

Autotuning FPGA design parameters for performance and power

Azamat Mametjanov*, Prasanna Balaprakash*[†], Chekuri Choudary[‡], Paul D. Hovland*, Stefan M. Wild*, and Gerald Sabin[‡]

* *Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA*

email: {azamat, pbalapra, hovland, wild}@mcs.anl.gov

[†] *Leadership Computing Facility, Argonne National Laboratory, Argonne, IL 60439, USA*

[‡] *RNET Technologies, Inc., Dayton, OH 45459, USA*

email: {cchoudary, gsabin}@rnet-tech.com

Abstract—Many factors affect the performance and power characteristics of FPGA designs. Among them are the optimization parameters for synthesis, map, and place-and-route design tools. Choosing the right combination of these parameters can substantially lower power requirements, while still satisfying timing constraints. Finding such an improvement, however, requires significant experimentation by the FPGA designer. Exhaustive search through the parameter space is an automated alternative to experimentation but is intractable because of the large search space and the long build time of each parameter combination. In this paper, we propose a machine-learning-based approach to tune FPGA design parameters. It performs sampling-based reduction of the parameter space and guides the search toward promising parameter configurations. In our experiments, such selective sampling finds parameter configurations that meet the timing constraints and are within 0.2% of the optimal power consumption. Furthermore, these configurations are found in an order of magnitude less time compared with exhaustive search. Such speedups can substantially alleviate bottlenecks in the FPGA design process.

Keywords—field programmable gate arrays; tuned circuits; optimal design and tuning; power optimization

I. INTRODUCTION

Field programmable gate arrays (FPGAs) have experienced an exponential growth in the past three decades. When FPGAs were introduced in the mid-1980s, the Xilinx XC2064 FPGA had 64 lookup tables (LUTs) [1], [2]. Current-generation FPGA chips such as Xilinx Virtex-7 offer 2,000,000 logic cells (i.e., LUTs), a >30,000X increase in logic capacity [3]. Such an increase enables implementation of highly complex system-on-a-chip designs by, for example, embedding the entire 1 GHz dual-core ARM Cortex-A9 processor in a device’s programmable logic fabric [4].

For programming such complex devices, design tools play a critical role in the compilation of design specifications and constraints into bit-streams to configure FPGAs. A typical design work flow invokes synthesis, map, and place-and-route tools. The placed-and-routed design undergoes area, timing, and power analyses to determine whether the design fits into available resources, whether it operates at the expected frequency, and what the expected power consumption is. Each of the build tools provides parameters that control its behavior in terms of metrics such as power, execution

speed, and build speed. While the initial development of an FPGA design may leave tool options at default values, optimizing redesign iterations may involve tightening of area and timing constraints and/or tuning of tool options in order to improve the power consumption.

Manual search for the best parameter values is impractical, and thus there exist tools to explore a user-defined parameter space by empirically evaluating area, timing, and power metrics by recompiling a design with each parameter configuration. Because of the time needed to rebuild and re-analyze a design for each configuration, however, automated exploration even for small parameter spaces may take days or weeks. If the expected time to optimize a design exceeds the allotted tuning time budget, one can be left with an FPGA design whose performance is far from optimal.

In this paper, we describe an automated tuning (autotuning) approach to finding the best parameter configuration. The primary contributions of this approach are the following:

- User-defined parameter space is selectively sampled instead of exhaustively enumerated to find the feasible configuration with the lowest power metric. The resulting tuning time is reduced by an order of magnitude in our experiments.
- The autotuning produces a machine-learning model over the parameter space. The model of how design parameters affect the metrics of interest enables “what-if” analysis of parameter customization in presence of trade-offs among area, speed, and/or power.

In Section II we present an overview of the FPGA design flow, and in Section III we review prior work. Our approach is outlined in Section IV, and the results are discussed in Section V. In Section VI we summarize our conclusions.

II. OVERVIEW OF FPGA DESIGN

In this section, we summarize the essential elements of FPGA design, including the architecture, design flow, and tools for programming FPGAs.

FPGAs consist of an array of logic blocks interconnected with memory banks, special-purpose circuits, and I/O blocks using routed wires [5]. The logic blocks consist of multiple, interconnected basic logic elements. Each element

consists of a LUT and a state-storing flip-flop (register). The logic blocks transform input to outputs depending on the programmed state of the registers. The interconnecting channels consist of multiple segmented wires of varying length connected in a fixed or switched manner. The width, length, and connectivity of a channel provide for flexible interconnection of logic blocks. This flexibility is also a source of complexity addressed by placement and routing tools. An essential component is a clocking network that connects logic blocks to the clock signal, which drives the computation state of a circuit. Because of signal propagation delays, chip area and composition of logic blocks and channels determine the maximum clock frequency of a chip.

A. Design Flow

An FPGA design typically involves a sequence of design flow steps. We consider the Xilinx command-line build flow; other tools, such as IDEs or tools from other vendors, use similar processes. The advantage of command-line tools is the complete control over the build process by using various options available in each tool to customize its processes.

Synthesis: This process converts HDL code into a netlist – a gate-level description of the source code. This involves architecture-independent control-flow and data-flow analysis and optimization to improve the performance and memory footprint of the code.

