2022 ICCAD CAD Contest Problem C: Microarchitecture Design Space Exploration

Invited Paper

Sicheng Li[1,†], Chen Bai[1,2,†], Xuechao Wei[1], Bizhao Shi[1], Yen-Kuang Chen[1], Yuan Xie[1]
[1]Alibaba Group [2]The Chinese University of Hong Kong

***Abstract*—It is vital to select microarchitectures to achieve good trade-offs between performance, power, and area in the chip development cycle. Combining high-level hardware description languages and optimization of electronic design automation tools empowers microarchitecture exploration at the circuit level. Due to the extremely large design space and high runtime cost to evaluate a microarchitecture, ICCAD 2022 CAD Contest Problem C calls for an effective design space exploration algorithm to solve the problem. We formulate the research topic as a contest problem and provide benchmark suites, contest benchmark platforms, *etc.*, for all contestants to innovate and estimate their algorithms.**

***Index Terms*—Microarchitecture, Design Space Exploration**

I. 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. 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% [1]. 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. We can conduct microarchitecture design space exploration at the circuit level via electronic design automation tools following the agile chip design paradigm. 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 two authors contributed equally to this work.

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA
© 2022 Copyright is held by the owner/author(s).
ACM ISBN 978-1-4503-9217-4/22/10.
https://doi.org/10.1145/3508352.3561109

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. 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 [2] or data-driven black-box models [3]–[6].

We formulate the research topic as a contest problem and deliver a data set based on RISC-V BOOM [7] and a contest benchmarking platform for all contestants. Contestants can innovate and estimate their algorithms using the data set 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.

II. Background

RISC-V is an open standard instruction set architecture (ISA), receiving great attention and support from academia and industry nowadays, *e.g.*, the out-of-order core BOOM [7] from UC Berkeley, Xuantie-910 [8] from the T-Head, Alibaba group *etc.* Microarchitecture defines an implementation of a processor *w.r.t.* a giving ISA. The implementation includes many boxes inside a processor, *e.g.*, instruction fetch unit, decoder, execution unit, etc. These boxes' structure and hardware resources have many alternatives, e.g., the decoder width, reorder buffer entries, return address stack size, load-/store queue size, cache structures, etc. Different structures and hardware resources *w.r.t.* these boxes formulate many microarchitectures. Microarchitectures can be easily parameterized and generated with high-level hardware description language, *e.g.*, Chisel.

Exploring good microarchitectures achieving good trade-offs among PPA metric values for a pre-determined design space within a limited time budget is an important research topic. We illustrate the preliminaries on RISC-V BOOM and the multi-objective design space exploration in this section.

A. RISC-V BOOM

Fig. 1 details the architecture of RISC-V BOOM. The fetch unit fetches instructions from I-cache and sends them to

