Resource-Constraint Bayesian Optimization for Soft Processors on FPGAs

CE GUO, HAORAN WU, and WAYNE LUK, Imperial College London, UK

This paper introduces a Bayesian optimization method tailored for soft processors on FPGAs. It addresses the challenge of tuning soft processor parameters to align performance, power efficiency, and resource allocation with application-specific demands. It presents three innovations: a strategy based on a constraint function for enforcing usage constraints across various FPGA resources, a refined approach for managing restricted integer parameters beyond simple rounding, and a comprehensive evaluation via four case studies on an open-source parametric RISC-V design. A key advantage of the proposed approach over existing ones is that it enables users to set constraints on hardware resource usage, ensuring that the optimized design fits into the target FPGA. This method not only streamlines the optimization process but also ensures adherence to hardware resource constraints. Experimental results show that our proposed approach substantially outperforms existing methods for optimizing FPGA-based soft processors, achieving a 12% higher success rate in identifying the best processor design and requiring approximately 23% fewer optimization iterations.

CCS Concepts: • **Hardware → Evolvable hardware**; **Software tools for EDA**; • **Computer systems organization →** *Embedded hardware*; **Reconfigurable computing**; *Reduced instruction set computing*; *Self-organizing autonomic computing*; *High-level language architectures.*

Additional Key Words and Phrases: Bayesian optimization, parametric design, soft processor, FPGA, parameter tuning, resource constraint

ACM Reference Format:
Ce Guo, Haoran Wu, and Wayne Luk. 2024. Resource-Constraint Bayesian Optimization for Soft Processors on FPGAs. In *14th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART '24), June 19–21, 2024, Porto, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3665283.3665291

1 INTRODUCTION

Soft processors running on the FPGA hardware leverage the reconfigurable nature of FPGAs to allow designers to tailor processor features, such as instruction sets and microarchitecture, to meet performance, power, and functional requirements. A soft processor can be viewed as a parametric hardware design where the components' characteristics are encoded into parameters that can be defined and adjusted by the designer to meet design requirements. These parameters can include aspects such as the size of memory elements, the number of processing units, algorithmic settings, and more.

Tuning the parameters for soft processors on FPGAs can be challenging due to the need to balance performance, power efficiency, and resource utilization specific to the application's requirements. Manual tuning of these parameters is often impractical due to the complexity and vastness of the parameter space. The parameter space is typically high-dimensional and non-linear, with numerous interdependent variables that need to be adjusted simultaneously. Additionally, the evaluation of parameter combinations can be extremely time-consuming, as the compilation of FPGA designs involves a full synthesis, placement, routing, bitstream generation, and application profiling, which can take

Manuscript submitted to ACM

hours or even days to complete. This lengthy evaluation cycle necessitates that the optimization process finishes with as few evaluations as possible.

Black-box optimization is a promising solution for optimizing parameters for soft processors. Such techniques do not require explicit knowledge of the underlying hardware structure. Instead, Black-box optimization algorithms, especially those employing heuristics like Bayesian optimization (BO), can efficiently explore the large design space with predictive models.

This study introduces an enhanced Bayesian optimization method for tuning parameters for soft processors on FPGAs, with three contributions specifically designed to account for the features and limitations of FPGAs:

- A mechanism for integrating a constraint function into Bayesian optimization, allowing users to specify limits for various types of on-chip resources, ensuring that the optimized design adheres to these limits. See Section 3.
- An innovative technique for more effective optimization of restricted integer parameters, surpassing traditional rounding methods. See Section 4.
- A demonstration of the effectiveness of the proposed method through its application to four distinct case studies involving an open-source parametric RISC-V soft processor design. See Section 5.

2 BACKGROUND

The exploration of soft processor optimization benefits from Bayesian optimization techniques. This section delves into the background of employing these techniques for design optimization.

2.1 Design Parameter Tuning and Bayesian Optimization

The optimization of soft processors can be considered a black-box optimization problem because the exact relationship between design parameters and outcomes like speed, power consumption, or resource usage is complex and not explicitly known. Designers often rely on simulations or empirical evaluations to determine the effectiveness of different parameter combinations. As a result, the optimization problem is challenging, since each evaluation of the objective function can be costly in terms of computational resources, time, or money. Also, there is no access to the derivatives of the objective function, making gradient-based methods inapplicable.

Bayesian optimization [15] is a sequential optimization strategy that addresses the challenges in this black-box optimization problem. A Bayesian optimizer searches the design space efficiently without requiring derivative information, as it can leverage a surrogate model to learn about the design space. Let f be an objective function. The input x is a configuration of design parameters, while the function value $f(x)$ is a performance metric such as the overall execution time, the cycle count, or the clock frequency. Bayesian optimization uses a surrogate model, typically a Gaussian process (GP), to model f. This surrogate model provides a probabilistic estimate of $f(x)$ for any x. Meanwhile, an acquisition function g is used to decide which parameter configuration to evaluate next. This function is derived from the surrogate model and balances the exploration of areas with high uncertainty against the exploitation of areas predicted to have good outcomes. Unlike the original black-box function f, the acquisition function g is cheaper to evaluate. The parameter configuration chosen with g is then evaluated with the true objective function to get the performance data, and the surrogate model is updated with the evaluation results. The optimization process iterates until a stopping criterion is met, which could be a maximum number of evaluations, a convergence in observed values, or other problem-specific criteria.

