Microelectronics Journal 132 (2023) 105679

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

SPARK: An automatic Score-Power-Area efficient RISC-V processor microarchitecture SeeKer

Qiang Li, Jun Tao, Jun Han [*]

State Key Laboratory of ASIC and System, Fudan University, Shanghai 200433, China

ARTICLE INFO

Keywords:
Multi-objective optimization
Design space exploration
Bayesian optimization

ABSTRACT

In this study, we provide an automatic multi-objective optimization framework for RISC-V processor microarchitecture. CoreMark benchmark and TSMC 28 nm CMOS process serve as the foundation for SPARK's investigation of the design space to SonicBOOM for three design criteria of performance, power, and area. The sequential-BOED method demonstrates a convergence speed of ADRS 2.125 times faster than baseline thanks to the benefits of the suggested sampling algorithm RED. In the meantime, the SPARK framework's SPA-Gen infrastructure parallelizes querying the VLSI flow of elite trials. Therefore, under the same convergence target of ADRS as sequential-BOED, the overall running time of Para-BOED algorithm can be further improved by a factor of 1.29. The official Two-Wide BOOM achieves a commendable compromise between performance score, power consumption, and area cost. SPARK framework, however, finds an optimal microarchitecture design of BOOM with improved performance-cost ratio compared to official Two-Wide BOOM within fully acceptable searching time.

1. Introduction

The open-source RISC-V instruction set architecture (ISA) has recently attracted a lot of interest and active support from both academics and business. The instruction set architecture is implemented in hardware by the RISC-V processor microarchitecture. Popular RISC-V processor generator SonicBOOM [1] from Berkeley is a superscalar Out-of-Order processor, which fully supports the RV64GC instruction standard and is particularly competitive with other processors in terms of performance and power consumption. Chisel [2], widely known as an agile hardware construct language, is utilized by BOOM designers to develop a high-performance RISC-V microarchitecture. BOOM includes an effective hardware parameterization interface and smart negotiation features, which enable designers to explore the design space of processor microarchitecture. Notably, the BOOM generator and related SoC resources additionally show accurate microarchitectural hardware behavior in comparison to platforms that provide open source software models for high-performance cores, such as gem5 [3], MARSSx86 [4], Sniper [5], or ZSim [6].

There are numerous parameter combinations for internal components of processor, which represent different configurations of hardware implementation. In fact, different parameter selections have distinct impact on the design metrics of chip under same technology process, such as performance, power consumption, and area cost(PPA) [7]. The optimal microarchitecture parameter design is typically believed to achieve a superb balance between PPA for considering energy efficiency. The study of microarchitecture optimization, which primarily comprises two barriers, is challenging to manage. First, there are tens of millions or even hundreds of millions of possible combinations in the immense design space of microarchitecture. It is necessary to take into account processor elements including instruction buffer, issue queues, vector execution units, and reorder buffer. Designers are therefore unable to explore and measure every microarchitecture design. Second, it takes a significant amount of time and hardware resources to simulate the processor microarchitecture with large-scale benchmark and execute VLSI flow to obtain performance measurement. Obviously, the above costs are restricted by hardware, software, and EDA tools support.

Traditional industrial solutions have placed a strong focus on prior engineering expertise from processor architects. Clearly, the proliferation of new processor is hard to be supported by reliance on specialists. In academia, researchers have offered a variety of approaches that can be broadly split into two groups to tackle the aforementioned challenges. First, build analytical models to map relationships from design parameter space to design metrics space, and then apply the model to predict metrics of new unseen designs. For the aim of predicting performance of designs in the design space. ANN and regression models were presented in [8–10], respectively, to train a model mapping relationship from parameter configuration to design performance.

* Corresponding author.
E-mail address: junhan@fudan.edu.cn (J. Han).

https://doi.org/10.1016/j.mejo.2022.105679
Received 14 November 2022; Received in revised form 8 December 2022; Accepted 26 December 2022
Available online 3 January 2023

However, due to the high dimensionality and immensity of the design space, researchers must employ statistical sampling together with prediction models to characterize the microarchitectural design space by sampling as few samples as feasible. An active learning-based AdaBoost model and an orthogonal array-based sampling method are suggested as a combination for exploring the design space of an Out-of-Order processor [11]. To further reduce simulation times, Bai [12]introduced the BOOM-Explorer algorithm with embedded microarchitecture prior knowledge for IPC and power tradeoff, finding a better microarchitecture design than the official Two-Wide BOOM. Secondly, in order to search more designs within a constrained time budget, researchers frequently adopt coarse-grained simulation tools rather than register-transfer-level (RTL) workflows to speed up the procedure of collecting performance metrics [13,14].

Recent techniques do, however, have some drawbacks. First, prior research has scarcely proposed appropriate sampling techniques to lessen the complexity of design space exploration(DSE), ignoring the representativeness and edginess of designs in design space. On the other hand, software modeling platforms and coarse-grained simulation tools are still employed to speed up the simulation. Unfortunately, the majority of them execute workflow at the penalty of accuracy loss and inconsistent results between the simulation and the processor's actual behavior, which can lead to inferior simulation results. More critically, even the widely used system-level processor simulators like Gem5 [3] find it challenging to estimate and analyze the power consumption of modern CPUs at architectural level. Third, Bayesian optimization(BO) of processor DSE demands attempts of parallel acceleration in terms of the effectiveness of new sample's PPA collection in each loop. Finally, a comprehensive automatic processor DSE framework is desired for the ultimate objectives, PPA in the chip design flow. Overall, due to the aforementioned drawbacks, more in-depth academic research is encouraged to discover optimization approach on processor microarchitecture.

