Prospector: Synthesizing Efficient Accelerators via Statistical Learning

Atefeh Mehrabi[1], Aninda Manocha[2], Benjamin C. Lee[1], Daniel J. Sorin[1]
[1]*Department of Electrical and Computer Engineering*, *Duke University*, Durham, NC, USA
[2]*Department of Computer Science, Princeton University, Princeton, NJ, USA*
atefeh.mehrabi@duke.edu, amanocha@princeton.edu, benjamin.c.lee@duke.edu, sorin@ee.duke.edu

***Abstract*—Accelerator design is expensive due to the effort required to understand an algorithm and optimize the design. Architects have embraced two technologies to reduce costs. High-level synthesis automatically generates hardware from code. Reconfigurable fabrics instantiate accelerators while avoiding fabrication costs for custom circuits. We further reduce design effort with statistical learning. We build an automated framework, called Prospector, that uses Bayesian techniques to optimize synthesis directives, reducing execution latency and resource usage in field-programmable gate arrays. We show in a certain amount of time designs discovered by Prospector are closer to Pareto-efficient designs compared to prior approaches.**

***Index Terms*—High-level synthesis, design space exploration, FPGA, Bayesian optimization**

I. Introduction

As Dennard scaling ends, the pursuit of specialized hardware for energy efficiency has become prevalent. Many platforms, including datacenters, instantiate accelerators on field-programmable gate arrays (FPGAs) to realize their benefits while avoiding the costs of custom circuits.

The need for accelerators has renewed interest in high-level synthesis (HLS) [7], a toolflow in which designers specify functionality in a high-level language and automated tools produce RTL code. Such toolflows offer large productivity gains when compared to manually writing RTL, a time-consuming process that requires expertise in digital design. But challenges remain because designers must rely on HLS directives to guide synthesis and produce quality RTL [20].

Directives are hints to the HLS tool, indicating that some code location could be optimized to improve performance. Tuning varied directives at different locations defines a vast design space. The size of the space depends on the number of lines of code, the choice of directives (e.g., loop unrolling), and the choice of settings for a given directive (e.g., unrolling factor). Furthermore, the effects of multiple directives interact to affect performance in subtle and unexpected ways. Thus, designers need automated solutions that quickly explore trade-offs between cost and performance of RTL variants.

We propose Prospector, a framework for synthesizing efficient accelerators with optimization directives. While Prospector supports both ASICs and FPGAs, this paper focuses on FPGAs given the growing interest in reconfigurable accelerators. The framework coordinates the placement and configuration of directives, seeking low execution time and efficient resource usage. Prospector achieves these goals in two ways. First, it encodes the design space so that statistical models can capture accelerator performance and FPGA costs (*e.g.*, flip-flops, lookup tables, block RAMs and digital signal processors) more effectively. Second, as HLS measurements are expensive, it samples the design space in order to reveal optimal designs more efficiently.

Prospector uses Bayesian optimization, a method starting to find success in digital design [14] [10], to judiciously collect data, incrementally train models, and efficiently optimize designs. Efficient data collection and analysis is critical because evaluating each point of the design space involves costly synthesis and place-and-route. We show that Prospector efficiently reveals design optima by running HLS measurements on a small percentage (e.g., $< 1\%$) of the whole design space. Such capabilities reflect Bayesian optimization's particular strengths, which are absent in popular search heuristics, such as genetic algorithms and simulated annealing. The following summarizes our contributions.

- **Effective Search Algorithms.** Prospector places and configures optimization directives by modeling their effect on accelerator performance and FPGA resource usage. Concise design encodings and intelligent design sampling permit the use of Bayesian optimization for HLS.
- **Efficient Resource Usage.** Prospector discovers designs that meet performance targets using fewer FPGA resources. Classic approaches discover designs that require up to 2.3× more FPGA resources.
- **Broad Pareto Frontiers.** Prospector captures high-dimensional trade-offs between performance and FPGA cost. Prior approaches reveal Pareto frontiers 1.9×, on average, more distant from the true frontier than Prospector's.

II. The Prospector Framework

We optimize accelerator design flows that use HLS to target FPGAs. HLS reduces design effort by compiling behavioral descriptions into RTL, which allows architects to simulate performance, estimate resource utilization, and verify functionality. RTL results of synthesis are used for more accurate resource usage estimates via place and route.