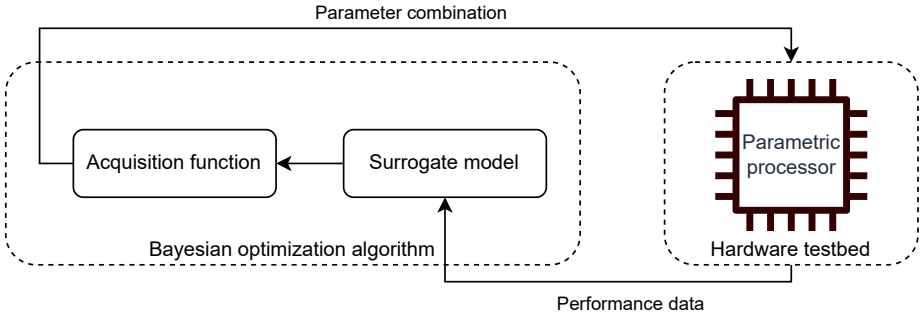


Fig. 1. Typical Bayesian optimization for parametric soft processors

Table 1. Recent Bayesian optimization methods for parametric hardware

Method	BOOM-Explorer	Boomerang	Orbit-ML	Proposed
Reference	[3]	[12]	[6]	This paper
Year	2021	2023	2023	2024
Target architecture	ASIC	ASIC	Generic	FPGA
Supported processor(s)	BOOM	SonicBOOM	Generic	Generic
Gaussian process surrogate	Regular	Regular	Regular	Resource-aware
Physical synthesis	No	Yes	Yes	Yes
Resource limit enforcement support	No	No	Partial	Yes
Restricted integer parameters	As integers	As integers	As integers	Special treatments
Per-application optimization support	No	No	Yes but not evaluated	Yes

2.2 Bayesian Optimization Methods for Parametric Hardware

Research has explored applying Bayesian optimization to ASIC and FPGA designs, focusing on architectural design optimization for parametric processors. Fig. 1 shows the typical workflow of these methods. The Bayesian optimization algorithm produces a parameter combination using a surrogate model and an acquisition function in each iteration. The hardware testbed then evaluates this parameter combination and provides performance data, which the algorithm uses to update the surrogate model and suggest the next parameter combination.

Recent techniques optimizing parametric processors with Bayesian optimization include BOOM-Explorer [3], Boomerang [12], and Orbit-ML [6]. Table 1 summarizes the features of these studies along with our proposed approach. BOOM-Explorer [3] is a method for design space exploration in the ASIC design of the BOOM processor. It addresses the complexities of parameter tuning for BOOM, showcasing an optimization workflow that not only reduces optimization time but also enhances design efficiency and accuracy. Boomerang [12] is another Bayesian optimization framework for processor design. Similar to BOOM-Explorer, Boomerang addresses the challenges of optimizing a highly parameterized processor. Unlike BOOM-Explorer, Boomerang incorporates physical synthesis techniques for a more accurate and efficient optimization. The Orbit-ML [6] framework is another Bayesian optimization approach for processor parameter optimization. Different from BOOM-Explorer and Boomerang, Orbit-ML does not focus on a particular processor but provides a generic framework to optimize parametric processors with any numeric and categorical parameters.

In addition to Bayesian optimization, there are also other block-box hardware optimization techniques. For example, an early optimization method, ARDEGO [10], aims at tuning FPGA designs through efficient black-box optimization.

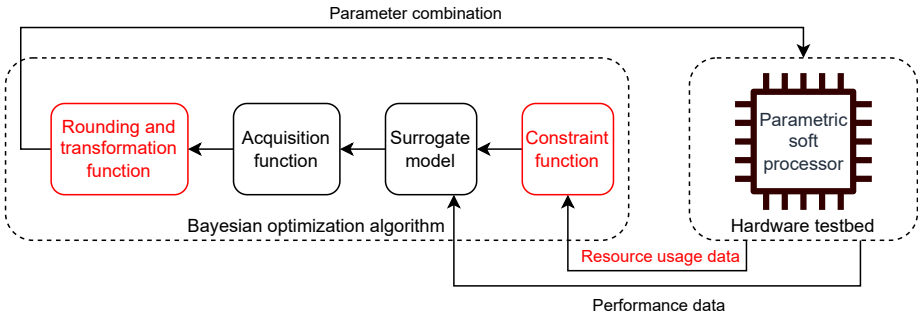


Fig. 2. Proposed Bayesian optimization workflow: new critical components highlighted in red

Like Bayesian optimization, ARDEGO uses a surrogate model to approximate the design space. However, unlike Bayesian optimization, ARDEGO does not use an acquisition function to suggest new parameter combinations. Another example is the hill climbing approach in [1]. This approach employs a greedy search strategy to optimize register file size and the resource allocation policy for simultaneous multithreading (SMT) processors.

Besides studies optimizing architecture-level design parameters, there are also studies applying Bayesian optimization to enhance computational modules, or to tune software logic that runs on the FPGA. For example, the approximate multiplier generation approach in [11] employs Bayesian optimization to balance between accuracy and hardware efficiency. This approach provides optimized approximate multipliers by tuning the parameters on half adders with precision and hardware cost considerations. Another example is that the design flow for the FPGA-based object detector in [14] uses Bayesian optimization to determine optimal layer sparsity. In other words, Bayesian optimization is used here to optimize the software running on the FPGA instead of optimizing the hardware directly.