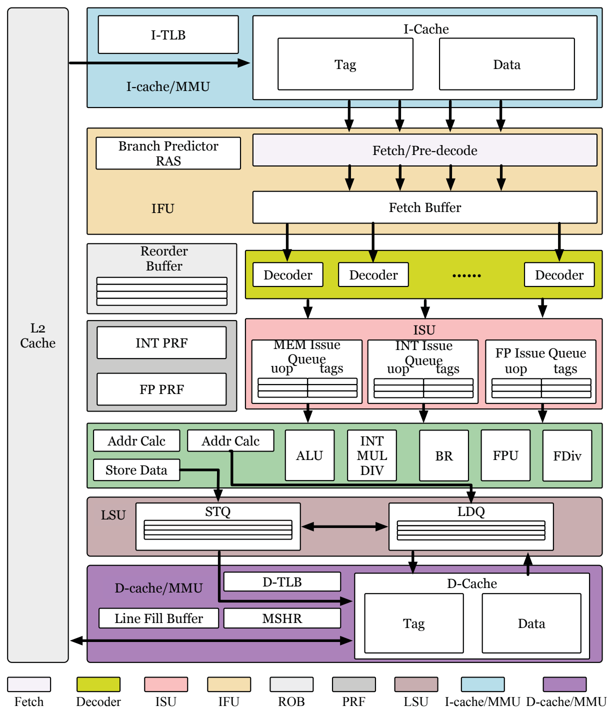


Fig. 1 The architecture of RISC-V BOOM

decoders. Decoders decode instructions as a set of micro-ops and dispatch them to the issue unit responsible for scheduling and issuing micro-ops. Simultaneously, the issue unit resolves dependencies and conflicts embedded in instructions. The load store unit (LSU) decouples the execution units and D-cache operations, delivering high instruction throughput. Each box may have several parameters to be decided, *e.g.*, the issue unit contains dispatch widths, issue widths, and issue queue entries for integer-related instructions, floating-point instructions, and memory-related instructions. We can leverage expert knowledge from CPU architects to prune the design space, *i.e.*, leaving solutions that are hard to be analyzed for design space exploration.

B. Multi-objective Design Space Exploration

We are considering exploring solutions for multiple correlated objectives, *i.e.*, performance, power, and area. In most cases, more hardware resources bring higher performance but dissipate more power and sacrifice a larger area. However, a better microarchitecture has higher performance, lower power dissipation, and smaller area. To compare whether a microarchitecture is better than another in multi-objective settings, we define the Pareto optimality.

Definition 1 (Pareto-optimal Set)**.** *For an n-objective maximization problem, a vector of objective values $\boldsymbol{y}'$ is said to dominate $\boldsymbol{y}$ if*

$$\begin{aligned} &\forall i \in [1, n], \quad \boldsymbol{y}' \geq \boldsymbol{y}; \\ &\exists j \in [1, n], \quad \boldsymbol{y}' > \boldsymbol{y}, \end{aligned} \tag{1}$$

Hence, we denote $\boldsymbol{y}' \succeq \boldsymbol{y}$. Otherwise, $\boldsymbol{y}' \not\succeq \boldsymbol{y}$. Given $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_n\}$, the Pareto-optimal set is to be defined as $\mathcal{P}(Y) = \{\boldsymbol{y}_i \in Y \mid \boldsymbol{y}_j \not\succeq \boldsymbol{y}_i, \forall j \in Y \setminus \{\boldsymbol{y}_i\}\}$.

We can specify the relations between any two objective values according to Def. 1.

III. 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.

IV. Problem Formulation

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

Based on Def. 1, our problem is formulated, as shown below.

Problem 1 (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.*

V. 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.

A. Microarchitecture Design Space

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 I.

Each number in TABLE I is an index, which specifies the detailed parameters, as listed in TABLE II and TABLE III. For example, if the ISU chooses an number 2 in TABLE I, 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 I, *i.e.*, $15,633$ RISC-V BOOM microarchitectures are created.

B. 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, formulated via $CONCAT$(FetchWidth, DecoderWidth, ..., D-TLBWays), where FetchWidth, DecoderWidth, *etc.*, are values obtained from TABLE II and TABLE III *w.r.t.* corresponding columns.

TABLE I 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	15633
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	
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	

TABLE II 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 III 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.

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.

VI. Evaluation

This section introduces the contest benchmarking platform and evaluation metrics for contestants' submissions.

A. Contest Benchmarking Platform

We provide a contest benchmarking platform for the problem to facilitate convenient implementation of the solver without considering the cumbersome handling procedure of the input and output. The contest 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".

Fig. 2 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

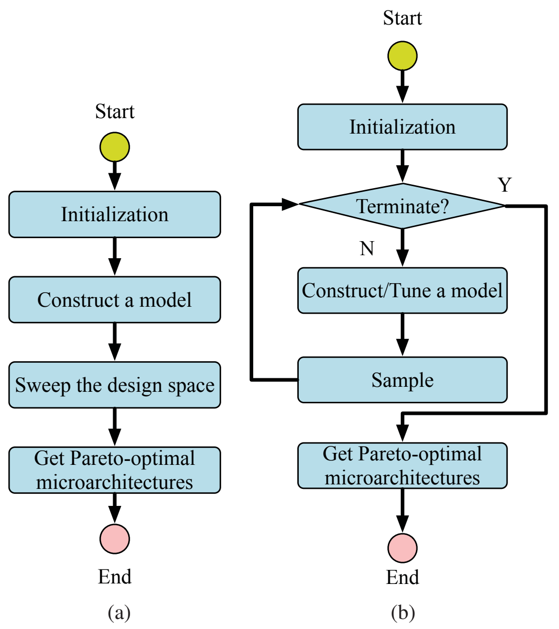


Fig. 2 (*a*) Offline design space exploration flow. (*b*) Online design space exploration flow.

space and retrieve the target microarchitectures, as shown in Fig. 2(a). 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 Fig. 2(b), 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.