To fulfill the aforementioned expectations, we present Score-Power-Area efficient RISC-V processor Seeker, an automatic DSE framework to processor microarchitecture based on Bayesian optimization(SPARK). Our main contributions are highlighted below:

1. Algorithm Level: The RED algorithm is suggested to obtain representative designs and edge designs, which are then combined to create the initial training set together. The convergence rate of Pareto front is greatly accelerated by this method. The SPA-Gen infrastructure is set up to gather measurements of mini-batched trials concurrently. Our Para-BOED approach can effectively reduce the overall running time while maintaining the same convergence target of Pareto front as baseline.
2. Framework Level: We present an automatic multi-objective DSE framework for processor microarchitecture. The BOOM processor serves as the experimental object to validate the workflow. Notably, SPARK discovery an optimal microarchitecture design with performance-cost ratio as evaluation criteria.
3. True commercial foundation: The experiment is built on the TSMC 28 nm CMOS process library, reliable CoreMark, and industry-standard EDA tools. These foundation shows that SPARK is more like a real-world application scenario.

The rest of this paper is organized as follows. Section 2 analyzes BOOM design space and its corresponding configurable parameters, refines their optional values and discusses their constraints. In Section 3, we model the combinatorial optimization problem and present in detail the elements of SPARK framework, as well as the sampling advantages of the proposed RED algorithm and the algorithmic flow of Para-BOED. Section 4 provides an application of the SPARK framework to the DSE problem of BOOM and Section 5 draws a conclusion.

Table 1
Microarchitecture design parameters of BOOM.

Module	Parameter	Choices
FrontEnd	FetchWidth	4, 8
	FetchBufferEntry	8, 16, 24, 32, 35, 40
	RasEntry	16, 24, 32
	BranchCount	8, 12, 16, 20
	ICacheWay	2, 4, 8
	ICacheTLB	8, 16, 32
	ICacheFetchBytes	2, 4
Decode Unit	DecodeWidth	1, 2, 3, 4, 5
	RobEntry	32, 64, 96, 128, 130
	IntPhyRegister	48, 64, 80, 96, 112
	FpPhyRegister	48, 64, 80, 96, 112
Issue Unit	MemIssueWidth	1, 2
	IntIssueWidth	1, 2, 3, 4, 5
	FpIssueWidth	1, 2
Load Store Unit	LDQEntry	8, 16, 24, 32
	STQEntry	8, 16, 24, 32
	DCacheWay	2, 4, 8
	DCacheMSHR	2, 4, 8
	DCacheTLB	8, 16, 32
Total	14bn	

2. Constraints and pruning

The microarchitecture of BOOM processor is chosen as the subject of study to evaluate the proposed SPARK framework in light of its representativeness, developer-friendliness, and configurability in the open-source hardware community. Front-end, decoder unit, issue unit, and load-store unit compose the majority of BOOM core. We concentrate on the four modules listed in Table 1 and describe each of their customizable parameters, which are in accordance with the choices made in BOOM-Explorer(BE) [12]. That allows meaningful comparisons with previous work. The first column shows the main modules of BOOM. In the second column, the parameters of each module are described, while in the third column, the range of parameter values is indicated. For example, the load queue (LDQ) has a minimum size of 8 entries and a maximum size of 32 entries varied in steps of 8, meaning 4 different designs. The last row gives the number of designs in design space of BOOM.

The most popular BOOM configurations are the classic SmallBoomConfig, MediumBoomConfig, LargeBoomConfig, and MegaBoomConfig, which differ greatly in each design parameter and consequently in each of their three performance metrics. Designers can choose from a variety of combinations of each parameter to generate quite different RTL. As a result, designers face a very large decision space since any random combination of all the parameter selections will compose a design space with 13,996,800,000 individuals. However, it is obvious that not all configurations are valid and acceptable combinations when taking into account the constraints between various parameters. In this paper, these parameters are subject to the necessary limitations in Table 2. The first 5 of them are general rules for processor design, while next 6–8 constraints are the same setting as BE's that we adhere to in order to narrow the design space. The last one is included since we noticed that the generated RTL fail to execute CoreMark program in the scenario that was violated. Judging by past design conventions, limiting IssueWidth to less or equal than DecodeWidth is reasonable for pipeline execution of Out-of-Order processor. The size of design space after constraint pruning to deconstruct is 35575200. There is no doubt that the processor designer remains unconvinced this quantity can be reached.