3 HARDWARE PARAMETER OPTIMIZATION UNDER RESOURCE CONSTRAINTS

FPGA designers must manage resource usage carefully. If a design exceeds an FPGA's resource limits, it can not be implemented without moving to a larger, more expensive FPGA or redesigning the logic to be more resource-efficient. The next two subsections respectively discuss our proposed Bayesian optimization method with a resource constraint function and the soft processor we use for development and evaluation.

3.1 Bayesian Optimization with Resource Constraint Function

When optimizing soft processors on FPGAs, it is crucial for the model to enforce resource constraints. These constraints include limitations on the usage of look-up tables (LUTs) and flip flops (FFs), which are determined by the specification of particular FPGA devices.

We propose to integrate resource constraint enforcement into Bayesian optimization. This is because generating designs that fail to meet resource constraints not only wastes optimization iterations but also may contaminate the surrogate model, leading to misguided optimization directions. For example, without considering output constraints, designs that violate the constraints could potentially achieve better performance. Therefore, the surrogate GP model might assign higher values to regions outside the constrained valid design space, making it more likely for the acquisition function to select points from these higher-valued spaces. As a result, subsequent optimization tends to get trapped in a cycle of iteratively selecting similar samples from regions that violate the constraints.

To enforce resource constraints, we design a constraint function $c(x) \in \mathbb{R}^d$ that specifies the design's adherence to d different resource utilization constraints, including LUTs, FFs, and DSPs. In the optimization training process, we combine it with the output of the objective function $f(x) \in \mathbb{R}$, denoted as $(f(x), c(x))$. The resulting tensor is inputted into the surrogate model so that the model can learn the output objective function as well as the constraint function with an independent GP [7, 8] at the same time. To accommodate this constraint function, we revise the optimization workflow as shown in Fig. 2 where the testbed provides resource usage data to the surrogate model via the constraint function. Note that resource usage cannot be determined prior to evaluating the parameter combination. Consequently, the optimizer cannot guarantee that a generated parameter combination can produce a design that remains within the resource limit before evaluating it.

A proper constraint function should only be positive when the design is within the resource constraints. Also, the function should be formulated within specific constraints and be independent of the constraint's scale in order to ensure uniform constraint strength. Additionally, it is preferable for the function to quickly reach a plateau for valid designs to prevent the model from favoring sample designs associated with higher constraint function values. Based on these requirements, we propose the following constraint function:

$$c_i(x) = \frac{1}{1 + e^{-k(r_i(x) - \mathcal{L}_i)}} - \frac{1}{1 + e^{k(r_i(x) - \mathcal{U}_i)}} - 0.5 \tag{1}$$

where $r_i(x)$ represents the resource utilization for certain aspects and $\mathcal{L}_i$ and $\mathcal{U}_i$ represent the corresponding lower and upper bound of the resource constraints. k represents the steepness of the curve. It allows the function to balance computational simplicity with the function's ability to quickly reach a plateau when within the bounds of the constraints. An experiment is conducted to measure the execution time and the number iterations taken to identify the best design in a custom parameter dataset. A moderate value of $k = 10$ is determined, balancing the model's performance in both measurements.

The derived function employs logistic functions to bound the output $c_i(x)$ in $[-0.5, 0.5]$ and centered around zero. Fig. 3 provides example realizations of the constraint function considering five different bounds $r_i(x)$. The enforcement of resource constraints is achieved via monitoring the sign of the constraint function. If all dimensions of the constraint function output $r_i(x)$ are within the predefined condition, the value of the constrained function is positive. Otherwise, the value is negative. Therefore, the sign of the auxiliary outcome, represented by the final element of the output tensor $(f(x), c(x))$, represents the validity of the design.

A key advantage of the proposed approach over existing ones is that it enables users to set constraints on hardware resource usage, ensuring that the optimized design fits into the target FPGA. Specifically, among current methods that use the Bayesian optimization method to customize processors, popular Bayesian optimization methods as mentioned in Section 2.2 do not have specific considerations for the constraints related to the design's resource utilization. Therefore, it is likely that after a few iterations of optimization, these models would produce high-performance designs but fail to adhere to the resource utilization constraints.

3.2 VeeR EL2 Soft Processor on FPGAs

There are multiple customizable open-source RISC-V processors, such as Ibex [13] and Rocket-chip [2]. In this study, we use the VeeR EL2 Core [5] to develop and evaluate our proposed approach. The remainder of this section introduces this soft processor and its parameters. The evaluation of our Bayesian optimization method applied to this processor is presented in Section 5.

