algorithm benchmarking platform in this problem. The design space exploration algorithm benchmarking platform accepts your' submitted solutions and returns their scores according to our pre-defined score functions.

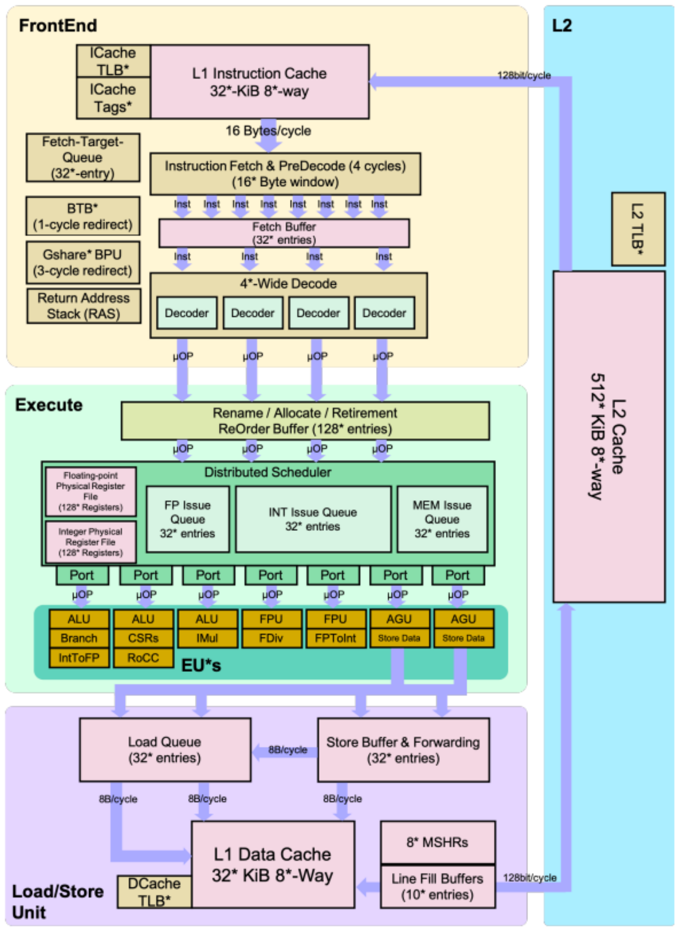
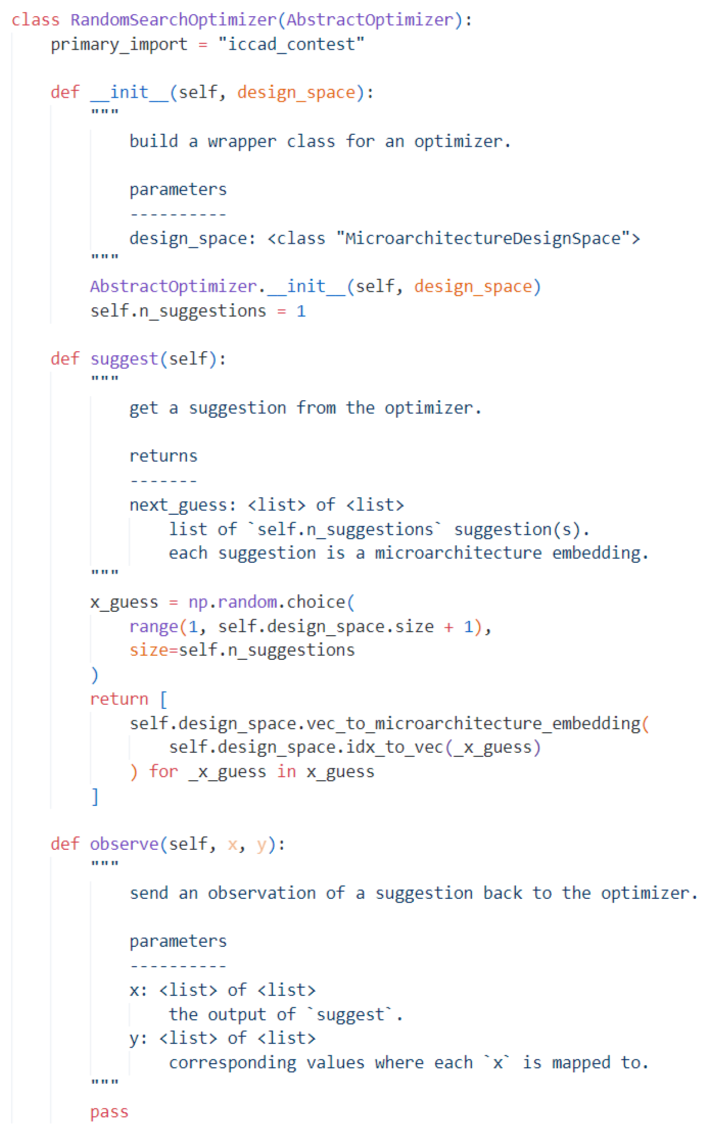