Mapping: This process converts a logical netlist into a physical circuit description using device-specific components such as embedded memory blocks and special-purpose logic blocks (e.g., multipliers).

Place and Route: A mapped netlist is placed onto specific logic blocks of a device, and routing is defined. This is an iterative process that can include restructuring or replication of logic to minimize interconnect delays.

Area, Timing, and Power Analysis: Having obtained a complete physical circuit description, these processes determine the area, timing, and power metrics of the design.

Bitstream Generation: This process creates a bitstream file that can be loaded onto an FPGA to program the device.

B. Design Problem

Given the high-level program P_{src} and constraint values $C = \{C_{\text{area}}, C_{\text{time}}, C_{\text{pwr}}\}$, the task of an FPGA designer is to either tune the source program P'_{src} or the build tools

$$B = \{B_{\text{syn}}, B_{\text{map}}, B_{\text{par}}, B_{\text{area}}, B_{\text{time}}, B_{\text{pwr}}\}$$

such that the low-level, placed-and-routed program

$$P_{\text{par}} = B_{\text{par}}(B_{\text{map}}(B_{\text{syn}}(P'_{\text{src}})))$$

satisfies constraints such as

$$B_{\text{area}}(P_{\text{par}}) \leq C_{\text{area}}, B_{\text{time}}(P_{\text{par}}) \leq C_{\text{time}}.$$

In practice, engineering of a feasible design goes through multiple iterations, with constraints being prioritized in

order of importance: area constraints, followed by timing constraints (a timing closure problem). Power is an objective that is usually minimized after satisfying the constraints.

C. Design Optimization

The design constraints specify minimal criteria for a satisfactory design. Hence, significant gains can be realized by searching for an optimal design among all the designs that satisfy the constraints. When posed as an optimization problem, for example, where power consumption is minimized, the goal is to find a design $P_{\text{par}}^{\text{opt}}$ that has the lowest power metric among all possible (semantically identical) versions that satisfy the area and time constraints. That is, $P_{\text{par}}^{\text{opt}}$ belongs to the set of satisfactory designs

$$\mathcal{P}_{\text{satisfactory}} = \left\{ P_{\text{par}}^i : \begin{array}{l} B_{\text{area}}(P_{\text{par}}^i) \leq C_{\text{area}} \quad \text{and} \\ B_{\text{time}}(P_{\text{par}}^i) \leq C_{\text{time}} \end{array} \right\}$$

and

$$B_{\text{pwr}}(P_{\text{par}}^{\text{opt}}) \leq B_{\text{pwr}}(P_{\text{par}}^i) \quad \forall P_{\text{par}}^i \in \mathcal{P}_{\text{satisfactory}}. \quad (1)$$

D. Optimization Parameters

Each of the design tools provides a set of options that affect its output metrics. The designer can change the options' default values (underlined below) to obtain better area, timing, and power metrics. We summarize some of the options that affect trade-offs between timing and power or between build and execution speed [6].

1) *Synthesis:* The synthesis tool `xst` has two options that affect the timing and area metrics:

- *-opt_mode speed|area* The optimization mode switch specifies the strategy that should be used in synthesis. The default value of *speed* reduces the number of logic levels and therefore increases the frequency, while *area* reduces the total number of gates used and hence the required area.
- *-opt_level 1|2* This flag defines how much effort should be spent on reducing the number of logic gates. A higher value leads to increased synthesis build time.

2) *Mapping:* The mapping tool `map` translates a logical design into a physical circuit description. Some of the most relevant optimization options are as follows:

- *-global_opt off|speed|area|power* Performs the specified optimization on a fully assembled netlist prior to any mapping.
- *-timing* Instructs to perform packing and placement in the `map` process, instead of exclusively in `par`, which can improve the performance.
- *-power off|on|high|xe* Performs power-optimized placement of gates onto logic elements to reduce switching logic and capacitive power loading on data and clocking nets. This option controls the trade-off in increased area and reduced performance for lower dynamic power to drive circuit signals.

- *-logic_opt off/on* Performs optimization of timing-critical connections by restructuring their gates and rerunning synthesis, placement, and timing.

3) *Place and route*: The place-and-route tool `par` takes the output of `map` and creates a complete physical circuit description that can be used to generate a bitstream.

- *-ol std/high* Defines the overall effort level to achieve better results through greater process run time. Similar options are available for placer effort level (*-pl*) and router effort level (*-rl*).
- *-power off/on* Optimizes the dynamic capacitance of non-timing-critical design signals.

This small set of parameters generates a discrete space of 1,024 ($2^2 \times 2^6 \times 2^2$) possible parameter configurations. There are other parameters, such as `map` and `par` cost tables (random seeds used for placement), which can affect design performance and whose values are integers in $\{1, \dots, 100\}$. Since a typical design build time ranges from several minutes to several hours, an exhaustive search of the entire parameter space for the best configuration is not practical. However, there exist techniques, which we review next, for finding the optimal configuration while considering the reduced search space.