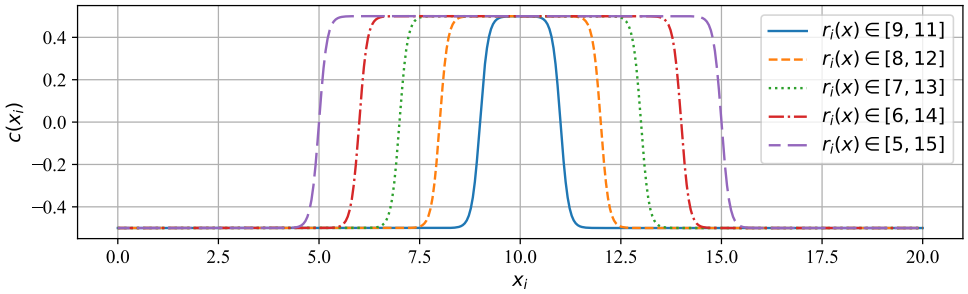


Fig. 3. Example constraint functions

Table 2. FPGA Resource Specifications

Device Name	Part Number	LUT Elements	Flip-Flops
AMD Zynq Z-7012S	XC7Z012S	32600	65200
AMD Zynq Z-7014S	XC7Z014S	40600	81200

Table 3. Benchmarks

Benchmark	**Description**
CoreMark	Benchmark for performance testing for basic operations, using code and data stored in external memories to simulate real-world applications.
CoreMark_ICCM	CoreMark variant with code preloaded into Instruction Closely Coupled Memory (ICCM).
CoreMark_DCCM	CoreMark variant with data and stack in Data Closely Coupled Memory (DCCM).
Dhrystone	Performance benchmark that is particularly sensitive to changes in cache size.

The Veer EL2 core is a high-performance, low-power RISC-V design that supports a large design space for customization. Additionally, the core is easy to implement on FPGAs, and its design is fully open-source under the CHIPS Alliance project. Furthermore, when deployed on an FPGA, the VeeR EL2 core can benefit from the FPGA reconfigurable nature of FPGAs, allowing developers to optimize and tailor the processor's configuration to their specific application needs.

Regarding on-chip resources, we primarily focused on the usage of Look-Up Tables (LUTs) and Flip-Flops (FFs) as the key criteria to determine the feasibility of implementing the core onto the FPGA. We selected two modern AMD Zynq-7000 SoC FPGAs and the chip resource limitations are shown in Table 2. The VeeR EL2 core GitHub repository includes several prevalent benchmarks based on simulations with Verilator, a tool for high-performance RTL simulation [5]. These benchmarks can quickly indicate the core's performance across various aspects.

To assess the reliability of our Bayesian optimization method, we examine its effectiveness in customizing the VeeR EL2 core to minimize its execution cycles for four different benchmarks based on CoreMark, as presented in Table 3. While CoreMark is traditionally designed to evaluate embedded systems performance, it can also be used in soft processor deployments where resource usage is a major concern. In the experiments, we use the cycle count as the performance objective, with on-chip resource usage as the constraint.

Our current study focuses on maximizing performance while keeping resource usage within a predefined limit. While we recognize the importance of power efficiency and acknowledge that it can be incorporated as an optimization

Table 4. Parameters for the VeeR EL2 core

Parameter	Description	Selected Range
btb_size	The size of the Branch Target Buffer (BTB)	2^3 to 2^9
lsu_num_nbload	Number of non-blocking loads executed by Load Store Unit (LSU)	2^1 to 2^3
lsu_stbuf_depth	Number of store operations the LSU buffer can hold simultaneously.	2^1 to 2^3
dccm_size	DCCM size	2^2 to 2^9
iccm_size	ICCM size	2^2 to 2^9
icache_size	Instruction Cache Size	2^3 to 2^8

objective or constraint, our development and evaluation do not include power efficiency measurements due to the limitations of our experimental platform. On the other hand, our evaluation focuses on the combined effect of the GP and the acquisition function in optimizing performance within the given constraints. This is because the GP influences overall performance indirectly through the acquisition function, which guides the optimization process. Therefore, it is challenging to measure the GP's influence on performance independently of the acquisition function.

We selected a combination of parameters with varying levels of impact on the performance of the customized processor to define the test design space. We investigated the impact of each parameter on the processor's resource utilization and performance in chosen benchmarks. The investigation is realized by trying different combinations of one parameter while keeping others as the default value and then comparing the performance outcomes with the default baseline design. The parameter selections were based on the observed effects, ranging from significant to negligible effects on the performance. The chosen parameters are detailed in Table 4.

The design space covers 30,618 distinct parameter combinations. Within this space, the parameters 'btb_size' and 'lsu_num_nbload' have a significant influence on performance across all the selected benchmarks. On the other hand, the 'lsu_stbuf_depth' parameter has a moderate effect. 'dccm_size', 'iccm_size', and 'icache_size' exhibit threshold behaviour; that is, their impact on performance is negligible as long as they meet a specific minimum threshold. However, failing to reach this threshold results in substantial performance degradation. Memory and cache sizes are commonly selected parameters in various algorithm applications. For example, the BOOM-Explorer method also takes these parameters into account when customizing the BOOM processor [3].

It is important to note that in our design space, all parameters are scaled on a logarithmic base of 2. Since the surrogate model does not inherently handle logarithmic scale variables, it is essential to transform them into a continuous scale to reduce the likelihood of generating designs that do not meet the required scale specifications. Additionally, to mitigate the impact of varying parameter scales, normalization is necessary. The overall transformation of the input variables is as follows:

$$x_i = \frac{\log_2(p_i) - \omega}{\log_2(\max(p_i)) - \log_2(\min(p_i))} \tag{2}$$

where ω is the parameter's offset, usually the smallest value of the parameters; x_i represents the normalized value of the parameter and p_i is the original parameter value. After this normalization process, we could ensure all the parameters tuned by our Bayesian optimization method are within the range of $[0, 1]$. The parameters produced by the Bayesian optimization method are then transformed back to the original scale to customize the core.

