Efficient and Robust High-Level Synthesis Design Space Exploration through offline Micro-kernels Pre-characterization

Zi Wang, Jianqi Chen and Benjamin Carrion Schafer
The University of Texas at Dallas
Department of Electrical and Computer Engineering
(zi.wang5@utdallas.edu, jianqi.chen@utdallas.edu, schaferb@utdallas.edu)

***Abstract*—Abstract—This work proposes a method to accelerate the process of High-Level Synthesis (HLS) Design Space Exploration (DSE) by pre-characterizing micro-kernels offline and creating predictive models of these. HLS allows to generate different types of micro-architectures from the same untimed behavioral description. This is typically done by setting different combinations of synthesis options in the form or synthesis directives specified as pragmas in the code. This allows, e.g. to control how loops should be synthesized, arrays and functions. Unique combinations of these pragmas leads to micro-architectures with a unique area vs. performance/power trade-offs. The main problem is that the search space grows exponentially with the number of explorable operations. Thus, the main goal of efficient HLS DSE is to find the synthesis directives' combinations that lead to the Pareto-optimal designs quickly. Our proposed method is based on the pre-characterization of micro-kernels offline, creating predictive models for each of the kernels, and using the results to explore a new unseen behavioral description using compositional methods. In addition, we make use of perceptual hashing to match new unseen micro-kernels with the pre-characterized micro-kernels in order to further speed up the search process. Experimental results show that our proposed method is orders of magnitude faster than traditional methods.**

I. Introduction

The need to further reduce the turn-around-time (TAT) in VLSI design, combined with the increase in complexity of current integrated circuits, has convinced many companies to adopt High-Level Synthesis (HLS) in their VLSI design flows. HLS takes as input an untimed behavioral description and generates an efficient RTL description in either Verilog or VHDL. Raising the level of abstraction from the RT-level to the behavioral level has many advantages. One extremely powerful advantage is that it allows the generation of a variety of micro-architectures with unique trade-offs from the same untimed behavioral description by setting different synthesis options [1]. These synthesis options are typical specified at the behavioral description in the form of pragmas and mainly control how to synthesize arrays, loops and functions. E.g. arrays can be synthesized as RAMs, registers or fully expanded. Loops can be fully unrolled, partially unrolled or pipelined and functions can be inlined or not. Out of all the possible micro-architectures, the designer is typically only interested in the Pareto-optimal ones.

HLS design space exploration (DSE) can be classified as a multi-objective optimization problem that has the goal to minimize conflicting design objectives like area, delay and power. The main problem is that the search space grows exponentially with the explorable operations in the code. Because the search space is typically very large, exhaustive enumerations of all combinations are not possible. This has opened the door to different types of heuristics methods to find the trade-off curve of Pareto-optimal designs, but because Pareto-optimality cannot be guaranteed, these designs are often referred to as the dominating designs. In this work, we will use these two terms indistinguishably. Thus, the main goal of HLS DSE is to find synthesis options combinations that lead to the Pareto-optimal designs in a reasonable amount of time. The main approaches so far can be classified in either (i) synthesis based or (ii) model based. In the first case, a cost function drives the generation of new combination of synthesis options, which are in turn evaluated by synthesizing (HLS) the behavioral description with these new options [2]–[6]. In the latter case, a predictive model is generated in order to avoid having to call the synthesis process, which is the most time consuming part of the exploration flow. The predictive model is build after N runs [7]–[9] or even no HLS runs [10], [11] to estimate the area and performance for each explored design. The main problem with this synthesis-less approach is that the proposed models are hard to be ported to another HLS tool, or even to a different version of the same tool, as the model assumes the HLS tool implementation are fixed. For example, older Xilinx SDx versions consistently used registers to implement small partitioned array. This has changed in new versions and if an array partition is required to support two reads in one cycle, the latest Xilinx SDx always synthesizes the array into a dual-port BRAMs. Such implementation details are hard to be captured and maintained in analytical models. Moreover, these HLS DSE methods have been only applied to FPGA HLS DSE because the analytical models are more straight forward and the search space considerable smaller than in the ASIC case.