III. PRIOR WORK

Existing design tools provide the capability of exploring the design space in order to meet or optimize the timing constraints. We focus on design exploration with Xilinx SmartXplorer, but other tools such as Altera’s Design Space Explorer use similar techniques [7], [8].

Each design variant is represented as a set of tool options (parameter configurations) to synthesize and implement the design in order to satisfy particular area and speed constraints or to minimize power. The build of different variants can be performed in parallel on multiple compute cores or nodes to reduce the overall search time.

A user can search for the best parameter configuration using the built-in search space specification or modify the parameters and explore the space using custom specification. Figure 1 illustrates the format of specifying the customized set of parameters to explore. Given a custom parameter set, the tool performs a two-stage design exploration. In the first stage, synthesis variants are built with default `map-par` options optimized for the shortest build-time, and the timing metric is obtained for each variant. In the second stage, implementation variants are built with the synthesized netlist, which has the lowest timing metric that was arrived at in the first stage. The parameter configuration that has the lowest timing metric in this two-stage exploration is chosen as the best one for a given device architecture.

If two or more parameter configurations result in an identical timing metric, the configuration with the lowest expected power consumption is picked. If two or more

```

{"virtex6": {
  "XST options": (
    {"name": "my_xst1",
     "xst": "-opt_mode speed -opt_level 1"},
    {"name": "my_xst2",
     "xst": "-opt_mode area -opt_level 2"},
    ...
  ),
  "Map-Par options": (
    {"name": "my_impl1",
     "map": "-timing -ol high -xe n -global_opt on
            -retiming on",
     "par": "-ol high"},
    {"name": "my_impl2",
     "map": "-timing -ol high -xe n",
     "par": "-ol high"},
    ...
  ),
}, "spartan6": ...
}

```

Figure 1. Sample SmartXplorer search space specification. Users specify parameters and allowable values to search for better timing or lower power.

configurations have identical timing and power metrics, then the configuration with the shortest build-time to synthesize and implement the design is chosen as the best one.

The ability to customize the search space of possible design variants, combined with parallelization of search, provides a user complete control over design space exploration. However, this does not mitigate the challenges presented by the large-scale design space; because of the number of possible tool options and the values that they can take, the size of the search space is very large.

To do a preliminary test of how quickly one could find the best parameter configuration, we used a toy example of a cyclic redundancy check design and generated a search space of 8,000 configurations that included all optimization parameters summarized above. Searching for the best parameter configuration with 16-way concurrency on a 16-core Intel Xeon E5620 2.40GHz machine took 10 days. Since a build of the CRC design variant with one parameter configuration takes about two minutes, the cost of exhaustive search is disproportionately high. Further, since the best parameter configuration may vary from one design problem to another, the search cost might not be amortizeable over multiple designs. Therefore, we seek a search method that can arrive at a near-optimal configuration within several hours (e.g., using overnight or weekend idle compute hours).

IV. APPROACH

Our approach to reducing the search time is based on the hypothesis that one can quickly arrive at models that can predict the timing and power metrics based on the build tool parameter configuration. In other words, we would like to build functions $\mathcal{M}_{\text{time}}$ and \mathcal{M}_{pwr} such that

$$\begin{aligned}
 |\mathcal{M}_{\text{time}}(I_{\text{param}}) - B_{\text{time}}(P'_{\text{par}})| &\leq \epsilon_{\text{time}}, \\
 |\mathcal{M}_{\text{pwr}}(I_{\text{param}}) - B_{\text{pwr}}(P'_{\text{par}})| &\leq \epsilon_{\text{pwr}},
 \end{aligned}$$

where I_{param} is the build tool parameter configuration, P'_{par} is the design built with tools set at values specified by I_{param} configuration, and ϵ_* is the tolerance error rate between actual and predicted metrics.

The models are built by randomly sampling the parameter space, evaluating the function being modeled at sampled points (i.e., building a design with tool parameters set to sampled values and obtaining timing and power metrics of the built design), and building a correlation function between inputs and outputs. If the derived models predict the actual timing and power within an acceptable error range, they can be used as surrogate functions over the parameter space.

Our initial goal is to obtain models with error rates of at most 5% that can be obtained by sampling no more than 5% of the parameter space. The intended 20x reduction in the portion of the search space that must be executed represents significant savings, which directly translate into the feasibility of the search completing within a reasonable tuning time budget expected by the designers in the field. The caveat of sampling is the possibility of missing a configuration that substantially outperforms the other parameter configurations. However, we hypothesize that such scenarios occur with low probability and can be guarded against by increasing the number of sampled points, which would lower the error rate and thereby improve the nearness of the optimality.

A. Problem and Experimental Setup

To test our approach, we performed model-building experiments and compared results obtained by the models and by exhaustive enumeration of the same parameter space. Below we summarize the testing environment of the experiments.

To approximate practical FPGA design scenarios, we chose a sample FPGA design of a serial digital interface (SDI) stream pass-through processor of uncompressed digital video with embedded audio [9]. Such streams are often used in professional broadcast studios and video production centers. The SDI design can process streams in various modes (SD, HD, 3G) and formats (NTSC/PAL for SD, frame rate for other modes). The design is a foundation of various video processing functions such as compression, scaling, deinterlacing, frame timing, and buffering.