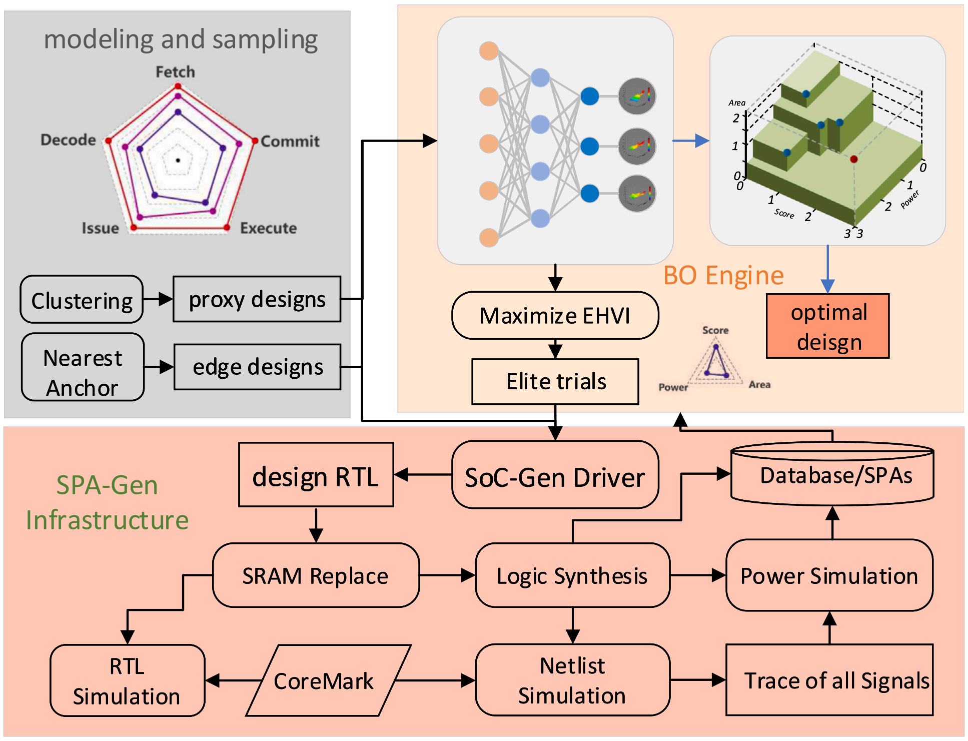


Fig. 1. SPARK framework.

Table 2
Parameter constraints.

Constraint	Rule
1	FetchWidth ≥ DecodeWidth
2	FetchBufferEntry > FetchWidth
3	RobEntry|DecodeWidth[a]
4	FetchBufferEntry|DecodeWidth[a]
5	FetchWidth = 2 × ICacheFetchBytes
6	IntPhyRegister = FpPhyRegister
7	LDQEntry = STQEntry
8	MemIssueWidth = FpIssueWidth
9	DecodeWidth ≥ IntIssueWdith

[a]The former should be divisible by the latter.

3. SPARK framework

In order to explore the designs with the highest performance-cost ratio for these factors, we establish the SPARK framework shown in Fig. 1. Three stages and one infrastructure constitute the multi-objective optimization framework:

1. Design sampling stage: The parameters of BOOM processor are modeled as feature vector to constitute a discrete design space. Initial design samples with extensive characterization are obtained by our proposed RED algorithm.
2. Model construction stage: To perform the fitting of the feature vector space to the object space, the DKL-GP model is built. The addition of edge designs to SPARK is advantageous since it largely simplify Pareto front searching for proxy model.
3. Exploration phase: Acquisition function, EHVI is maximized for querying a batch of new elite trials. Next, trigger parallel SPA-Gen flow to fetch SPA metrics back BO Engine, which achieve faster ADRS convergence and speed of optimal design searching.
4. SPA-Gen infrastructure: responsible for parallel implementation of different design configuration of RTL generation, automatic completion of functional simulation, logic synthesis, netlist simulation, power analysis and feed PPAs back algorithm.

3.1. Problem modeling

The entire design space is discrete, since the design variables' option are constrained to a specific range with discrete values. We define the individual design variable as x_i, and the combination of parameters satisfying the constraints can be characterized as the design feature vector, defined as $\boldsymbol{x} = \{x_1, x_2, x_3, \dots, x_N\}$. Therefore, different feature vectors form the design space $\boldsymbol{D}$.

Performance score, power consumption, and area cost are design metrics that we take into account. CoreMark is a simple, yet sophisticated benchmark that is designed specifically to test the functionality of a processor core. Running CoreMark produces a single-number score, allowing users to make quick comparisons between processors [15]. Therefore, cycle-accurate simulation of CoreMark for BOOM is completed to yield a score as performance measurement of the processor, which is defined as s. The power consumption includes dynamic and short-circuit power consumption that corresponding to the real toggling of signals by netlist simulation and the static power consumption of the T28 standard cell library respectively, defined as p. The area cost is the area overhead obtained from logic synthesis, defined as a. Thus, the indicators of three metrics constitute a response vector, defined as $\boldsymbol{y} = \{s, p, a\}$. Our design space exploration problem can therefore be concluded to explore a set of configuration of parameters $\boldsymbol{X}_{\text{Pareto}}$ with optimal performance metrics $\boldsymbol{Y}_{\text{Pareto}}$ within design space $\boldsymbol{D}$.

$$\boldsymbol{Y}_{\text{Pareto}} = \text{ParetoFrontier}(\text{Response}(\boldsymbol{D})) \tag{1}$$

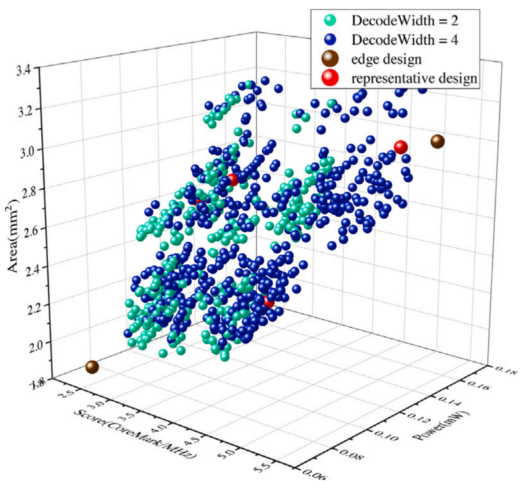


Fig. 2. Clusters, representative design and edge design.

The final step after finding Pareto front of design space is to choose a design from the Pareto front where the performance-cost ratio is appropriate for the anticipated criteria. The performance score divided by the product of power consumption and area cost is specified as SPAR (Score-Power-Area Ratio) judgment rule in this work. This rule indicates the performance score that a processor can exhibit on per-unit wafer area and per-unit power consumption. Here, We only apply this rule to assess design metrics of non-dominant designs in the end of workflow, but not employ it as single objective for optimization algorithm.

$$\mathrm{SPAR} = \frac{s}{p \times a} \tag{2}$$

The procedure of finding the optimal SPAR microarchitecture design is calculated as follows:

$$\boldsymbol{y}_{\mathrm{opt}} = \mathrm{argmax}_{\boldsymbol{y}\,\in \boldsymbol{Y}_{\mathrm{Pareto}}} \mathrm{SPAR}\,(\boldsymbol{y}) \tag{3}$$

3.2. RED algorithm

As design space normally contains a great number of possible combinations of design configuration, the selection of initial samples often follows two goals. First, the number of initial samples must be as small as practicable because of long time consumed for gathering performance metrics of initial samples. Secondly, the target space exhibits a nonlinear and multi-peak distribution to the feature space as a result of high-order effects and crossover effects of feature variables. Thus, not only representative designs should be sampled, but also best-performance and worst-performance designs with uniqueness should be included into initial samples to embrace both generality and distinctiveness.