4 TUNING RESTRICTED INTEGER PARAMETERS

We find that many tunable parameters for FPGA-based soft processors are restricted integer values. These parameters often represent discrete quantities or selections, such as the number of logic blocks, memory units, parallel execution

units, or specific configuration options that cannot be fractional or continuous. These parameters must align with the physical and architectural constraints of the FPGA hardware, which inherently operates on integral configurations for resources and architectural features.

The discreteness of these parameters hamper the workflow of Bayesian optimization. Although a restricted integer parameter has a limited range of choices, tuning such parameters with Bayesian optimization poses additional challenges compared to tuning numeric parameters. This is because the optimizer needs to navigate a discrete and often non-continuous search space. This discreteness limits the application of gradient-based optimization techniques in the sampling process of the next design, and makes the search for optimal configurations more complex. Bayesian optimization must employ specific strategies, such as acquisition functions tailored for discrete spaces, to efficiently explore and evaluate the performance of these integer-based values.

The existence of restricted integer parameters conflicts with the design of common surrogate models like the GP, which usually assumes real-valued input variables. To address this problem, common methods are to introduce approximations such as the integer rounding at the output of the GP model. This transforms the invalid floating-point variables into valid integer parameter sets suitable for processor tuning. However, these approaches would negatively influence the optimization process [9]. Considering the integer rounding method, the primary problem is that the objective function remains unchanged in the regions of the input space where variables can be approximated to the same integer-valued input value, leading the surrogate model to allocate some probability mass to invalid points. In addition, other methods to process discrete variables, like one-hot encoding, largely increase the dimensionality and complexity of the design space.

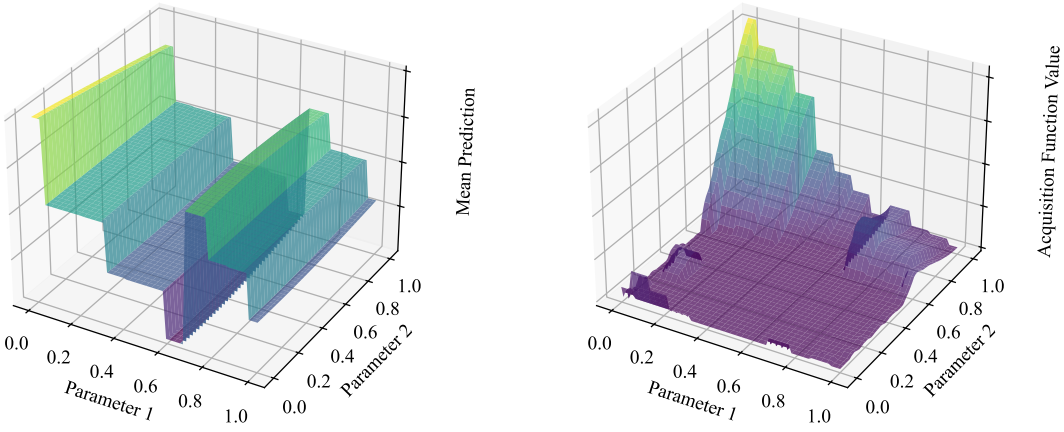
One method has been proposed to mitigate the negative impact caused by the approximations for integer-valued input variables [9]. This method is based on customizing the covariance method within the GP. The covariance function $k(\mathbf{x}, \mathbf{x}')$ defines the relationship between pairs of sample points. Together with the mean function $m(\mathbf{x})$, it describes the GP used to approximate the objective function $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$ [4]. This approach involves treating every input variable as continuous quantities and rounding them before the calculation of the covariance kernel. The modified kernel is as follows:

$$k'(\mathbf{x}, \mathbf{x}') = k(T(\mathbf{x}), T(\mathbf{x}')) \tag{3}$$

where T is the transformation that rounds the originally real-valued input variables to the nearest integers. This transformation enables the surrogate model to recognize the discontinuous nature of the input variables. Additionally, it improves the acquisition function's performance in handling these discrete variables.

Fig. 4 illustrates an example of the surrogate model distribution and the acquisition function, respectively. The GP and acquisition function both assume that the objective remains constant within certain real-value intervals that round to identical integer values. Furthermore, the uncertainty of the objective function, as represented by the variance obtained by the modified kernel, is also the same in the interval. After a single measurement in one interval, the interval's corresponding uncertainty about the objective function decreases to zero directly [9]. This reduction could also enhance sampling efficiency, as it eliminates the need to sample points in nearby regions that would round to the same integer values.

The proposed surrogate model results in a different workflow from existing approaches. Fig. 5 illustrates the modifications we have made to our proposed surrogate model in comparison to the common surrogate model employed in all Bayesian optimization methods mentioned in Section 2.2. Typically, these Bayesian optimization methods round the generated parameters at the output of the Bayesian optimization process. After the objective function is evaluated



(a) Sample Surrogate Model Distribution

(b) Sample Acquisition Function

Fig. 4. Example of Surrogate Model Distribution and Acquisition Function after the Kernel Transformation.