Our test platform consists of a Xilinx ML605 base board that includes a Virtex-6 XC6VLX240T-1FFG1156 FPGA chip, which contains over 240,000 LUTs, 37,680 CLBs, 768 DSP blocks, 15 Kb of on-chip block RAM, and 720 I/O blocks [10].

After building the design in its default configuration, we performed complete enumeration of several design tool parameters. This produced approximately 8,000 configurations for use in the model-based search: i.e., DT below. In addition, the SDI design uses IPCore modules, which are reusable logic blocks that can be called from a design for a given library function. IP Cores also provide tunable parameters. Modifying these parameters affects the power

Algorithm 1 Our model-based search framework.

Input: Parameter configuration pool \mathcal{X}_p , batch size bs , max evaluations n_{max}

```

1  $\mathcal{X}_{\text{out}} \leftarrow$  sample  $\min\{bs, n_{\text{max}}\}$  distinct configurations
  from  $\mathcal{X}_p$ 
2  $\mathcal{Y}_{\text{out}} \leftarrow$  Evaluate_Parallel( $\mathcal{X}_{\text{out}}$ )
3  $\mathcal{M} \leftarrow$  fit( $\mathcal{X}_{\text{out}}, \mathcal{Y}_{\text{out}}$ )
4  $\mathcal{X}_p \leftarrow \mathcal{X}_p - \mathcal{X}_{\text{out}}$ 
5 for  $i \leftarrow 1$  to  $\lfloor n_{\text{max}}/bs \rfloor$  do
6    $\mathcal{Y}_p \leftarrow$  predict( $\mathcal{M}, \mathcal{X}_p$ )
7    $x_i^{bs} \leftarrow$  select  $bs$  best configurations from  $\mathcal{X}_p$ 
8    $y_i^{bs} \leftarrow$  Evaluate_Parallel( $x_i^{bs}$ )
9    $\mathcal{X}_{\text{out}} \leftarrow \mathcal{X}_{\text{out}} \cup x_i^{bs}$ ;  $\mathcal{Y}_{\text{out}} \leftarrow \mathcal{Y}_{\text{out}} \cup y_i^{bs}$ 
10   $\mathcal{M} \leftarrow$  fit( $\mathcal{X}_{\text{out}}, \mathcal{Y}_{\text{out}}$ )
11   $\mathcal{X}_p \leftarrow \mathcal{X}_p - x_i^{bs}$ 
12 end for
13  $x_{\text{best}} \leftarrow$  best (feasible) configuration(s) from  $\mathcal{X}_{\text{out}}$ 

```

Output: Best parameter configuration x_{best}

metrics of the overall design. To understand their impact we enumerated the parameters and obtained approximately 2,400 configurations. We used this parameter set for a second experiment of model-based search: i.e., IPC below. Overall, exhaustive enumeration found a parameter configuration that builds a design, whose expected power consumption is 10% lower than the one built with default settings (timing closure is achieved in both cases).

V. MODEL-BASED SEARCH

Our search framework consists of sampling a small number of input parameter configurations, empirically evaluating the build processes with sampled parameter values to obtain the corresponding output metrics, and fitting a surrogate model over the input-output space. The surrogate model is then iteratively refined in the promising input parameter region by obtaining new output metrics at unevaluated input configurations predicted to be high-performing by the model.

The search framework is shown as Algorithm 1. The symbols \cup and $-$ denote set union and difference operators, respectively. Given a pool \mathcal{X}_p of unevaluated configurations, a tuning budget expressed in the form of the maximum number n_{max} of allowed evaluations, and sample size bs , the algorithm proceeds in two phases. In the initialization phase, the algorithm first samples bs configurations at random and evaluates them in parallel to obtain their corresponding output values. A surrogate modeling method uses these points to build predictive models. In the iterative phase, the algorithm predicts the outputs of all remaining unevaluated configurations using the model, evaluating the

best bs configurations, and retraining the model with the results.

When the evaluation output is a vector rather than a scalar, we can fit one model for each output. In this case, the search problem becomes: (1) a constrained single-objective problem, where only one output needs to be optimized subject to some constraints on other outputs; (2) a multi-objective problem, where all the outputs need to be optimized without a priori weight for each output; or (3) a constrained multiobjective problem, where more than one output needs to be optimized subject to some constraints on the other outputs. Without loss of generality, Algorithm 1 can be used in all these scenarios. Note that for the multiobjective case, the best configurations are typically defined in terms of a Pareto front (see, e.g., [11]).

A. Search Results

In this section, we demonstrate the applicability of the proposed search framework on the IPCore parameter dataset (IPC) and design-tool parameter dataset (DT). IPC is a single-objective search problem with the goal of minimizing power. DT represents a constrained single-objective search problem with the goal of minimizing the power subject to the constraint that the timing metric should be less than 1.0: i.e., timing constraints are satisfied. Therefore for DT, Algorithm 1 builds models of the power and timing metrics, but these models are for different purposes: the timing model is used to predict points that satisfy the constraints.