For the consideration of generality, among all the processor's design factors, a few significant design variables have a major impact on performance. DecodeWidth is an indicator of how many instructions can be decoded in a given amount of time. The Decode Unit breaks down the format of instructions, extracts their information, and then processes these data to direct execution of the pipeline. Therefore, it is at the throat of pipeline and requires special consideration. For considering distinctiveness, extreme configurations of design parameters, which tend to achieve best or worst performance can benefit proxy model to reach more rich features to recognize the boundary of design space. We believe the convergence speed of optimization algorithm will take advantage of this consideration.

In this paper, we present RED algorithm to sample the initial designs from design space in order to satisfy both two requirements.

```
Algorithm 1 RED($\boldsymbol{P}, \boldsymbol{D}, \mathrm{K}$)
Input : Parameter Choices $\boldsymbol{P}$, Cluster number K,
        Design Space $\boldsymbol{D}$, Distance Weight $\boldsymbol{w}_r$, $\boldsymbol{w}_e$
Output : Initial Samples
1  $\boldsymbol{Clusters} = \text{K-Means}\left(\boldsymbol{D}, \mathrm{K}, \boldsymbol{w}_r\right)$;
2  $\boldsymbol{a}_{\max} = \mathrm{MaxComb}(\boldsymbol{P})$,  $\boldsymbol{a}_{\min} = \mathrm{MinComb}(\boldsymbol{P})$;
3  for $\boldsymbol{C}_i$ in $\boldsymbol{Clusters}$ do
4      $\boldsymbol{X}_{\mathrm{r}} = \boldsymbol{X}_{\mathrm{r}} \cup \mathrm{TED}(\boldsymbol{C}_i)$;    $\forall\, i \in \{1, 2, \cdots, \mathrm{K}\}$
5  End for
6  for $\boldsymbol{C}$ in $\{C_1,\ C_{\mathrm{K}}\}$ do
7      $\boldsymbol{x}_{\mathrm{e\_max}} = \mathrm{argmin}_{\boldsymbol{x}\in\boldsymbol{C}} \mathrm{Distance}(\boldsymbol{x}, \boldsymbol{a}_{\max}, \boldsymbol{w}_{\mathrm{e}})$;
       $\boldsymbol{x}_{\mathrm{e\_min}} = \mathrm{argmin}_{\boldsymbol{x}\in\boldsymbol{C}} \mathrm{Distance}(\boldsymbol{x}, \boldsymbol{a}_{\min}, \boldsymbol{w}_{\mathrm{e}})$;
8  End for
9  $\boldsymbol{X}_{\mathrm{initial}} = \boldsymbol{X}_{\mathrm{r}} \cup \boldsymbol{x}_{\mathrm{e\_max}} \cup \boldsymbol{x}_{\mathrm{e\_min}}$;
10 Return $\boldsymbol{X}_{\mathrm{initial}}$
```

```
Algorithm 2 BOED($\boldsymbol{P}, \boldsymbol{D}, \mathrm{K}, \mathrm{R}$)
Input : Parameter Choices $\boldsymbol{P}$, Design Space $\boldsymbol{D}$,
        Cluster number K, Round number R
Output : Optimal SPAR design
1  $\boldsymbol{X} = \mathrm{RED}\,(\boldsymbol{P}, \boldsymbol{D}, \mathrm{K})$;
2  $\boldsymbol{Y} = \mathrm{SPA{-}Gen}(\boldsymbol{X})$;
3  for $i \leftarrow 0$ to R do
4      Train DKL−GP model on $(\boldsymbol{X}, \boldsymbol{Y})$
5      $\boldsymbol{X}_{\mathrm{new}} = \mathrm{argmax}_{\boldsymbol{x}\in\boldsymbol{D}} \mathrm{EHVI}(\boldsymbol{x})$;
6      $\boldsymbol{Y}_{\mathrm{new}} = \mathrm{SPA{-}Gen}(\boldsymbol{X}_{\mathrm{new}})$;
7      $\boldsymbol{X} = \boldsymbol{X} \cup \boldsymbol{X}_{\mathrm{new}},\ \boldsymbol{Y} = \boldsymbol{Y} \cup \boldsymbol{Y}_{\mathrm{new}}$;
8      Construct Pareto Frontier $\boldsymbol{X}_{\mathrm{Pareto}}, \boldsymbol{Y}_{\mathrm{Pareto}}$ from $\boldsymbol{Y}$;
9      for $\boldsymbol{y}$ in $\boldsymbol{Y}_{\mathrm{Pareto}}$ do
10         if $\mathrm{SPAR}(\boldsymbol{y}) \geq \mathrm{SPAR}(\boldsymbol{y}_{MBC})$
11             $\boldsymbol{Opt} = \boldsymbol{Opt} \cup (\boldsymbol{x}, \boldsymbol{y})$;
12 End for
13 Return $\boldsymbol{X}_{\mathrm{Pareto}}, \boldsymbol{Y}_{\mathrm{Pareto}}, \boldsymbol{Opt}$
```

Representative designs selection: Acquired from BE, we adapt the conventional K-means method to group design feature vectors into five separate design clusters, taking DecodeWidth as the primary distance influence factor. Then apply the TED algorithm [16] within each cluster to identify the most representative feature vectors. For example, The blue and cyan balls in Fig. 2 show the distribution of the designs in the target space for DecodeWidth of 2 and 4, respectively. The five red balls are the representative designs obtained by RED.