We develop a statistical framework, called Prospector, for identifying synthesis directives that best optimize an accelerator design. We leverage Bayesian optimization, which builds a probabilistic model that approximates an unknown function

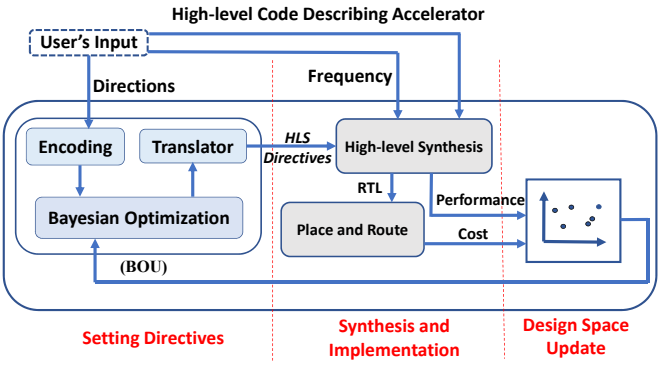

Fig. 1: Prospector Framework

and serves as its surrogate [18]. Bayesian techniques are particularly effective when the function is a black box and evaluating the function is costly. The technique iteratively samples parameter values, evaluates the function with those values, and updates the probabilistic model. Over multiple iterations, the optimization approaches optimal parameter values $\hat{x} = \arg\max_{x \in X} f(x)$ for function f.

We consider the HLS toolflow (*i.e.*, simulate, synthesize, place-and-route*) an unknown function to be modeled and optimized. The function's inputs specify the placement and configuration of synthesis directives. The function's outputs quantify design quality, which include execution time and multiple measures of FPGA resource utilization. Although we can invoke the HLS toolflow to evaluate the function for any possible set of inputs, evaluations are prohibitively expensive when exploring large, complex design spaces.

Figure 1 summarizes the Prospector framework. Its input includes high-level source code that describes accelerator functionality. It outputs RTL implementations that reflect varied performance and cost trade-offs. Within Prospector, the Bayesian optimization unit (BOU) explores the design space iteratively. In each iteration, the BOU controls the choice of synthesis directives and the HLS toolflow converts source code to RTL. After multiple iterations, Prospector identifies synthesis directives and accelerator designs that balance execution time and FPGA resource utilization.

A. Design Space

Architects use directives to specify optimizations and guide HLS but determining which directives are effective requires time-consuming simulation. The problem is complicated by two inter-related questions.

- **Placement.** Which code locations should be targeted by an optimization?
- **Configuration.** What values should an optimization's arguments take?

Both placement and configuration heavily influence optimization effectiveness and design quality. The design space size increases exponentially with the number of loops, arrays, and functions targeted for acceleration. An architect may need to try many directives because interactions between the different optimizations are difficult to anticipate. If each trial were to invoke HLS, the computation for simulation, synthesis, and place-and-route would require days if not weeks. Prospector reduces search time by exploring the design space intelligently via Bayesian optimization.

*Place-and-route is a user option for more accurate reports in Prospector, and we assume it is chosen in this paper.

B. Bayesian Optimization Overview

Statistical Model. The Gaussian process is a statistical model of an unknown function. For each input, the model estimates the function's output and the uncertainty around that estimate. The model is trained by sampling inputs, evaluating the function, and obtaining outputs. These sampled measurements supply data that refine estimates for unobserved outputs. Moreover, they reduce uncertainty around estimates for the corresponding outputs. As data become available, the Gaussian process updates its model of the function to produce increasingly accurate and confident estimates.

Figure 2 illustrates the Gaussian process. Suppose we obtain data $\{(x_1, f_1), (x_2, f_2), (x_3, f_3)\}$ by evaluating the function. We model output f_* for previously unobserved input x_* using a Gaussian random variable with mean μ_* and standard deviation σ_*. Intuitively, μ_* should be near the outputs for inputs similar to x_*, and σ_* should decrease when more outputs are observed for inputs similar to x_*.