For the surrogate modeling in Algorithm 1, we use statistical machine-learning methods [12]. We experimented with a number of supervised-learning methods. Although random forest [13], cubist [14], and support vector machines [15] produced promising results, we found linear regression sufficient for modeling and identifying promising regions for power optimization. The main advantage of linear regression is its simplicity: a model is obtained with the least squares approach that minimizes the sum of squared differences between each observed and fitted value of the model. We note that the framework does not restrict the user from using nonlinear models within the search; support vector machines with a nonlinear kernel or random forests can be adopted for search when linear models prove ineffective. To enable modeling, we apply feature binarization to convert categorical values (e.g., the `map's -power` parameter values $\in \{off, on, high, xe\}$) to binary parameters. For the timing model, we use the cubist method – a recursive partitioning algorithm with boosting [14]. This allows our implementation of Algorithm 1 to quickly identify the points that violate timing constraints, minimize power, and break ties between identical power values using worst case timing slack.

For the experimental analysis, we set the maximum evaluations, n_{max} , to 1,000 and vary the batch size $bs \in \{10, 25, 50\}$. To minimize the effects of randomness in the initialization phase and the machine-learning methods, we repeated the search 10 times for each combination of

problem and batch size. To evaluate the effectiveness of the search algorithm, we measure the percentage away from the optimal point found by exhaustive enumeration. This is given by $100 \times \frac{y_{best,k} - y_{opt}}{y_{opt}}$, where $y_{best,k}$ and y_{opt} are the minimal power value found until iteration k and the optimal power value, respectively.

Figure 2 shows the results of the model-based search for the experiments. In each plot, for an iteration k , we show the distribution of $y_{best,k}$ with a box-whisker from 10 runs. From the distribution, we also plot the best, median, and worst-case power values. In practice, the search algorithm will be run only once; therefore, we focus on the worst-case results. For IPCores, the search algorithm finds the optimal in 45, 15, and 6 iterations for batch sizes 10, 25, and 50, respectively. Clearly, increasing the batch size has a positive effect on the worst-case performance of the algorithm, but has a relatively minor effect on the median and best-case performance. Each iteration denotes an SDI design build that takes 20–25 minutes. The batching allows for a higher degree of parameter space exploration and increases the probability of finding high-quality configurations in fewer iterations. Given a multinode computing environment (even a small cluster) that can perform bs evaluations at the same time, and assuming each evaluation requires the same wall clock time, the search time will be significantly reduced (especially for large batch sizes). This trend is similar for DT except that in the worst case, the search can find the configurations whose power values are 0.2% away from optimal. Here, the near-optimal point is found in 40, 9, and 6 iterations. We conjecture that DT is harder to model than IPC because of the timing constraint and the size of the parameter space.

Overall, model-based search provides substantial savings in tuning time. Exhaustive enumeration of 2,400 IPCore parameter configurations in concurrent batches of 10, 25, and 50 would need 240, 96, and 48 design builds. Model-based search provides 5x, 6x, and 8x speedups. Exhaustive enumeration of 8,000 design tool parameter configurations would require 800, 320, and 160 design builds. Model-based search provides 20x, 35x, and 26x speedups.

B. Sensitivity and Modeling Results

Typically, search algorithms do not give insights into the impact of the input parameters and the relationship between the inputs and outputs. Therefore, we devised a subsidiary procedure that takes the parameter configurations and their resulting timing and power metrics to develop surrogate models of input-output relationships over the entire input parameter space. For this, we used 1,000 evaluations ($\mathcal{X}_{out}, \mathcal{Y}_{out}$), solely from the evaluations performed in the search stage. The main caveat is that since the empirical evaluations are biased by the search algorithm toward the promising regions of the space, the obtained models from these points may not accurately capture the entire, global input space.