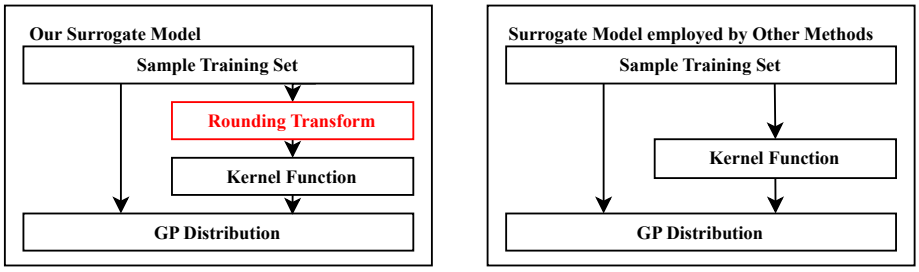


Fig. 5. Comparison of our proposed surrogate model configuration with the common surrogate model.

using these rounded parameters, the results are fed back into the surrogate model to update the GP distribution. However, without modifications to the kernel, the surrogate model fails to realize the constant behavior in regions that could round to the same integer-valued parameters and corresponding objective values. As a result, the model may exhibit significant variability in these regions, which potentially lead to optimization inefficiency [9].

5 EVALUATION

To evaluate our Bayesian optimization method, we designed a rigorous experimental setup considering the core's design space, resource constraints, and performance across multiple benchmarks. The evaluation focused on how effectively the method navigates the complex design space to identify parameter sets that meet resource constraints and enhance performance.

5.1 Experiment Setup

Experiments are conducted using our Bayesian optimization method to customize the VeeR EL2 core within the predefined design space, with the resource constraints set by the two FPGAs. In each experiment, the model was tasked with customizing the core for the four chosen benchmarks while adhering to the resource constraints. The performance objectives were obtained by analyzing simulation results provided by Verilator, and the resource utilization was extracted from Vivado's synthesis reports. The overall anticipated quantities of the LUTs and Flip-Flops usages are obtained in this process. The resource utilization is passed to the constraint function and used for GP model update.

We introduce two baseline design space exploration methods to compare with our Bayesian optimization method. First, we explored the fundamental method hill climbing [1]. The hill climbing method starts with a random solution and iteratively makes small adjustments to the current solution. At each step, it evaluates the new solution with the adjustment and selects the one that offers a performance improvement. This algorithm continues until no further improvements can be found. Second, we introduced a standard Bayesian optimization method without the modifications for output constraints and integer-valued parameters as described in Sections 3 and 4. This model involves a basic GP surrogate model and an expected improvement-based acquisition function. In every iteration, the model will select a point with greater potential for achieving high objective values to evaluate but without considering whether it conforms to the resource constraints.

We also compare our approach with two prevalent Bayesian optimization methods, Bayesian BOOM-Explorer [3] and Boomerang [12]. As they do not inherently consider resource constraints, a direct comparison is not feasible. We adapted the standard Bayesian optimization workflow to approximate their settings and assess their optimization capability. Specifically, we implemented additional logic to ensure that if a design fails to meet resource constraints, the surrogate model is not updated with the failed design. We do not compare our approach with ARDEGO [10] because it is dated and it lacks a clear method for applying its support vector machine technique to generate parameter combinations for our soft processor.

5.2 Results and Analysis

The first experiment aimed to evaluate the model's efficacy in enhancing the performance of the customized VeeR EL2 core across the four benchmarks outlined in Table 3, within the resource constraints set by the Z-7012S FPGA. The outcomes are shown in Fig. 6. This optimization task is challenging for several reasons. Firstly, there are multiple designs, despite varying parameter settings, that can achieve similar performance in certain benchmarks. This results in numerous potential global and local minima. Algorithms like hill climbing, which optimize based on the proximity of current solutions, are prone to getting trapped in these local minima. This might explain why hill climbing often converges quickly to a suboptimal core design. Moreover, the strict resource constraints imposed by the Z-7012S FPGA limit the number of valid parameter combinations that meet resource constraints. Consequently, other Bayesian optimization methods, which do not specifically consider resource constraints, are more likely to generate invalid designs compared with our Bayesian optimization method. Lastly, the design space also contains many undisclosed constraints. Some designs within the design space might fail to obtain benchmark performance or pass the synthesis stage for resource utilization assessment. Despite the numerous challenges outlined above, the results shown in Fig. 6 indicate that our Bayesian optimization method consistently outperforms all other optimization methods in the first experiment.