Data Acquisition. Gaussian processes and Bayesian optimization use data sparingly. An acquisition function incrementally selects inputs for which the target function should be evaluated, producing a sequence of implemented designs (x, f). Early in the procedure, the acquisition function favors exploration and selects inputs for which the function's output is uncertain. Later, it favors exploitation and selects inputs for which the function's output is likely closer to the optimum. As the acquisition function supplies data to refine the Gaussian process, predictions become more accurate.

Figure 2 illustrates updates to the Gaussian process and the acquisition function's estimate of utility from data. As we seek to minimize execution time and FPGA resource utilization—measured in terms of the number of FFs, LUTs, DSPs, BRAMs—we use the PESMO acquisition function for multi-objective Bayesian optimization. PESMO selects inputs to reduce the estimated Pareto frontier's entropy [6].

C. Directive Encodings

We incrementally train Gaussian processes by synthesizing and evaluating a sequence of designs (x, f).

Inputs. We define input x to be a vector of variables, each of which encode the use of a synthesis directive. First, we encode directive placement using a binary variable, indicating whether a directive is applied, for each code location that could be optimized. Next, we encode directive configuration using a categorical variable to describe the optimization's mode and integer variables to specify the optimization's tunable argument.

Outputs. The synthesis of input x produces output f. We define f to be a vector of metrics relevant to our optimization

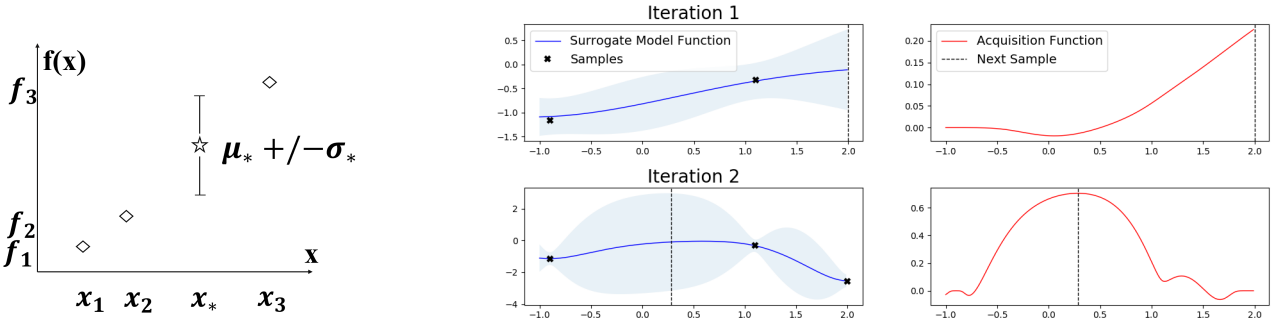

Fig. 2: Gaussian process with measurements for x_1, x_2, x_3 and prediction for x_* (left). Bayesian optimization updates model as data is acquired (center). Acquisition function selects data measurements based on expected benefits (right).

Directive	Parameter	Values
Loop Unrolling	Factor	$\forall 2x \mid x \in [1:M]$
Loop Pipelining	Initiation Interval	$\forall x \mid x \in [1:7]$
Array Partitioning	Factor	$\forall 2^x \mid x \in [1:N]$
Function Inlining	Setting	on/off
Allocation	Instance and Limit	instance $\in$ [add,mul] $\forall x \mid x \in [1:N_{add}]$ $\forall 2x \mid x \in [1:N_{mul}]$

TABLE I: Directives' ranges that define the design space. The values of M, N differ across benchmarks. See Table II.

Benchmark	Optimization Targets	Range	Design Space Size
fdtd-2d	4 loops	M=9	6,561
2mm	4 loops, 1 array	M=12, N=1	15,625
fft	9 loops, 3 arrays, 3 functions, 2 allocations	M=8, N_{add}=10 N=2, N_{mul}=8	36,000
bbgemm	2 loops, 1 array	M=16, N=4	960
stencil-3d	2 loops, 1 array	M=16, N=4	960
heat-3d	4 loops	M=16	2,304

TABLE II: Design space summary.

objective. These metrics include execution time and utilization for each type of FPGA resource including FFs, LUTs, DSPs, and BRAMs (5 dimensions).