FrontEnd
ICache TLB*
ICache Tags*
L1 Instruction Cache 32*-KiB 8*-way
128bit/cycle
16 Bytes/cycle
Fetch-Target-Queue (32*-entry)
Instruction Fetch & PreDecode (4 cycles) (16* Byte window)
BTB* (1-cycle redirect)
Gshare* BPU (3-cycle redirect)
Return Address Stack (RAS)
Inst
Fetch Buffer (32* entries)
4*-Wide Decode
Decoder
μOP
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
ALU CSRs RoCC
ALU IMul
FPU FDiv
FPU FPToInt
AGU Store Data
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)
DCache TLB*
Load/Store Unit
L2
L2 TLB*
L2 Cache 512* KiB 8*-way

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



```python
class RandomSearchOptimizer(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, design_space)
        self.n_suggestions = 1

    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.
        """
        x_guess = np.random.choice(
            range(1, self.design_space.size + 1),
            size=self.n_suggestions
        )
        return [
            self.design_space.vec_to_microarchitecture_embedding(
                self.design_space.idx_to_vec(_x_guess)
            ) for _x_guess in 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.
        """
        pass
```

2.2.1 Input Format

We only use the design space from the iccad-contest python package, e.g., one possible subset of Table 1. Each microarchitecture embedding is a high-dimensional feature vector. Thus, the candidate values are known to yours. You then implement your design space exploration algorithms to predict the Pareto optimal set. You 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 code snippet listed above demonstrates the input sample of the problem. The programming language is restricted to Python. You 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 you to familiarize the design space exploration flow. Random search will serve as a baseline example for you to get started quickly. Third-party Python packages are allowed to implement the algorithm, e.g., basic operations with tensors, etc. If you leverage third-party Python

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

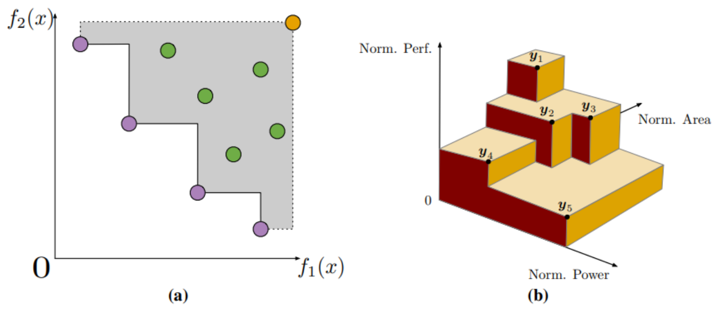


Figure 3: (a). An example overview of the Pareto hypervolume in the two dimensional space (b). An example overview of the Pareto hypervolume in the three dimensional space, i.e., power, performance, and area.

packages, you should create a requirements.txt to tell us your dependency.

2.2.2 Output Format

The benchmarking platform, iccad-contest package, will automatically cacluate the pareto hyper-volume and total runtime for us. We will use these two parameters to evaluate your DSE algorithm.

2.2.3 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 2.2.1 and increase the running time spent evaluating such an invalid microarchitecture. It incentivizes you 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.

2.3 Evaluation

In this problem, we rank your submissions based on two metrics, **pareto hypervolume** 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 v_{ref} where $\mathbb{I}()$ is the indicator function, which outputs 1 if its argument is true and 0 otherwise, $P(Y)$ 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 project 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., y_1, y_2, and so on.

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

Overall running time measures the total time of algorithms, including 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 your 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 the score equation shows: where α is an ORT score baseline, equal to 6, and θ is a pre-defined ORT budget, equivalent to 2625000. You 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 higher Pareto hypervolume and lower ORT as much as possible.

3 Submission Details

You should submit following files, so we can evaluate the correctness and quality of your result.

- An individual file namely "num-query.txt" to tell us the number of queries you want to perform in your algorithm.
- The source code and a "requirements.txt" file to list the libraries your program depends on.
- A sample run of you method, put the log generated by the iccad-contest package to the "out" folder, we use this result to check whether we correctly run your program, do not depends on the UUID random number, we will always use other random number to run your program.
- A README.md to tell us how to run your program.
- A report to tell us what you have done, some implementation details , etc.
- All the source code/num-query.txt/README.md should be put into the "src" folder, all output files should be put into the "out" folder, the report should be put into "rpt" folder.

4 Resources

This is project is strongly based on the ICCAD22 Contest problem C, except we restrict us to the dataset publicly available at the iccad-contest python package.

- To check out the original problem description, please check out this document, it has more reference papers that you can read.
- To prepare your development env, please check out this documentation: ICCAD Contest Platform
- To have a initial random search algorithm implementation, please check out this example.

We also include an offline-version of the iccad-contest package source code, in case you want inspect code in this package and the pdf files mentioned above under the "resource" folder of the project problem zip file.