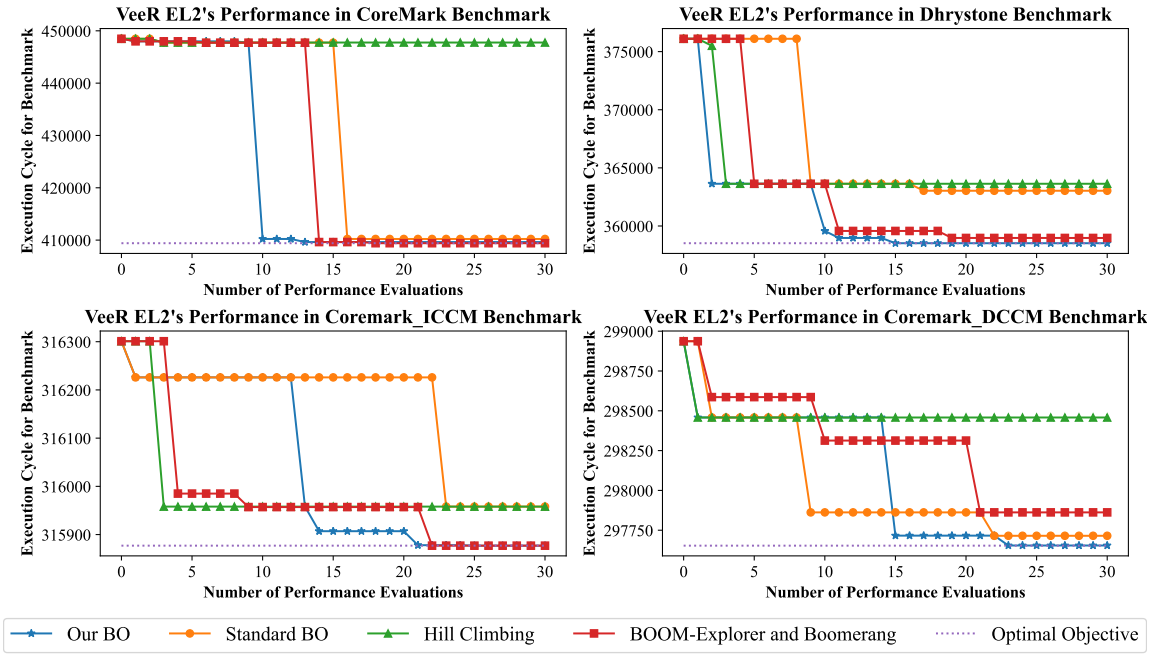


Fig. 6. Benchmark Testing with Z-7012S FPGA (32600 LUTs and 65200 FFs)

The target FPGA in the second experiment is AMD Zynq Z-7014S, which offers a larger number of LUTs and Flip-Flops. As Fig. 7 demonstrates, our Bayesian optimization method still demonstrates the best performance in customizing the VeeR EL2 core. With the slight relaxation of resource constraints, most of the customized cores could meet the resource constraints. This could reduce the probability of other Bayesian optimization methods generating invalid customized cores. The disparity between the other Bayesian optimization methods and ours might be caused by the integer-valued parameters. Analysis was taken to each sample generated by the three Bayesian optimization methods and it was observed that the standard Bayesian optimization method and the Bayesian optimization method that approximates BOOM-Explorer and Boomerang, frequently generated samples that could be rounded to the same integer values, which significantly decreased the optimization efficiency. This suggests that incorporating an integer-valued parameter handler contributes to the custom Bayesian optimization method's performance.

In both experiments, only the Bayesian optimization methods have the opportunity to reach the optimal design. The optimal designs are pre-determined by exhaustively going through all parameters selection. According to the results shown in Fig. 6 and Fig. 7, on average, within 30 iterations, with our Bayesian optimization method achieving a 75% success rate and BOOM-Explorer and Boomerang achieving 63%. The remaining methods had success rates below 50%. Additionally, our model required 23% fewer iterations than BOOM-Explorer and Boomerang to reach the optimal design, resulting in shorter optimization time.

6 CONCLUSIONS AND FUTURE WORK

This paper proposes a Bayesian optimization method to customize soft processors for optimized performance while enforcing resource utilization constraints. Additionally, we explore a technique for tuning restricted integer parameters

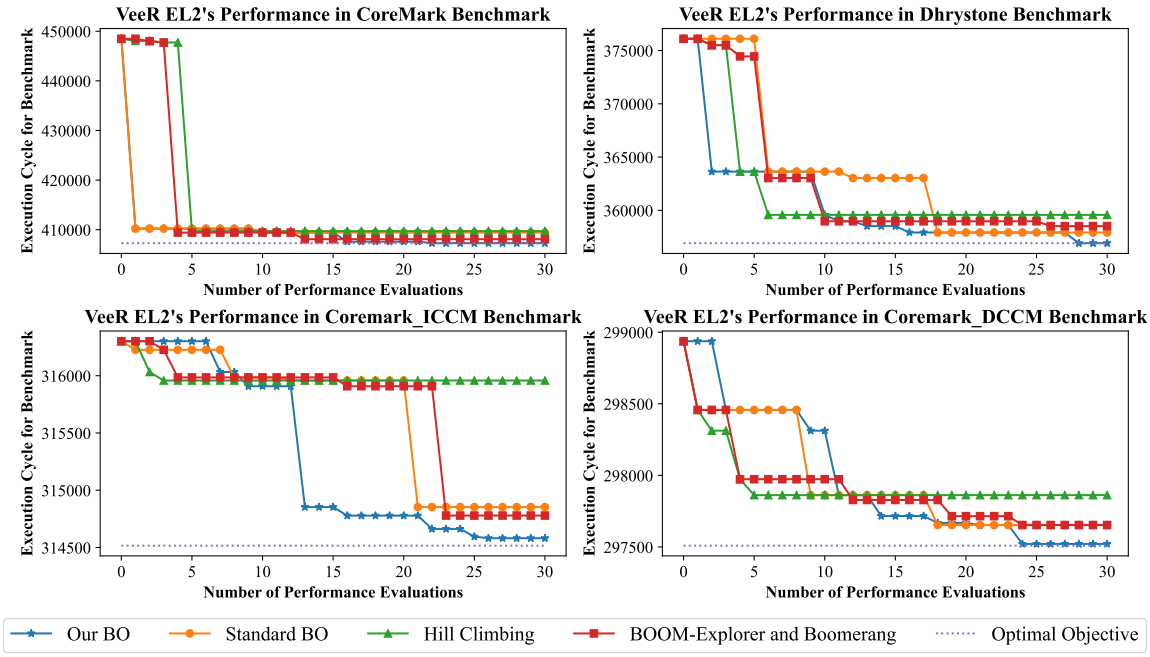


Fig. 7. Benchmark Testing with Z-7014S FPGA (40600 LUTs and 81200 FFs)