In this work we propose a hybrid method, that makes use of library of fully pre-characterized micro-kernels for ASIC and FPGAs. Given an unseen new design, the proposed method first analysis the new behavioral description partitioning it into micro-kernels. It then maps these micro-kernels to kernels available in the library. These micro-kernels have different granulates, and include e.g. FIR filters, average calculations, RGB2YUV conversion. The proposed method then continues by pre-characterizing the micro-kernels not found in the library and finishes by exploring the complete behavioral description using a compositional method. In

978-3-9819263-4-7/DATE20/©2020 EDAA

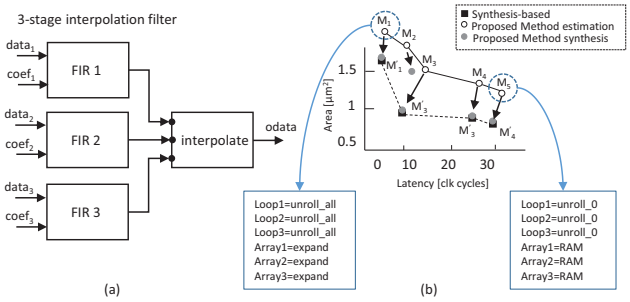

Fig. 1: Three stage interpolation Filter motivational example. (a) Filter structure. (b) DSE exploration results comparison with synthesis directives samples that lead to fastest and smallest designs.

summary, this work makes the following contributions:
(1) Proposes the pre-characterization of behavioral micro-kernels into paramterizable predictive models to maximize their reuse for HLS DSE.
(2) Presents a method to map micro-kernels from a new unseen behavioral description to micro-kernels stored in a library based on perceptual hashing.
(3) Presents a compositional HLS DSE method that mixes previously pre-characterized micro-kernels and newly explored micro-kernels.

II. Motivational Example

HLS has shown to work very well for data-intensive applications like image and DSP processing applications. These applications often rely on basic functions that we call micro-kernels in this work. Fig. 1 shows a motivational example in the form of a three-stage interpolation filter. Fig. 1(a) shows the internal structure of this filter that is composed of three FIR filters connected in parallel. In this case, the FIR filter is the micro-kernel. The outputs of the filters are in turn passed to the interpolation logic to generate the final output.

Fig. 1(b) shows the HLS DSE results when the untimed behavioral description is fully explored using an exhaustive synthesis based method that generates all the possible synthesis directives. This is possible in this case because the description is small. There are a total of 4,096 combinations.

Because the interpolation filter is composed of FIR filters, we propose to pre-characterize these by extracting the FIR filter and performing a HLS DSE on the FIR filter offline. The exploration results are added to a library that in turn expresses their area and latency in terms of the most salient parameters. In this case the filter taps, the bitwidth of the data and coefficients when different set of pragmas are used. The library is in turn used to explore the interpolation filter by using the predictive models for area and latency.

Fig. 1(b) shows the results of our proposed method on two trade-off curves. The trade-off curved formed by micro-architectures $\{M_1, M_2, M_3, M_4, M_5\}$ are obtained by using the area and latencies predicted by our method. As it can be observed the area is over-estimated. Moreover, some designs, in this case M_2 do not lead to a Pareto-optimal micro-architecture.

As we will show later, we claim that the precision of the estimation is not so important, as the main goals is to find the combination of synthesis directives that lead to the optimal design implementations. Thus, after the exploration using estimates, our proposed method synthesizes (HLS) the behavioral description with the synthesis directives that have been predicted to lead to the dominating designs. This approach leads to finding all the Pareto-optimal micro-architectures $\{M_1^{'}, M_3^{'}, M_4^{'}, M_5^{'}\}$ and discarding M_2 as it turns out to be dominated by the other micro-architectures, as shown in Fig. 1(b) . Fig. 1(b) also shows two examples of the synthesis directives configurations that lead to two of the optimal designs. In particular to the fastest design, but also largest, and the smallest, but also slowest one.

One additional source of errors is that different parts of the untimed description, those not being part of any kernel, are fully ignored. In this example, the interpolation part is completely ignored during our proposed exploration method. In the case of having larger micro-kernels, that are not present in the pre-characterized library, our proposed method will extract that kernel and pre-characterize it and the use it for the DSE.

The main advantage of our proposed method is that it could find the optimal solution in 50 seconds (only the exploration phase, excluding the pre-characterization part), while the synthesis based method took over 2 hours. Based on these results we can formulate the problem to be solved as:

Problem Formulation: Given an unseen behavioral description (C_{unseen}) for HLS, which leads to a trade-off curve of Pareto-optimal designs $PO = \{D_1, D_2, \ldots, D_n\}$ when setting a unique set of synthesis directives (SD) in the form of pragmas that control how to synthesize arrays, loops and functions, find the SD combinations that lead to the PO designs quickly.

TABLE I: Overview of proposed HLS DSE techniques.