Edge designs selection: The RED algorithm computes a combination of the maximum/minimum choice of design parameters to create two genuine extreme designs, which we name as anchor designs. According to perception of prior knowledge, the maximum anchor design tends to have the highest performance score, power consumption, and area cost after being mapped to target space, whereas the minimum anchor design is the reverse. Hence, among the K clusters formed from prior clustering, two clusters with DecodeWidth of 1 and 5 respectively are naturally chosen. The design nearest to the anchor inside each cluster is determined by distance calculation with respect to the maximum anchor and minimum anchor. When choosing an edge design, we consider distance evaluation in greater detail. The implications of each pipeline component on design metrics differ for superscalar processors. The Out-of-Order processor has most significant trait on disorder execution of instructions. The last commit stage, where ROB is in charge of committing and retiring instructions, is typically incorporated into the pipeline in order to maintain the validity of the program execution result. Additionally, hardware resources for data store are provided by integer/floating-point physical registers for execution of instructions. The parameters of these components receive larger weights in distance computation based on the superscalar Out-of-Order processor's features. Therefore, RED algorithm will extract edge designs from cluster using predefined distance weight. Here, the term "edge design" refers to such extreme design. Obviously, this consideration additionally covers the unique designs of the vast design space. In the end, initial

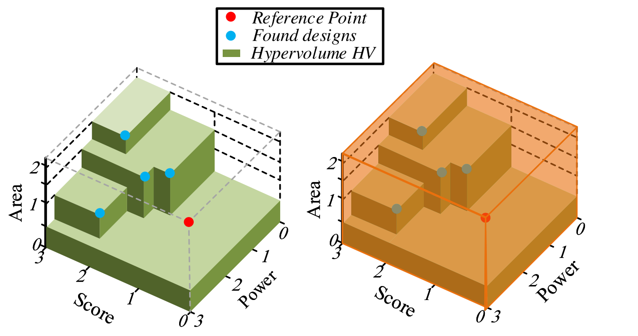


Fig. 3. Pareto front in objective space and hypervolumn surrounded by reference point and Pareto front.

training samples of proxy model are made up of edge designs and representative designs from clusters. The brown ball in Fig. 2 represents the maximum and minimum edge design obtained by RED.

3.3. Para-BOED algorithm

with generated initial sample set, Bayesian optimization algorithm with edge designs(BOED) is proposed. Due to the optimal-seeking effectiveness of the DKL-GP model presented in BE, BOED follow this proxy model. However, it should be highlighted that we generalize the model to the triple objective search issue in our problem.

Following training and converged, the model would be used to predict the mean and variance of three design metrics in object space, which determine the average level of performance prediction and its uncertainty. Each performance variable distribution is recognized as normal distribution. In order to efficiently solve the problem of multi-objective optimization, the acquisition function, Expected Hyper-Volumn Improvement(EHVI) is employed to compute for each sample. After setting a reference point that is slightly inferior to the poorest design metrics, the Pareto front and reference point surround three-dimensional hypervolume of the target space. The Pareto front in target space can be obtained by maximizing EHVI values of all samples. As shown in Fig. 3, the blue points are non-dominant designs in objective space, which compose Pareto front and surround a hypervolumn with reference point. The hypervolume is obtained by subtracting the volume of the green blocks from the orange block. For fast prototyping, we utilize relevant EHVI function in Botorch [17] to calculate the expected improvement for each sample and rank all candidates. Therefore, the multi-objective optimization problem can be concluded by consistently selecting the design that can dominate the old Pareto front to the maximum extent. To obtain their respective real SPA metrics, generated new trails need to query the SPA-Gen flow.

We sort the new design candidates obtained by maximizing the acquisition function in order to obtain a series of new trials from the highest rank to the lowest by the algorithm control, which are referred to as elite trials under the current iteration. This method was inspired by the elite-preserving strategy in the NSGA-II algorithm [18]. The parallel SPA-Gen Flow is designed and implemented to support parallel performance simulation process of elite trials, which can concurrently gather SPA of trials and significantly shorten overall running time of BOED algorithm. Hence, This mutually supportive set of method and tool increases the speed of algorithm convergence and optimal searching. BOED with parallelism assistance is what we refer to as Para-BOED here.

The original training data and elite trail data are joined after retrieving SPA metrics, and the proxy model is then refitted. Next, BOED predict the learned Pareto front once for each loop and compare it with the real Pareto to generate ADRS. When determining whether a design is superior to the official Two-Wide BOOM, SPAR metric mentioned before is adopted. BOED algorithm returns the optimal microarchitecture design with the highest SPAR after exiting the Bayesian optimization loop.

4. Experiment

4.1. SPA-Gen infrastructure

SPA-Gen infrastructure play the strength of Chipyard [19] platform resided by BOOM, to generate RTL for different BOOM configurations using its automatic processor-generated flow and configurable support features. The offline dataset samples used in our experiments are obtained by uniform random sampling in the design space. The 1032 designs in this dataset are the basis for our experiments. SPA evaluation process for samples and trials is roughly as follows:

1. Generate RTL: different BOOM parameters are automatically generated into Scala files and then compiled to RTL hardware description.
2. Performance simulation: In order to determine the processor's performance score in terms of MHz units, we evaluate CoreMark program on the BOOM SoC running on a bare-metal environment (without OS) using Verilator 4.225.
3. Logic synthesis and power consumption simulation: All created BOOM instances with design frequencies of 1.5 GHz are submitted to logic synthesis with front-end libraries of the TSMC 28 nm CMOS process in order to be as realistic as possible to real application scenarios in academia and industry. The RTL design's high-capacity register banks are substituted by SRAM macros using MacroCompiler prior to synthesis. In order to automate the power and area collection for each design, SPA-Gen also integrates the plugins of Synopsys Design Compiler Q-2019.12-SP5 for logic synthesis and PrimeTime O-2018.06-SP5 for power simulation to the Hammer flow [20]. Next, gate-level netlist simulation of the designs is carried out to extract the signal-toggling waveform, and PrimTimePX is used to analyze power consumption.