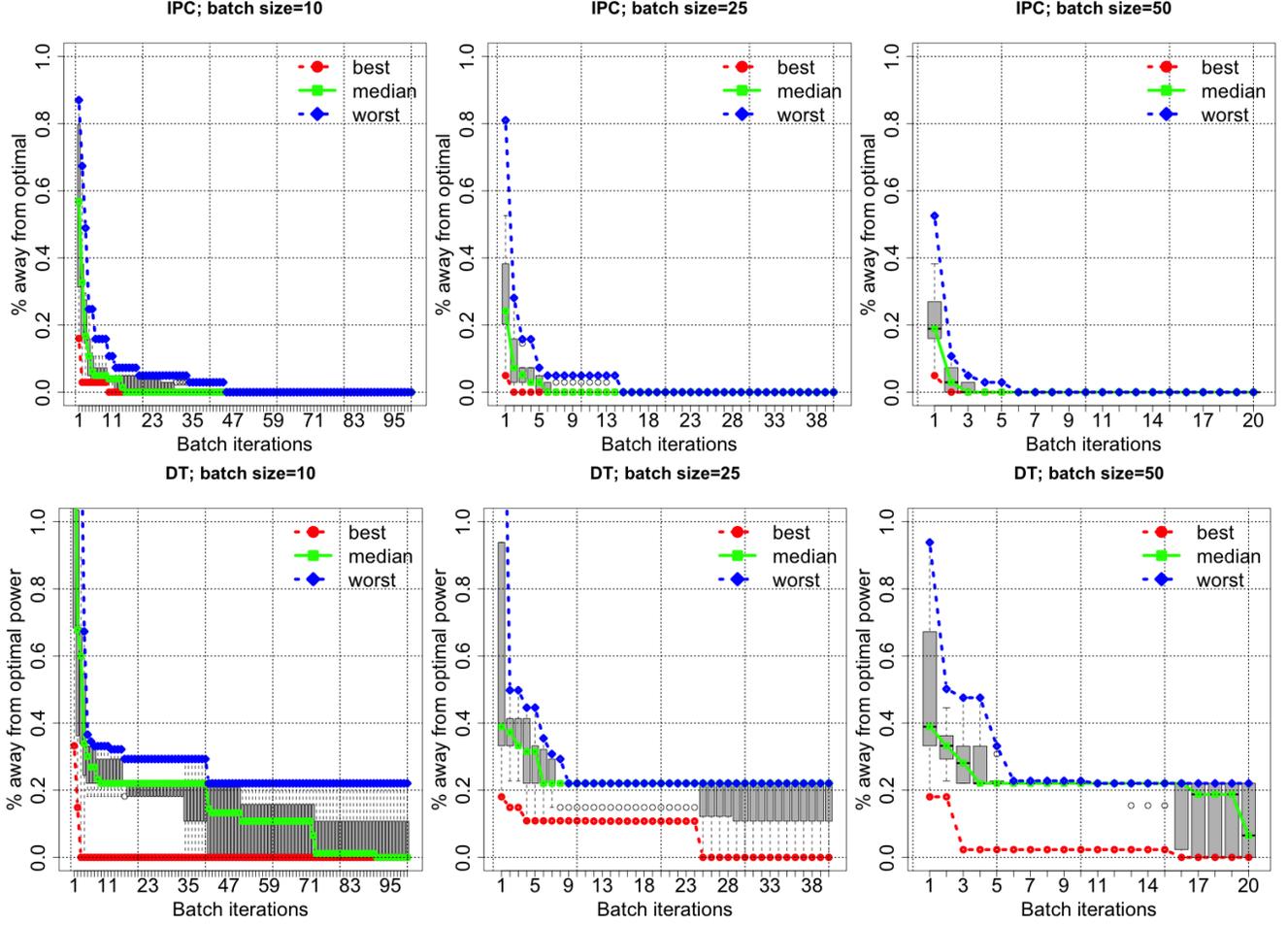


Figure 2. Search trajectory of best feasible configuration for IPC (row 1) and DT (row 2) when using concurrent evaluation batch sizes of $bs = 10, 25,$ and 50 . For IPC, the best parameter configuration is found in 45, 15, and 6 concurrent design builds. For DT, the near-optimal parameter configuration is found in 40, 9, and 6 parallel design builds.

The development of global surrogate models proceeds in two stages. The first stage is sensitivity analysis, where we reduce the dimensionality of the input space by analyzing the impact of the input parameters on the outputs and removing uncorrelated inputs. For this purpose, we use the random forest method, a state-of-the-art machine learning approach that constructs a large number of decision trees at training time [13]. For an unseen input, each tree predicts a value of the desired output; the final output is an average over all the trees. First, the random forest model is fit on the training set. The mean squared error (MSE) on the training set is given by $\frac{1}{l} \sum_{i=1}^l (y_i - \hat{y}_i)^2$, where l is the number of training points and y_i and \hat{y}_i are the observed value from the simulation and predicted value from the random forest model at the input parameter configuration x_i , respectively. In order to assess the impact of an input parameter m , the values of m in the training set are randomly permuted. Again, a random forest model is fit on this permuted training set, and the MSE is computed. This procedure is repeated for a number of random permutations. If a parameter m

is important, permuting the values of m should affect the prediction accuracy significantly, resulting in a large increase in the MSE. A permutation example is shown below. To assess the importance of the first parameter in the training set represented by the matrix \mathbf{T} , we permute the values of the first column (corresponding to the values of the first parameter) in the matrix, shown in \mathbf{T}'_1 .

$$\mathbf{T} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 & \dots & \dots \\ 6 & \ddots & \vdots \\ 7 & \dots & \dots \end{bmatrix} \quad \mathbf{T}'_1 = \begin{bmatrix} 7 & \dots & \dots \\ 5 & \ddots & \vdots \\ 6 & \dots & \dots \end{bmatrix}$$

Figure 3 shows the most sensitive input parameters for each output metric using the percentage increase in MSE (%IncMSE). The IPCore parameters are the depth of the data buffer (sample_data_depth), the match unit type to use for all the match units connected to the trigger port (match_type), whether to use relationally placed macros (use_rpms), the edge of the clock to capture and trigger upon (sample_on), optional trigger output port (enable_trigger_output_port), whether to enable optional stor-