	Technique and short description
Synthesis based	**Meta-heuristics**
	Simulated annealing [12] SA with adaptive cost function
	Genetic algorithm [13]
	Dedicated Heuristics
	Gradient [3]. Gradient-based pruning technique
	Loops&mem [4]. Analysis the dependencies between loops and arrays
	Loops&mem [14]. Analysis the dependencies between loops and arrays
	Clustering [5]. Sampling and clustering
	S2FA [6]. Partition the design space based on loop hierarchy
	Lattice [15]. Build a lattice structure and traverse it
Model based	**Supervised Learning**
	Random Forest [8]. Smart sample with linear regression
	TED & Random Forest [7]. TED with Random forest
	Regressor based [16]. Adaptive threshold with non-pareto elimination
	Graph Based
	CDFG analysis [10]. Static analysis of CDFGA (loops and arrays)
	CDFGA analysis [11]. Improved static analysis of CDFGA

III. Previous Work

Table I summarizes the most recent related work, classifying the HLS DSE methods into synthesis based and model

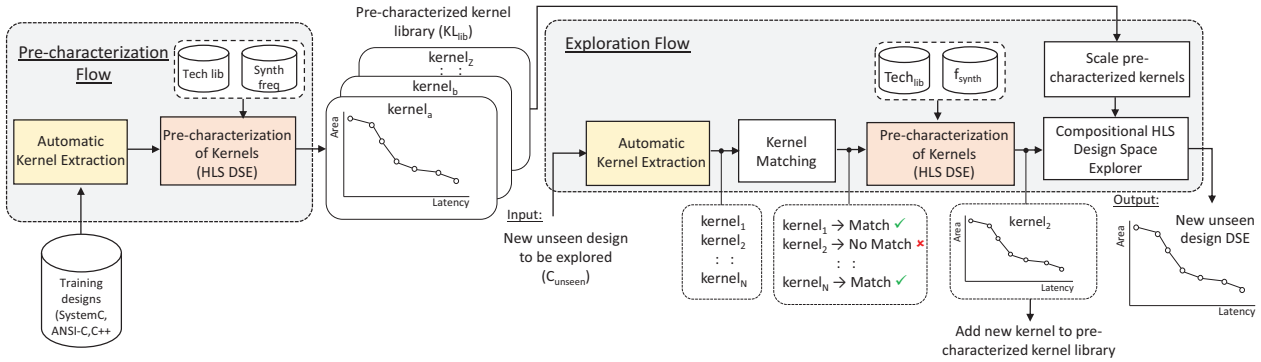

Fig. 2: Overview of complete proposed flow (micro-kernel pre-characterization and HLS DSE).

based approaches. A good survey for HLS DSE methods can be found at [17].

Synthesis based methods generate a new set of of unique synthesis combinations settings and call the HLS tool in order to evaluate the effect of these new settings on the quality metrics. This category can be further sub-divided into meta-heuristics like e.g. simulated annealing [12] or genetic algorithm [13] and dedicated heuristics [3]–[6], [14], [15]. It is this second category as shown in Table I that has attracted the most attention.

A third category (supervised learning methods) is a mixture between synthesis and model based [7], [8], [16]. The most time-consuming part of the DSE process is every time that the HLS tools needs to be invoked. Thus, these techniques use supervised learning to generate predictive models to predict the area and performance given a new set of synthesis settings in order to avoid having to invoke the HLS tool. In order to generate these models they first sample the search space by synthesizing a certain number of designs and continuously build a predictive model until the model is stable (the error between the model and the synthesized designs is smaller than a certain maximum error threshold). These methods then continue the exploration using that predictive model instead of invoking the HLS tool. The last category has been recently introduced and is graph analysis based [10], [11]. This means that no HLS runs are needed at all. These techniques generate and calibrate offline prediction models that are then used in combination with diverse graph analysis techniques. One of the problems with these graph-based methods is that they rely on accurately modeling what the HLS tools does, including its inefficiencies and bugs. Also, they have mainly used when FPGA are targeted because the search space is much smaller and because the predictive models are easier to generate (e.g. arrays are mapped to BRAM and multiplications to DSP macros).

Our proposed approach could be considered a mixture between synthesis and model based and hence, having the best of both worlds. Our proposed method can guarantee to achieve good results quickly by characterizing a set of of micro-kernels that we extract offline, combined with the exploration of only those kernels not found in the library.

The proposed method is based on two distinct parts. Fig. 2 shows an overview of the complete flow. The first part extracts micro-kernels from different training benchmarks and pre-characterizes them by performing an exhaustive DSE on them. Then, a predictive model of the area and latency is generated for each micro-kernel. The resultant predictive model is added to a database of pre-characterized micro-kernels. In addition, each micro-kernel is fingerprinted with a unique hash value. This allows to quickly identify if the same or a similar micro-kernel is later found. A hash function is considered good if it minimizes the probability of hash value collision for two structurally different kernels. Thus, we create a hash value based on the abstract syntax tree (AST) representation of the micro-kernels.