1) *Solution Implementation*

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

Listing 1 Template of the Solution Implementation

```
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.

          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(YourAlgorithm)
```

Listing 2 Data structure Definition of the Design Space

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

components_mappings = {
  "Fetch": {
    "description": ["FetchWidth"],
    "1": [4]
  },
  "Decoder": {
    "description": ["DecodeWidth"],
    "1": [1]
  },
  "ISU": {
    "description": [
      "MEM_INST.DispatchWidth",
          "MEM_INST.IssueWidth"
      "MEM_INST.NumEntries",
          "INT_INST.DispatchWidth",
      "INT_INST.IssueWidth",
          "INT_INST.NumEntries",
      "FP_INST.DispatchWidth",
          "FP_INST.IssueWidth",
      "FP_INST.NumEntries"
    ],
    "1": [1, 1, 8, 1, 1, 8, 1, 1, 8],
    "2": [1, 1, 6, 1, 1, 6, 1, 1, 6],
    "3": [1, 1, 10, 1, 1, 12, 1, 1, 12]
  },
  "IFU": {
    "description": ["BranchTag",
        "FetchBufferEntries",
        "FetchTargetQueue"]
    "1": [8, 8, 16],
    "2": [6, 6, 14],
    "3": [10, 12, 20]
  },
  ...
}
```

Listing 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 V-B. 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.

Listing 3 Benchmarking Platform Help Menu

```
$ 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
```

Listing 4 Example Command of Benchmarking Platform