Performance score, power consumption, and area cost of a single BOOM design instance can be obtained with the help of the aforementioned three processes. The experiment time required to carry out the aforementioned procedures in order for a single design is quite expensive, lasting roughly 8 to 12 h. Obviously, continuing with sequential flow is quite time intensive for dataset preparation and algorithm iteration. In order to achieve a completely automatic parallel SPA throughput flow, SPA-Gen flow specifically considers the parallelizability between phases of EDA flow of single and different designs, which is supported by the resources capabilities of the EDA service cluster. In particular, SPA-Gen flow is advantageous for the elite trial obtained during sampling phase in BOED, and SPAs can be queried back in parallel to mask overall running time.

4.2. BOED efficiency

The Pareto front is obtained by non-dominated ordering of designs in the target space. Distance between learned Pareto front and real Pareto front is normally called ADRS, which means average distance from reference set. $\boldsymbol{\gamma}$ and $\boldsymbol{\omega}$ are designs on the real Pareto and learned Pareto front respectively, and here Dist is calculated using the Euclidean distance:

$$\mathrm{ADRS}(\boldsymbol{\Gamma}, \boldsymbol{\Omega}) = \frac{1}{|\boldsymbol{\Gamma}|} \sum_{\boldsymbol{\gamma} \in \boldsymbol{\Gamma}} \min_{\boldsymbol{\omega} \in \boldsymbol{\Omega}} \mathrm{Dist}(\boldsymbol{\gamma}, \boldsymbol{\omega}) \tag{4}$$

The effectiveness of learned Pareto designs is reflected by the ADRS indicator. The learned Pareto front more closely near the real Pareto front, the more it fulfills the expectations of the designer. In addition to ADRS, DSE problems typically take overall running time(ORT) into account, which illustrates the searching speed for optimal design could be conducted. Experiment baseline conditions are identical to those reported in respective papers [11] for fair comparisons. To calculate the mean, we performed the experiment ten times in order to rule out chance.

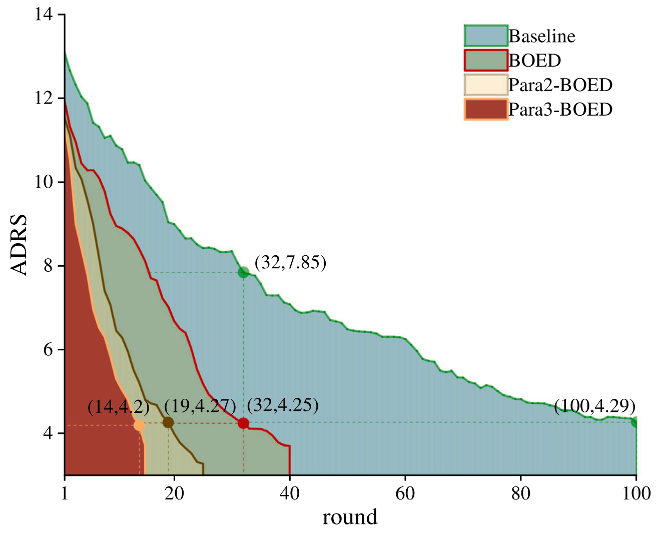


Fig. 4. ADRS convergence of baseline, seq-BOED, para2-BOED and para3-BOED.

4.2.1. *seq-BOED efficiency*

We contrast the sequential-BOED(seq-BOED) with the baseline to show the BOED algorithm's efficacy. Here, the triple objective problem's baseline results are produced by implementing algorithms in BE.

The convergence of ADRS metric for baseline and seq-BOED as optimization iteration runs is shown by green line and red line in Fig. 4. Results indicate that baseline starts to converge after 95 rounds. Using ADRS value attained by baseline at the 100th round as criteria, seq-BOED algorithm achieves a similar amount of ADRS convergence as baseline at the 32nd round. As the designer is more concerned with the number of queries in DSE issue, we set timing cost for each new sample to obtain their SPA metric from SPA-Gen flow as 8 h, which is an objective and understandable value. Thus, the overall running time for 100 rounds of baseline is 800 h, whereas the overall running time for 32 rounds of seq-BOED is just 256 h, which achieve a reduction of 2.125. Additionally, with 32 iterations as same time budget, compared to the ADRS number of BOED, the ADRS number of baseline is 7.85. Therefore, improved observations on convergence of ADRS show that seq-BOED finds Pareto front solutions with better quality than Baseline for the same number of rounds. Seq-BOED, from another perspective, runs less than one-third of the total searching time of baseline while achieving similar ADRS convergence targets.

4.2.2. *Parallelism sensitivity*

To verify the effectiveness of parallel strategy in Para-BOED and its parallelism sensitivity, we set the parallelism parameter to 2 and 3, repeated 10 times, and take the average value. We name these two experiments as para2-BOED and para3-BOED. For the consideration of redundancy of elite trials and algorithm running time, we do not increase Parallelism further. The convergence of ADRS for seq-BOED, parallel2-BOED, and parallel3-BOED, respectively, is shown in Fig. 4 by the red, brown, and yellow curves. Results reveal that the technique of running multiple elite trials simultaneously against the VLSI flow can significantly increase search speed. While Para3-BOED only requires 14 shots to complete the identical ADRS target, Para2-BOED requires 19 rounds. The ORT is reduced into 0.14 and 0.19 times as fast as baseline, based on the mask of SPA delivery time for new trials using SPA-Gen flow. Respectively, the total running time is 112 h and 152 h. In light of this, Para3-BOED can identify the suitable Pareto front in less time than 5 days.