Code Locations. Given high-level source code and the user's directions, we specify the number of possible directives D and the number of code locations L for each directive. Considering a bitmap for each directive, the map's length equals the number of potential locations L. The result is D maps with L bits each. Each bit indicates whether a directive $d \in [1, D]$ is active at location $l \in [1, L]$. We convert each directive's map into an integer between zero and $2^L - 1$. For example, if each of three loops could be unrolled, we encode the unrolling strategy with a three-bit map that corresponds to an integer between zero and seven. Thus, the bitmaps permit Gaussian processes to model and explore directive placement. Placement is monitored either via this separate integer parameter or by considering inactive state as one possible configuration value for all locations (e.g., unroll factor = 1 means the unrolling directive is off for that location).

Directive Configurations. We specify the range of possible values for each directive's categorical and numerical settings. Prospector explores and samples directive configurations within these ranges. For example, a loop could be unrolled by a facor of $2x$ where $x \in [1:4]$. Note that we sample parameters for directive placement and configuration independently.

III. Experimental Methods

Benchmarks. We evaluate application kernels from two benchmark suites, PolyBench [12] and MachSuite [13], which cover a wide range of functionality and complexity. We explore design spaces for fdtd-2d, 2mm, and heat-3d from PolyBench and fft, bbgemm, and stencil-3d from MachSuite.

Bayesian Optimization Unit (BOU). The BOU generates new values for target directives at each iteration in the optimization procedure. The BOU integrates our encoding and translation mechanisms with Spearmint [1], a software package that implements Gaussian processes and acquisition functions. The encoding unit converts a user-defined space of HLS directives and code locations into input parameters for Spearmint. Spearmint's acquisition function, which Prospector chooses as PESMO, selects parameter values, which the translation unit converts into a TCL script that describes how directives are to be located and configured for HLS. Hardware generated by HLS is used for place-and-route and performance/cost measurements. Data from the design profile updates Spearmint's Gaussian process and influences the acquisition function's subsequent selections.

High-Level Synthesis and Place-and-Route. We use Xilinx Vivado HLS to synthesize RTL from high-level code. The RTL is fed into Vivado Design Suite to perform place-and-route and generate the bitstream to program the FPGA.

Directives. HLS tools offer several directives among which the set in Table I covers those that were found most effective [15]. Without loss of generality, we focus on this list to generate our design spaces. Table II reports the number of loops, arrays, and functions that are targeted by our optimizations, the range of parameter values that were considered, and the total size of the design space.

IV. Evaluation

We evaluate Prospector's ability to find optimal design points and reveal the Pareto frontier. We first perform an exhaustive characterization of the design space by running every possible design point through HLS and place-and-route, measuring execution time and FPGA utilization, and

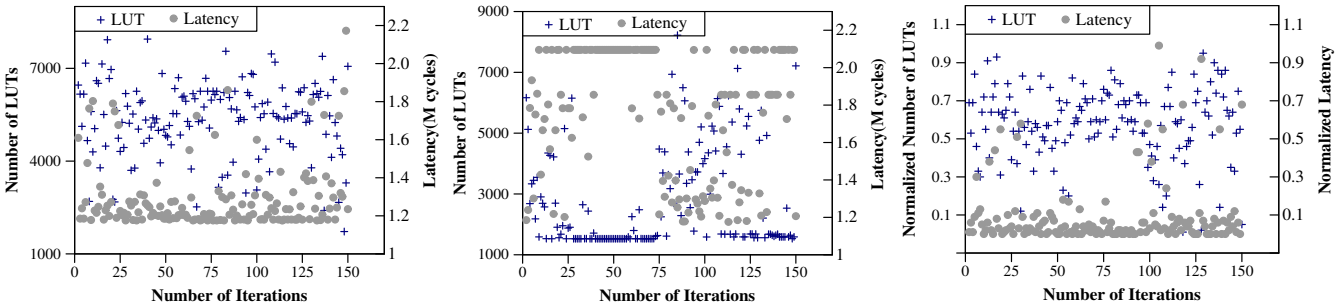

Fig. 3: Latency, LUT usage across iterations of Prospector-1D when minimizing (a) latency, (b) latency-LUT product, (c) latency-LUT product with normalized, scaled measures. Data for fdtd-2d.

identifying the "golden" Pareto optima. We then determine how closely Prospector and alternative heuristics track these golden optima.