to enhance the efficiency of the optimization process. According to our experiments, the proposed approach significantly outperforms existing optimization methods. Our Bayesian optimization method has promising implications for future research and development in related fields. Integrating constraint functions directly into the optimization process could lead to more efficient FPGA-based systems beyond soft processors, such as data-flow computers and AI accelerators. Additionally, our method has the potential to support co-optimization of hardware and software, where both are optimized together.

Future work includes the following directions. First, our current Bayesian optimization approach supports only single-objective optimization. We aim to extend the model to accommodate multi-objective optimization, broadening its applicability to a wider range of real-world scenarios. Second, we plan to expand the proposed optimization approach to support various architectures, including additional soft processors, non-FPGA systems, and heterogeneous computing environments involving GPUs and FPGAs.

ACKNOWLEDGMENTS

The support of UK EPSRC (grant number EP/V028251/1, EP/S030069/1 and EP/X036006/1), Intel and AMD is gratefully acknowledged.

REFERENCES

[1] Jesús Alastruey, Teresa Monreal, Francisco Cazorla, Víctor Viñals, and Mateo Valero. 2008. Selection of the Register File Size and the Resource Allocation Policy on SMT Processors. In *2008 20th International Symposium on Computer Architecture and High Performance Computing*. ACM, Washington, DC, United States, 63–70. https://doi.org/10.1109/SBAC-PAD.2008.17

[2] Krste Asanović, Rimas Avižienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Palmer Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Benjamin Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. 2016. *The Rocket Chip Generator.* Technical Report UCB/EECS-2016-17. EECS Department, University of California, Berkeley. If used for research, please cite Rocket Chip by the technical report.

[3] Chen Bai, Qi Sun, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin DF Wong. 2021. BOOM-Explorer: RISC-V BOOM microarchitecture design space exploration framework. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, Munich, Germany, 1–9.

[4] Eric Brochu, Vlad M. Cora, and Nando de Freitas. 2010. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *CoRR* abs/1012.2599 (2010), 49 pages. arXiv:1012.2599 http://arxiv.org/abs/1012.2599

[5] Chips Alliance. 2024. Cores-VeeR-EL2: A RISC-V Core. https://github.com/chipsalliance/Cores-VeeR-EL2. Available online: https://github.com/chipsalliance/Cores-VeeR-EL2.

[6] Jose GF Coutinho, Ce Guo, Tim Todman, and Wayne Luk. 2023. Exploring Machine Learning Adoption in Customisable Processor Design. In *2023 IEEE 15th International Conference on ASIC (ASICON)*. IEEE, Nanjing, China, 1–4.

[7] Paul Feliot, Julien Bect, and Emmanuel Vazquez. 2016. A Bayesian approach to constrained single- and multi-objective optimization. *Journal of Global Optimization* 67, 1–2 (April 2016), 97–133. https://doi.org/10.1007/s10898-016-0427-3

[8] Jacob Gardner, Matt Kusner, Xu Zhixiang, Kilian Weinberger, and John Cunningham. 2014. Bayesian Optimization with Inequality Constraints. In *Proceedings of the 31st International Conference on Machine Learning (Proceedings of Machine Learning Research, 2)*, Eric P. Xing and Tony Jebara (Eds.). PMLR, Bejing, China, 937–945. https://proceedings.mlr.press/v32/gardner14.html

[9] Eduardo C. Garrido-Merchán and Daniel Hernández-Lobato. 2020. Dealing with categorical and integer-valued variables in Bayesian Optimization with Gaussian processes. *Neurocomputing* 380 (March 2020), 20–35. https://doi.org/10.1016/j.neucom.2019.11.004

[10] Maciej Kurek, Tobias Becker, Thomas CP Chau, and Wayne Luk. 2014. Automating optimization of reconfigurable designs. In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, Boston, MA, USA, 210–213.

[11] Zhen Li, Hao Zhou, Lingli Wang, and Xuegong Zhou. 2023. AMG: Automated Efficient Approximate Multiplier Generator for FPGAs via Bayesian Optimization. In *2023 International Conference on Field Programmable Technology (ICFPT)*. IEEE, Yokohama, Japan, 294–295.

[12] Yen-Fu Liu, Chou-Ying Hsieh, and Sy-Yen Kuo. 2023. Boomerang: Physical-Aware Design Space Exploration Framework on RISC-V SonicBOOM Microarchitecture. In *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, Porto, Portugal, 85–93.

[13] lowRISC contributors. 2024. Ibex Core. https://github.com/lowRISC/ibex/tree/master. Accessed: 2024-05-15.

[14] Duy Thanh Nguyen, Hyun Kim, and Hyuk-Jae Lee. 2020. Layer-specific optimization for mixed data flow with mixed precision in FPGA design for CNN-based object detectors. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 6 (2020), 2450–2464.

[15] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.