Table 3
Comparison between official Two-Wide BOOM and optimal SPAR BOOM from SPARK.

Paramter	Official TWB	SPARK
FetchWidth	4	4
FetchBufferEntry	16	40
RasEntry	32	16
BranchCount	12	12
ICacheWay	4	2
ICacheTLB	8	32
ICacheFetchBytes	2	2
DecodeWidth	2	2
RobEntry	64	64
IntPhyRegister	80	96
FpPhyRegister	64	96
MemIssueWidth	1	1
IntIssueWidth	2	2
FpIssueWidth	1	1
LDQEntry	16	24
STQEntry	16	24
DCacheWay	4	2
DCacheMSHR	2	2
DCacheTLB	8	8
Score (CoreMark/MHz)	3.839785	**3.89595**
Power (mW)	0.088313	0.091531
Area (mm^2)	2.047098	**1.996553**
SPAR	21.23954	**21.318877**

Table 4
Query times of baseline and our approaches for SPAR optimal search.

Method	Official TWB	SPAR optimal
baseline	2	37
seq-BOED	1	17
para2-BOED	1	7
para3-BOED	**1**	**6**

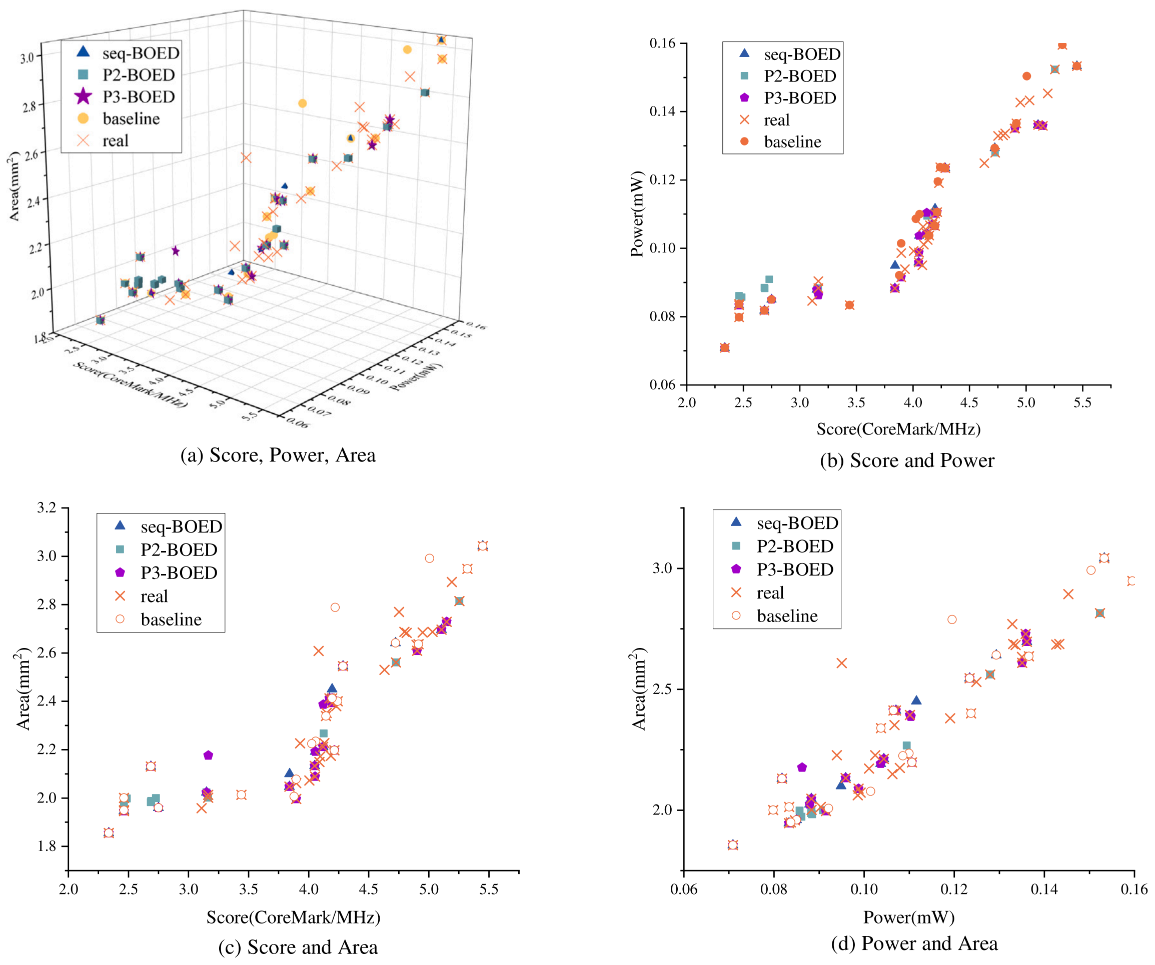
4.3. *Pareto front*

Fig. 5 displays the learned Pareto front for seq-BOED, para2-BOED, para3-BOED and baseline respectively. In contrast to subplot (a), which depicts a metrics view in three dimensions, subplots (b)–(d) present arbitrary two of three metrics as a 2D plan view. From the standpoint of designers, the Pareto front obtained by SPARK has a reliable range of distribution, allowing the designer to select in accordance with certain design requirements.