Prospector relies on search. Thus, we compare Prospector (with both single and multidimensional Bayesian optimization), as well as popular search heuristics for design space exploration. Evaluated methods include:

- **Prospector.** Models and optimizes one or multiple objective function(s), including latency and multiple dimensions of FPGA usage, with Gaussian processes. Searches for Pareto optima based on objective(s). If optimizing all 5 dimensions of our problem (latency, FFs, LUTs, DSPs, and BRAMs), we denote it simply as Prospector. If optimizing $k < 5$ dimensions, we denote it as Prospector-kD (e.g., Prospector-2D).
- **Random Search.** Samples designs uniformly at random [3].
- **Simulated Annealing.** Samples designs that are likely to improve upon prior measurements. Samples are increasingly focused as iterations progress [16].
- **Genetic Algorithm.** Samples design populations by using the fittest designs from the previous generation of samples [5]. Searches for Pareto optima based on latency and cost.

For all methods, each iteration of the exploration algorithm is composed of parameter selection and synthesis flow for the selected parameter values, plus evaluation of the performance and cost. The runtime per iteration is completely dominated by HLS toolflow delays, which are common to all methods. Thus, we can fairly compare the different methods by giving them each the same number of iterations (or search time) and comparing the quality of the designs produced in that time.

We evaluate the ability of design frameworks to reveal the Pareto frontier, which is essential to reasoning about design trade-offs. A good Pareto frontier identifies a broad spectrum of designs for which no other design improves one metric without harming another. We find that popular techniques—simulated annealing, random search, and genetic algorithms—do not accurately describe the Pareto frontier and reveal efficient trade-offs. These techniques converge to optima poorly or slowly, get stuck in local minima, or restrict the optimization to only parts of the design space.

We find that Prospector can navigate multiple objectives, producing low-latency designs that use resources efficiently.

A. Visualizing Pareto Efficiency

We start evaluating Prospector by applying Prospector-1D. Figure 3(a) indicates that Prospector-1D explores the design space to minimize latency without regard for cost. Prospector-1D successfully converges toward optimal latency but that does not guarantee the optimal LUT usage. This outcome arises from the narrow optimization objective.

Designers often care about more than one objective (*i.e.* cost and latency). Architects sometimes account for two metrics by optimizing their product or putting constraints on one when optimizing the other [10], but defining constraints is a burden on user. We also show product optimization is a fragile solution. Figure 3(b) applies Prospector-1D on the LUT-latency product. LUT usage is reduced but latency remains high when minimizing this fused metric. LUT usage dominates the product and optimization procedure. One could normalize and re-scale metrics to overcome the range difference between latency and LUTs, but Figure 3(c) indicates that doing so does not solve the issue. Latency is reduced but LUT usage remains high. These challenges increase with the number of dimensions in the design space. Prospector overcomes this limitation by modeling and optimizing multiple objectives simultaneously.

Figure 4 shows how Prospector-2D for latency and LUT usage reveals the broad latency-LUT Pareto frontier after 50 iterations. One might hypothesize that, while Prospector-1D is insufficient, perhaps Prospector-2D suffices and there is little or no need for higher dimensional optimization. However, although Prospector-2D outperforms Prospector-1D, its results are pareto sub-optimal and fall short of coordinated analysis across all dimensions. Neglecting other FPGA resources in Prospector-2D results in missing sophisticated resource interactions and opportunities to use the FPGA more efficiently.

Table III details these limitations by presenting Prospector-2D's "Pareto optimal" designs, which do not actually satisfy criteria for optimality and miss interactions between resources in the 5D space. First, Prospector-2D misses Pareto optima within the neglected dimensions of the 5D space. We can find designs that incur the same costs for one resource but lower costs for another resource. For example, Design 5 is not Pareto optimal because Design 7 performs equally well, uses the same number of DSPs, but reduces the number of LUTs and FFs by 25% and 31%, respectively. Design 5 is Pareto dominated yet Prospector-2D discovers it when optimizing latency and DSP

Design	2-D Pareto	FFs	LUTs	DSPs	Cycles (M)
1	Latency-DSPs	4,926	7,289	32	1.175
2	Latency-LUTs	2,927	5,048	32	1.185
3	Latency-FFs	3,199	5,151	32	1.180
4	Latency-FFs	2,768	4,808	32	1.204
5	Latency-DSPs	3,810	5,536	20	1.194
6	Latency-LUTs	2,659	4,670	32	1.214
7	Latency-FFs	2,620	4,108	20	1.220

TABLE III: Each design is Pareto optimal in 2D analysis but, in the 5D space, is actually sub-optimal and obscures interactions between FPGA resource types. Data for fdtd-2d.