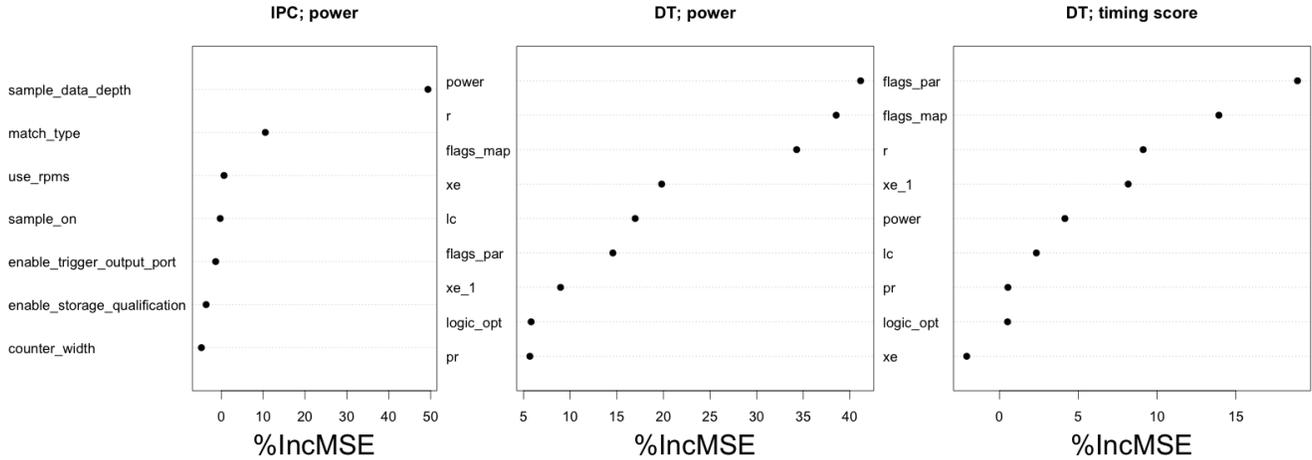


Figure 3. Sensitivity analysis for global models. Power and timing are more sensitive toward changes in the parameters with a higher percentage increase in the mean squared error.

age qualification (`enable_storage_qualification`), and width of the match unit counters (`counter_width`). Among these parameters, the `sample_data_depth` is empirically shown to have significant impact on the power consumption: 50%IncMSE. This is expected because high samples consume more FPGA resources and hence more power. With these models, we can quantify the sensitivity impact of the different ICore options on the estimated power consumption of the design.

The most sensitive `map` tool parameters are power optimization (`power`), register ordering (`r`), combinatorial logic optimization (`logic_opt`), placing registers in I/O (`pr`), extra effort level (`xe`), and LUT-combining of two LUT components into a single LUT with dual output pins (`lc`). The most sensitive `par` tool parameter is extra effort level (`xe_1`). Logic restructuring by passing one of `{none,-ntd,-x}` to `map` (`flags_map`) or `par` (`flags_par`) also shows significant sensitivity. Overall, we can observe that input parameter `r` has a significant impact on both output metrics whereas other parameters have a varying impact on each output. The parameter `r` controls the register ordering; that is, the `map` tool optimizes the grouping of registers into CLBs. From this model, we can deduce that tuning the register ordering has a significant impact on power consumption of the design.

The second stage of modeling is construction of the relationship between the most sensitive inputs and outputs over the whole space. While for search, linear regression was sufficient to detect promising regions, modeling required a more effective machine learning method to take into account the nonlinear interactions among the input parameters. Therefore, we used the random forest algorithm. For validation purposes, Figure 4 shows the correlation plot of actually observed and model-predicted power metrics. The results show that, for both problems, the average prediction error is within 1.0%. Moreover, the prediction accuracy of low power values is better than that of high power values. The reason is that the training points obtained from the

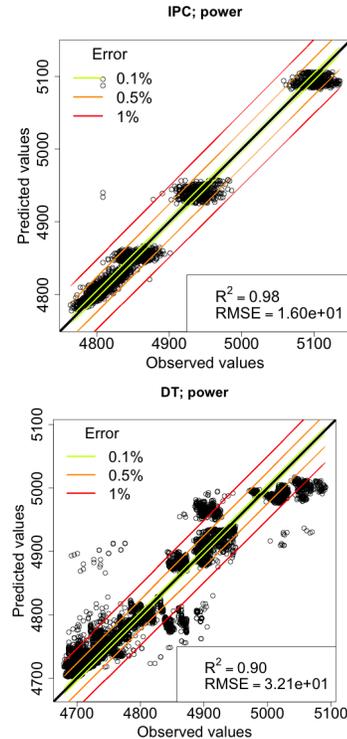


Figure 4. Validation results on IPC and DT datasets for global power models built using the 1,000 evaluations performed in the search.

search algorithm are biased toward low power values.

Overall, global models can be useful to quantify how design parameters affect timing and power metrics. For example, plugging in different values of design parameters into the model enables “what-if” analyses of their impact on output. Such analyses can help a designer in making informed customization decisions in presence of trade-offs among area, speed, and/or power.