The second part of the exploration flow, makes use of this library to speed up the exploration process. The proposed explorer takes as input a new unseen behavioral description to be explored and extracts all the micro-kernels. Currently these are grouped by loops and independent functions. The next step searches in the micro-kernel library for *similar* kernels, based on their hash values. A full DSE is in turn performed for the micro-kernels not found in the library. The explorer then continues by using a compositional technique to obtain the synthesis directives that lead to Pareto-optimal designs. Finally, the explorer synthesizes the designs with these synthesis directives combinations and reports the dominating designs. The next subsections describe our proposed method in detail.

IV. Proposed Exploration Framework

Part 1: Micro-Kernel Pre-characterization: This first part takes as input a set of training benchmarks and decomposes them into micro-kernels. This is done by parsing the behavioral descriptions and automatically encapsulating individual loops and functions into independent descriptions. These micro-kernels are in turn explored by enumerating all possible synthesis directives of the micro-kernel. Because the size of the kernel is typically very small, an exhaustive search is possible. In addition, because the result of the HLS process is heavily dependent on the target synthesis frequency and target technology, the proposed pre-characterization is repeated for different standard target frequencies and available technologies. In particular $f_{target} = \{50Mhz, 100Mhz, 150Mhz, 200Mhz\}$ and $tech_{lib} = \{ASIC_{45nm}, ASIC_{60nm}, ASIC_{90nm}\}$ and for different FPGA families. Although time-consuming, this process is only done once. The result of this training is

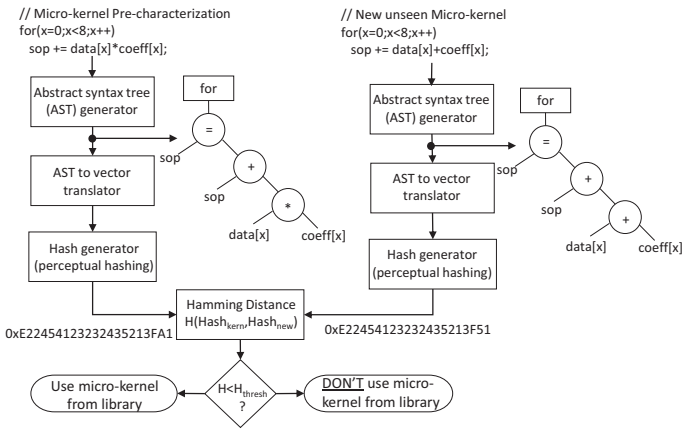

Fig. 3: Overview of Micro-kernel fingerprinting through perceptual hashing of abstract syntax tree (AST).

then passed to a predictive model generator (scikit-learn) in order to get a predictive model for the Area (A) and latency (L) of the kernel. The main predictors found were: (i) the synthesis directives of loops and arrays in the kernel, (ii) target synthesis frequency, (iii) delay of functional units (FUs), (iv) bitwidth of input and output signals and (v) loop iterations of all the loops in the kernel (if any). Depending on the kernel complexity either simple linear regression models are used or more sophisticated random forest models.

Moreover, to make the identification of the kernel easier when analyzing new unseen behavioral descriptions to be explored, each kernel is fingerprinted by creating a unique hash value that uniquely identifies it. This hash value should detect when two micro-kernels are structural different, but also allow to identify if they are similar. When kernels are similar, the effect of the synthesis directives on them will most likely be similar, albeit the area and latency estimates not being accurate. Thus, as we will show in the experimental results section, the template can still be used for DSE.

Fig. 3 shows an overview of the kernel fingerprinting flow as well as how this fingerprint is used for new unseen kernels. To generate the hash value, we build an abstract syntax tree of the micro-kernel and create a vector specifying each single node-type (loops, conditional statement, block statements, variables, etc..) following a depth-first tree traversal method. This vector is in turn passed to the hash generator. The hash generator uses perceptual hashing [18], which is an algorithm commonly used in digital forensics. The key advantage of perceptual hashing is that it is robust enough to take into account dissimilarities, but flexible enough to distinguish between different micro-kernel structures. Traditional cryptographic hashing methods, e.g. MD5 and SHA result in completely different hashes if a single bit changes, thus, making them not useful for this application. Perceptual hashing, in contrast, can be used to compare two micro-kernels by calculating their Hamming distance (H). This perceptual hash value is considered the micro-kernel signature represented in this case by a single 64-bit value. The similarity/difference between two micro-kernels is in turn estimated by computing the hamming distances between heir hash values, e.g. $Hash_i$ and $Hash_j$,$Hash_{diff}(i,j) = \{H(Hash_i, Hash_j)\}$, where a large H represents a large difference in the micro-kernels being compared. In the example given in Fig. 3, only one operation changes between the two kernels (the multiplication is converted into an addition). Thus, the hash values are almost identical, and hence, the result is that the pre-characterized kernel can be used in the exploration processes. The threshold value in terms of acceptable hamming distances used was set to 20% which has shown empirically to lead to good results.