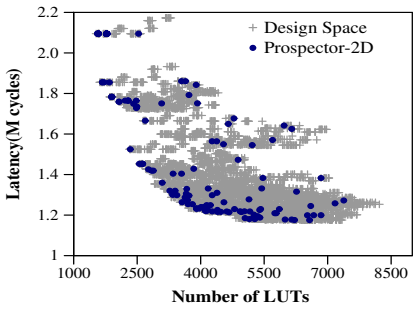

Fig. 4: Prospector-2D for latency and LUTs (fdtd-2d).

cost while neglecting other resources. Design 7 is truly Pareto optimal and discovered by Prospector.

Second, Prospector-2D misses opportunities to substitute and exchange resources in the pursuit of performance. Substitution effects are important for FPGAs, because competition for shared resources may require flexible resource requests for an accelerator. For example, Designs 5 and 6 illustrate the possibility of substituting DSPs for LUTs and FFs. We could reduce the number of DSPs by 37% and increase the number of LUTs and FFs by 18% and 43% without affecting latency. Optimization in higher dimensions is more likely to discover substitutability, mitigating resource bottlenecks and revealing multiple paths to the same performance.

Finally, Prospector-2D misses complementary resource demands that require coordinated allocation. Points in the 2D Pareto frontiers indicate that LUTs and FFs are neither substitutes nor independent. LUTs and FFs are often used in related proportions such that if LUT usage changes, so does FF usage. Comparing Design 1 to Designs 2, 3, and 4, we find that LUTs and FFs are reduced together by 30-40%. Comparing Design 5 to Designs 6 and 7, we find that LUTs and FFs are reduced together by 16-32%. These conclusions are missing from 2D analysis and require 5D analysis.

B. Quantifying Pareto Efficiency

We assess goodness by measuring the distance between two Pareto frontiers. We calculate the Euclidean distance from every design on the source frontier to the closest point on the destination frontier, producing a number of distances equal to the number of design points in the source. Note that each design point is represented by a five-dimensional vector that quantifies latency and usage for four FPGA resources. Our measures build on related work that explored several indicators to evaluate multi-dimensional Pareto frontiers [4].

Scheme	Low Latency	Medium Latency	High Latency
Prospector 2-D Latency-LUT	(0, 43, 42)	(0, 10, 5)	(0, -12, -11)
Prospector 2-D Latency-FF	(0, 3, 5)	(0, -2, -4)	(0, -12, -11)
Prospector 2-D Latency-DSP	(0, 19, 20)	(0, -2, -4)	(0, 54, 54)
Prospector 1-D	(50, 60, 58)	(100, 129, 121)	(100, 54, 59)
Genetic Algorithms	(50, 19, 18)	(100, -17, -16)	(0, 8, 8)
Simulated Annealing	(0, 53, 50)	(0, 12, 9)	(0, 20, 35)
Random Search	(0, 5, 8)	(0, 53, 48)	(0, 76, 77)

TABLE IV: Optima from other algorithms use more resources than Prospector, given same target latency and budget for optimization time. Results shown are percentage increases in resource usage (%DSP, %FF, %LUT). fdtd-2d benchmark.

When measuring distances, the source is the golden frontier identified by exhaustive evaluation of the design space, and the destination is an estimated frontier identified by Prospector or other heuristics. For each point on the golden frontier, we measure the shortest distance to any point on the estimated frontier. We mitigate the inconsistency between ranges measured for latency and FPGA resource usage by normalizing values so that they are in [0,1].

Figure 5 reports average distances, normalized to the distance between the golden and Prospector frontiers [†]. Prospector most accurately reveals the Pareto frontier and reports the shortest distances to the golden frontier. Optimization in fewer dimensions or using alternative heuristics are less accurate and report greater distance to the golden pareto frontier.