4.4. *Optimal SPAR design*

The parameters of the optimal SPAR design, as determined by Para-BOED, are displayed in Table 3. It is obvious that our design's SPAR metric is higher than official Two-Wide BOOM's. Our optimal design outperforms than the Official one in terms of performance score and area overhead, but it uses a little more energy. Both of them are two-width processors, and their structural parameters – such as issue queue parameters, ROB size, branch count, DCache MSHR, and DCache TLB size – are essentially the same. And subtly different on smaller but crucial elements, like the number of physical registers and the size of the load-store queue. We compare the least Query times acquiring optimal SPAR design from ten experiments of different approaches in Table 4

It is worth noting that reaching the SPAR optimal point by our SPARK framework takes only 6 queries, which represents spending only a few days to obtain the optimal design of BOOM processor. However, manually designing the official Two-Wide BOOM might need several weeks or even longer. The optimization methodology of SPARK framework clearly demonstrates its efficacy and great speed advantage.



(a) Score, Power, Area

(b) Score and Power

(c) Score and Area

(d) Power and Area

Fig. 5. Pareto front obtained by baseline, seq-BOED, para2-BOED and para3-BOED.

5. Conclusion

In this article, we propose a framework for automatic multi-objective processor microarchitecture seeking. Pareto front search is remarkably accelerated by the addition of edge designs, and overall running time of BOED algorithm is significantly reduced based on the recommended SPA-Gen flow. According to experimental findings, SPARK is capable to discover an optimal processor microarchitecture on SPAR criteria within reasonable time budget. Our upcoming research on design space exploration will concentrate on creating more effective sampling techniques and proxy models in order to condense the design space and potentially undertake the task of global optimization. We anticipate that the community of processor design and optimization will consider SPARK to be instructive.

CRediT authorship contribution statement

Qiang Li: Conceptualization, Methodology, Software, Hardware design, Data analysis, Writing – original draft. **Jun Tao:** Supervision, Technical guidance. **Jun Han:** Funding acquisition, Supervision, Project management, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors are unable or have chosen not to specify which data has been used.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant 61934002.

References

[1] K. Asanovic, D.A. Patterson, C. Celio, The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor, Technical Report, University of California at Berkeley Berkeley United States, 2015.
[2] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, K. Asanović, Chisel: constructing hardware in a scala embedded language, in: DAC Design Automation Conference 2012, IEEE, 2012, pp. 1212–1221.
[3] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, et al., The gem5 simulator, ACM SIGARCH Comput. Archit. News 39 (2) (2011) 1–7.
[4] A. Patel, F. Afram, K. Ghose, Marss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors, in: 1st International Qemu Users' Forum, Citeseer, 2011, pp. 29–30.
[5] T.E. Carlson, W. Heirman, L. Eeckhout, Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, 2011, pp. 1–12.
[6] D. Sanchez, C. Kozyrakis, Zsim: Fast and accurate microarchitectural simulation of thousand-core systems, ACM SIGARCH Comput. Archit. News 41 (3) (2013) 475–486.

[7] S. Salamin, M. Rapp, A. Pathania, A. Maity, J. Henkel, T. Mitra, H. Amrouch, Power-efficient heterogeneous many-core design with NCFET technology, IEEE Trans. Comput. 70 (9) (2021) 1484–1497.

[8] E. İpek, S.A. McKee, R. Caruana, B.R. de Supinski, M. Schulz, Efficiently exploring architectural design spaces via predictive modeling, Oper. Syst. Rev. 40 (5) (2006) 195–206.

[9] C. Dubach, T.M. Jones, M.F. O'Boyle, An empirical architecture-centric approach to microarchitectural design space exploration, IEEE Trans. Comput. 60 (10) (2011) 1445–1458.

[10] B.C. Lee, D.M. Brooks, Illustrative design space studies with microarchitectural regression models, in: 2007 IEEE 13th International Symposium on High Performance Computer Architecture, IEEE, 2007, pp. 340–351.

[11] D. Li, S. Yao, Y.-H. Liu, S. Wang, X.-H. Sun, Efficient design space exploration via statistical sampling and AdaBoost learning, in: 2016 53nd ACM/EDAC/IEEE Design Automation Conference, DAC, IEEE, 2016, pp. 1–6.

[12] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, 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.

[13] M. Moudgill, P. Bose, J.H. Moreno, Validation of turandot, a fast processor model for microarchitecture exploration, in: 1999 IEEE International Performance, Computing and Communications Conference (Cat. No. 99CH36305), IEEE, 1999, pp. 451–457.

[14] D. Brooks, P. Bose, V. Srinivasan, M.K. Gschwind, P.G. Emma, M.G. Rosenfield, New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors, IBM J. Res. Dev. 47 (5.6) (2003) 653–670.

[15] CoreMark, 2022, URL https://www.eembc.org/coremark/, (accessed 6 December 2022).

[16] K. Yu, J. Bi, V. Tresp, Active learning via transductive experimental design, in: Proceedings of the 23rd International Conference on Machine Learning, 2006, pp. 1081–1088.

[17] M. Balandat, B. Karrer, D.R. Jiang, S. Daulton, B. Letham, A.G. Wilson, E. Bakshy, BoTorch: A framework for efficient Monte-Carlo Bayesian optimization, in: Advances in Neural Information Processing Systems 33, 2020, URL http://arxiv.org/abs/1910.06403.

[18] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.

[19] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, et al., Chipyard: Integrated design, simulation, and implementation framework for custom socs, IEEE Micro 40 (4) (2020) 10–21.

[20] H. Liew, D. Grubb, J. Wright, C. Schmidt, N. Krzysztofowicz, A. Izraelevitz, E. Wang, K. Asanović, J. Bachrach, B. Nikolić, Hammer: a modular and reusable physical design flow tool, in: Proceedings of the 59th ACM/IEEE Design Automation Conference, 2022, pp. 1335–1338.