In related work, Altera Design Space Explorer and Xilinx SmartXplorer are the closest design parameter exploration tools. Autotuning and machine learning-based search have

been extensively studied in high-performance computing (e.g., [16]). Our approach applies the autotuning techniques used in HPC to the problem of parameter tuning in FPGA design. Design tool parameters have been recently studied in [17]. Our work focuses on Xilinx tools whereas [17] study Altera tools. Exploration of parameters of the design itself has been studied in [18]. This work is similar to our exploration of IPCore parameters, which are specific to a particular IPCore and design. Finally, various C-to-RTL synthesis parameters affecting operation scheduling, resource allocation and control flow synthesis have been studied in [19]. Although our search methodology is similar to these methods, the key differences are in the adopted learning algorithm, which results in interpretable models and can give insights into the impact of the parameters. Among the limitations of presented work is whether model-based search produces similar results for other FPGA designs. This is to be addressed in future work by applying the approach to larger and more sophisticated designs.

VI. CONCLUSION

In this paper we presented a preliminary investigation into an autotuning approach for model-based FPGA design space exploration. Our experiments show that it picks a near-optimal parameter configuration that is within 0.2% of the globally optimal power consumption among designs with satisfactory timing and area constraints. The tuning is based on selective sampling and can find a near-optimal value after about 300 samples each as opposed to the 8,000 design tool or 2,400 IPCore parameter configurations required by exhaustive search. Furthermore, without requiring additional parameter configuration sampling, the models resulting from optimization produce relatively accurate predictions of performance and power across the entire design parameter space. Based on these speedups and accuracy, machine learning-based autotuning shows promise of becoming an important part of FPGA design and optimization workflows.

REFERENCES

- [1] W. Carter, K. Duong, R. H. Freeman, H.-C. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo, and S. L. Sze, "A user programmable reconfigurable logic array," in *Proceedings of the Custom Integrated Circuits Conference*, 1986, pp. 233–235.
 - [2] D. Chen, J. Cong, and P. Pan, "FPGA design automation: A survey," *Foundations and Trends in Electronic Design Automation*, vol. 1, no. 3, pp. 139–169, Oct 2006.
 - [3] Xilinx, "7 Series FPGAs Overview [DS180 (v1.16)]," http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf, Oct 8, 2014.
 - [4] —, "Zynq-7000 All Programmable SoC Overview [DS190 (v1.7)]," http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf, Oct 8, 2014.
- This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.
- The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.
- [5] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: Survey and challenges," *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, Apr 2008.
 - [6] Xilinx, *Command Line Tools User Guide [UG628 (v14.5) March 20, 2013]*, 14th ed., Xilinx Inc., 2100 Logic Dr, San Jose, CA 95124, March 20, 2013.
 - [7] —, *SmartXplorer for ISE Project Navigator Users Tutorial [UG689 (v12.1.1) May 28, 2010]*, 12th ed., Xilinx Inc., 2100 Logic Dr, San Jose, CA 95124, May 2010.
 - [8] Altera, *Quartus II Handbook Volume 1: Design and Synthesis, qii5v1 ed.*, Altera Corporation, 101 Innovation Dr, San Jose, CA 95134, Aug 2014.
 - [9] Xilinx, "Implementing Triple-Rate SDI with Virtex-6 FPGA GTX Transceivers [XAPP1075 (v1.1)]," Nov 2, 2010.
 - [10] —, "Virtex-6 FPGA ML605 Evaluation Kit," http://www.xilinx.com/publications/prod_mktg/ml605_product_brief.pdf, Oct 2012.
 - [11] P. Balaprakash, A. Tiwari, and S. M. Wild, "Multi-objective optimization of HPC kernels for performance, power, and energy," in *Proc. PMBS13*, Nov 2013.
 - [12] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, New York, 2006, vol. 1.
 - [13] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
 - [14] G. Holmes, M. Hall, and E. Prank, "Generating rule sets from model trees," in *Advanced Topics in Artificial Intelligence*, N. Foo, Ed. Springer Berlin Heidelberg, 1999, vol. 1747, pp. 1–12.
 - [15] M. A. Hearst, S. Dumais, E. Osman, J. Platt, and B. Scholkopf, "Support vector machines," *Intelligent Systems and Their Applications, IEEE*, vol. 13, no. 4, pp. 18–28, 1998.
 - [16] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick, "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, ser. SC '08. Piscataway, NJ, USA: IEEE Press, 2008, pp. 4:1–4:12. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1413370.1413375>
 - [17] N. Kapre, H. Ng, K. Teo, and J. Naude, "InTime: A machine learning approach for efficient selection of FPGA CAD tool parameters," in *Proceedings of the 23rd ACM/SIGDA Int'l Symposium on FPGAs*, 2015, pp. 23–26.
 - [18] M. Kurek, T. Becker, T. C. Chau, and W. Luk, "Automating optimization of reconfigurable designs," in *Proceedings of the 22nd IEEE Int'l Symposium on Field-Programmable Custom Computing Machines*, 2014, pp. 210–213.
 - [19] C. Pilato, D. Loiacono, A. Tumeo, F. Ferrandi, P. L. Lanzi, and D. Sciuto, "Speeding-up expensive evaluations in high-level synthesis using solution modeling and fitness inheritance," in *Computational Intelligence in Expensive Optimization Problems. Adaptation, Learning, and Optimization*, Y. Tenne and C.-K. Goh, Eds. Springer, 2010, vol. 2, pp. 701–723.