Even though distances are the most important metric, they can be difficult to interpret. For another perspective, Table IV shows how Prospector often identifies designs that use far fewer FPGA resources than those identified by other methods. The table compares resource usage for three performance targets (low, medium, high) that correspond to the 25th, 50th, and 75th percentiles in the golden frontier's latency distribution. For example, Prospector-1D's designs require 50-129% more resources than Prospector's.

V. Related Work

Statistical learning constructs surrogate models for unwieldy design flows [8], [9], [17]. These methods sample the parameter space, evaluate those design samples, and learn models for design quality. Creating a training dataset for HLS pragma optimization and constructing models for multiple figures of merit is challenging. Pre-RTL frameworks support early-stage accelerator design (e.g., Aladdin [19]), but the architect must still perform design space exploration, relying on expert design or tuned directives to identify efficient implementations.

Heuristics search the design space defined by tunable knobs. Random search samples designs randomly [3]. Simulated annealing (SA) [16] and gradient descent judiciously sample designs likely to improve upon prior measurements but may discover local minima [14]. Genetic algorithms (GA) use the fittest designs in a population of samples to produce the next generation of samples [2], [11]. Prospector outperforms these heuristics for HLS and FPGAs' large parameter spaces.

[†]HLS automatically allocates FFs and BRAMs, sometimes using only FFs. The BRAM-latency bar is shown only for fft, which required BRAMs.

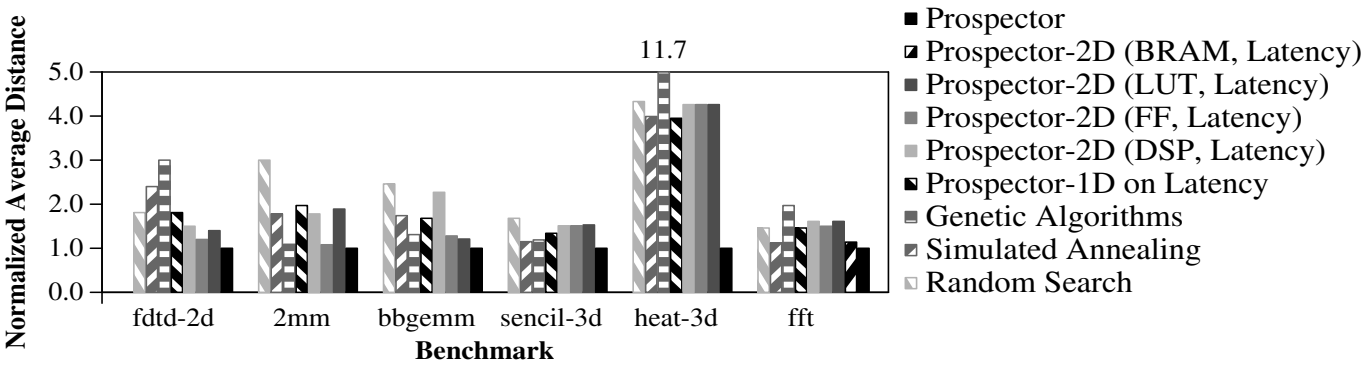

Fig. 5: Distances between golden and estimated Pareto frontiers, normalized to that from Prospector.

Bayesian optimization can tune directives to minimize design latency [10]. This work explores only the placement, but not configuration, of HLS directives due to its limits in its design encoding. Furthermore, it suggests fusing metrics into a single objective (*i.e.*, latency-LUT product), which we show is problematic, to leverage standard data collection procedures. Bayesian optimization has also tuned neural networks, optimizing both hyperparameters and continuous hardware parameters such as operand bit width [14]. This study does not consider HLS directives needed to instantiate accelerators on reconfigurable hardware.

Analytical models explore simpler parameter spaces and do not extend easily to directive placement and configuration for multiple interdependent code targets. Some models study directives for a single code target [22] or optimize directives to reduce usage for specific FPGA resource types such as BRAMs and DSPs [21]. None of these approaches model directive placement and configuration for multiple HLS code targets and comprehensive FPGA design spaces.