```
$ 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.
...
```

2) *API for Microarchitecture 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 V-A.

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

3) *Platform 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 3

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

B. *Evaluation Metrics*

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

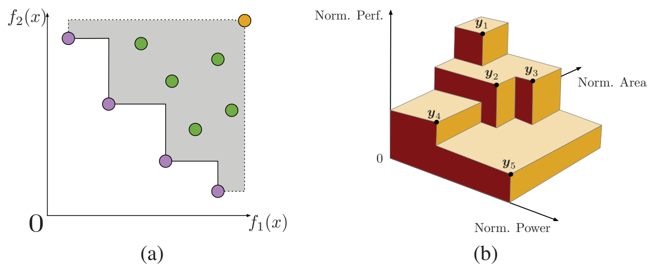


Fig. 3 (a). An example overview of the Pareto hypervolume in the two dimensional space (b). An example overview of the Pareto hypervolume in the three dimensional space, *i.e.*, power, performance, and area.

1) *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 $\boldsymbol{v}_{\text{ref}}$ [6], as shown in Equation (2).

$$\text{PVol}_{\boldsymbol{v}_{\text{ref}}}(\mathcal{P}(Y)) = \int_Y \mathbb{1}[\boldsymbol{y} \nsucceq \boldsymbol{v}_{\text{ref}}][1 - \prod_{\boldsymbol{y}_* \in \mathcal{P}(Y)} \mathbb{1}[\boldsymbol{y}_* \nsucceq \boldsymbol{y}]]\mathrm{d}\boldsymbol{y}, \tag{2}$$

where $\mathbb{1}(\cdot)$ is the indicator function, which outputs 1 if its argument is true and 0 otherwise. The integral characterized by Equation (2) sums up all regions bounded by $\mathcal{P}(Y)$ and $\boldsymbol{v}_{\text{ref}}$.

Fig. 3(a) 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 hypervolume. Intuitively, if a new point is searched out and not dominated by any points, $\text{PVol}_{\boldsymbol{v}_{\text{ref}}}(\mathcal{P}(\boldsymbol{Y} \cup \{\boldsymbol{y}_{\text{new}}\}))$ 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. Fig. 3(b) 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.

2) *Overall Running Time*

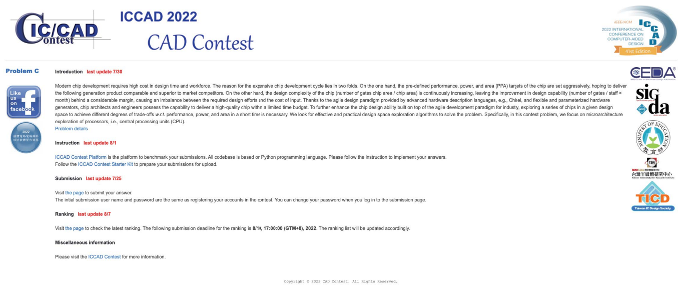
Overall running time (ORT) measures the total time of algorithms, including the submission of contestants' algorithms and the time spent on the VLSI flow.

3) *Total Score*

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

$$\text{score} = \text{PVol}_{\boldsymbol{v}_{\text{ref}}} \cdot \begin{cases} \alpha - \dfrac{\text{ORT} - \theta}{\theta}, \text{ORT} \geq \theta \\ \alpha + |\dfrac{\text{ORT} - \theta}{\theta}|, \text{ORT} < \theta \end{cases}, \tag{3}$$

where α is an ORT score baseline, equal to 6, and θ is a pre-defined ORT budget, equivalent to 2625000. 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.



Introduction last update 7/30

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 (number of gates chip area / chip area) is continuously increasing, leaving the improvement in design capability (number of gates / staff × 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, 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 his contest problem, we focus on microarchitecture exploration of processors, i.e., central processing units (CPU).
Problem details

Instruction last update 8/1

ICCAD Contest Platform is the platform to benchmark your submissions. All codebase is based or Python programming language. Please follow the instruction to implement your answers.
Follow the ICCAD Contest Starter Kit to prepare your submissions for upload.

Submission last update 7/25

Visit the page to submit your answer.
The intial submission user name and password are the same as registering your accounts in the contest. You can change your password when you log in to the submission page.

Ranking last update 8/7

Visit the page to check the latest ranking. The following submission deadline for the ranking is **8/10, 17:00:00 (GTM+8), 2022**. The ranking list will be updated accordingly.

Miscellaneous information

Please visit the ICCAD Contest for more information.

Copyright © 2022 CAD Contest. All Rights Reserved.

Fig. 4 An overview of the online ranking platform.

C. *Online Ranking Platform*

We provide contestants with an online ranking platform since the dataset discussed in Section V-B is not disclosed to the public. The ranking platform, as shown in Fig. 4, 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.

VII. Conclusions

Microarchitecture design space exploration is an important research topic. We formulate the topic as a 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.

Acknowledgment

The authors would like to thank Prof. Yuzhe Ma, from the Microelectronics Thrust, The Hong Kong University of Science and Technology (Guangzhou), and Prof. Bei Yu, from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong SAR. They suggested valuable ideas and helped to review the problem.

References

[1] "SEMICO Research Corp. SoC Silicon and Software 2018 Design Cost Analysis: How Rising Costs Impact SoC Design Starts," https://semico.com/content/soc-silicon-and-software-2018-design-cost-analysis-how-rising-costs-impact-soc-design-starts, 2018.

[2] 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.

[3] E. Ïpek, 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.

[4] 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.

[5] 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.

[6] 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 *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[7] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "SonicBOOM: The 3rd Generation Berkeley Out-of-order Machine," in *Fourth Workshop on Computer Architecture Research with RISC-V*, vol. 5, 2020.

[8] C. Chen, X. Xiang, C. Liu, Y. Shang, R. Guo, D. Liu, Y. Lu, Z. Hao, J. Luo, Z. Chen *et al.*, "Xuantie-910: A Commercial Multi-core 12-stage Pipeline Out-of-Order 64-bit High Performance RISC-V Processor with Vector Extension: Industrial product," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 52–64.