VI. Conclusion

Prospector uses multi-dimensional Bayesian optimization to efficiently search large accelerator design spaces defined by HLS directives. Despite the plethora of possible HLS directive placements and configurations and multiple metrics optimized, Prospector efficiently finds Pareto optimal designs. Prospector is much more effective than alternative search heuristics.

Acknowledgment

This work is supported by National Science Foundation grants CCF-1149252, CCF-1337215, SHF-1527610, and AF-1408784. This work is also supported, in part, by the Semiconductor Research Corporations Global Research Collaboration (GRC) program under task 2821.001.

References

[1] *Spearmint: https://github.com/HIPS/Spearmint/tree/PESM*.

[2] G. Ascia, V. Catania, and M. Palesi, "A multiobjective genetic approach for system-level exploration in parameterized systems-on-a-chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 4, pp. 635–645, 2005.

[3] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, 2012.

[4] P. Bosman and D. Thierens, "The balance between proximity and diversity in multiobjective evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, 2003.

[5] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, 2002.

[6] D. Hernández-Lobato, J. Hernandez-Lobato, A. Shah, and R. Adams, "Predictive entropy search for multi-objective bayesian optimization," in *Proc. International Conference on Machine Learning*, 2016.

[7] D. Koeplinger, R. Prabhakar, Y. Zhang, C. Delimitrou, C. Kozyrakis, and K. Olukotun, "Automatic generation of efficient accelerators for reconfigurable hardware," in *Proc. International Symposium on Computer Architecture (ISCA)*, 2016.

[8] B. Lee and D. Brooks, "Illustrative design space studies with microarchitectural regression models," in *Proc. International Symposium on High-Performance Computer Architecture (HPCA)*, 2007.

[9] H. Liu and L. Carloni, "On learning-based methods for design-space exploration with high-level synthesis," in *Proc. Design Automation Conference (DAC)*, 2013.

[10] C. Lo and P. Chow, "Model-based optimization of high level synthesis directives," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, 2016.

[11] M. Palesi and T. Givargis, "Multi-objective design space exploration using genetic algorithms," in *Proceedings of the tenth international symposium on Hardware/software codesign*. ACM, 2002, pp. 67–72.

[12] L. Pouchet, "Polybench: The polyhedral benchmark suite," *PolyBench: http://www. cs. ucla. edu/pouchet/software/polybench*, 2012.

[13] B. Reagen, R. Adolf, Y. Shao, G. Wei, and D. Brooks, "Machsuite: Benchmarks for accelerator design and customized architectures," in *Proc. International Symposium on Workload Characterization*, 2014.

[14] B. Reagen, J. Hernández-Lobato, R. Adolf, M. Gelbart, P. Whatmough, G. Wei, and D. Brooks, "A case for efficient accelerator design space exploration via bayesian optimization," in *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, 2017.

[15] B. Reagen, Y. Shao, G. Wei, and D. Brooks, "Quantifying acceleration: Power/performance trade-offs of application kernels in hardware," in *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, 2013.

[16] B. Schafer, T. Takenaka, and K. Wakabayashi, "Adaptive simulated annealer for high level synthesis design space exploration," in *Proc. International Symposium on VLSI Design, Automation and Test*, 2009.

[17] B. Schafer and K. Wakabayashi, "Machine learning predictive modelling high-level synthesis design space exploration," *IET Computers & Digital Techniques*, 2012.

[18] B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. D. Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, 2016.

[19] Y. Shao, B. Reagen, G. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *Proc. International Symposium on Computer Architecture (ISCA)*, 2014.

[20] F. Winterstein, S. Bayliss, and G. Constantinides, "High-level synthesis of dynamic data structures: A case study using vivado hls," in *Proc. International Conference on Field-Programmable Technology*, 2013.

[21] J. Zhao, L. Feng, S. Sinha, W. Zhang, Y. Liang, and B. He, "Comba: A comprehensive model-based analysis framework for high level synthesis of real applications," in *Proceedings of the 36th International Conference on Computer-Aided Design*, 2017.

[22] G. Zhong, A. Prakash, Y. Liang, T. Mitra, and S. Niar, "Lin-analyzer: a high-level performance analysis tool for fpga-based accelerators," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 136.