Part 2: HLS Design Space Exploration: This second part does the actual design space exploration for an unseen new behavioral description. The input is a new unseen behavioral description (C_{unseen}), the technology library ($tech_{lib}$), a target synthesis frequency (f_{synth}) and the library of pre-characterized micro-kernels (KL_{lib}). The output will be the set of synthesis directives (SD_list) that lead to Pareto-optimal/dominating micro-architectures (PO_{list}). As shown in Fig. 2, this process is composed of 4 steps, with 2 of the steps being shared with the pre-characterization flow. In particular the automatic kernel extraction and the pre-characterization of individual kernels. Algorithm 1 summarizes these four main steps.

Step 1: Automatic Kernel Extraction: This stage takes as input a newly unseen behavioral description and decomposes it into micro-kernels (line 2). For this it re-uses the same kernel extraction mechanism described in the pre-characterization phase of the micro-kernels. The output of this stage is a list of kernels $KL = \{kernel_1, kernel_2, \ldots, kernel_N\}$.

Step 2: Kernel Matching: This second step takes as input the list of kernels (KL) generated in the previous step and matches them to kernels in the library (KL_{lib}) (lines 4 to 9). Two conditions have to bet met. First that the number and type of explorable operations match and second the hash values are similar. This is done by computing the hamming distance of their hash values as explained previously and shown in Fig. 3. The micro-kernel with lowest hamming distance is then selected that is within a specified maximum threshold value (set to 20% in this work). All the kernels found in the library are grouped together (KL_{match}), and are in turn used for the exploration process (line 8). It should be remembered that the goal is not to accurately estimate the area and performance, but to find the synthesis directives that lead to the optimal designs.

The kernels that do not have any similar pre-characterized kernel are grouped together ($KL_{nomatch}$) and fully characterized in the next step. Thus, after this step we will have two kernel lists $KL = KL_{matched} \cup KL_{nomatch}$.

Step 3: Characterization of Kernels not found in Library: This third steps takes as inputs the micro-kernels list that have no matches in the library ($KL_{nomatch}$) and fully characterizes them (lines 11 to 13). This step is the same as the one in the pre-characterization part of the flow. All possible synthesis attributes are enumerated and the entire search space exhaustively explored. The result of this step is then added to

ALGORITHM 1: Proposed Exploration flow.

input : $\{C_{unseen}, KL_{lib}, Tech_{lib}, f_{synth}\}$
C_{unseen} : Unseen Behavioral Description to be explored
KL_{lib}: Library for pre-characterized micro-kernels
$Tech_{lib}$: Technology library for HLS
f_{synth}: HLS synthesis frequency
output: $\{PO_{list} = \{SD_1, SD_2, \ldots, SD_n\}\}$
PO_{list}: Synthesis directives (SD)that lead to Pareto-optimal configurations of unique Area vs. Latency

1 **/* Step 1: Automatic Kernel Extraction */**
2 $KL = \{kernel_1, \ldots, kernel_N\} = kernel\_extraction(C_{unseen})$;
3 **/* Step 2 : Kernel Matching */**
4 foreach($kernel_i \in KL$){
$kernel\_lib_i = kernel\_match(kernel_i, KL_{lib})$;
5 if($kernel\_lib_i == NULL$)
6 $KL_{nomatch} \leftarrow kernel_i$;
7 else
8 $KL_{match} \leftarrow kernel\_libi$;
9 }
10 **/* Step 3: Characterization of Kernels */**
11 foreach($kernel_i \in KL_{nomatch}$){
$kernel_i = hls\_dse\_kernel(kernel_i)$;
12 $KL_{lib} \leftarrow KL_{match} \leftarrow kernel_i$;
13 }
14 **/* Step 4: HLS DSE of Complete Description */**
15 while($Cost = set\_cost\_func(Area, Latency)$){
foreach($kernel_i \in KL_{match}$){
16 $SD_i = select\_SD\_min\_cost(kernel_i, C)$;
17 $SD_{list} \leftarrow SD_i$;
18 }
19 $QOR(A, L) = estimate\_quality(SD_{list})$;
20 $PO_{candidate} \leftarrow (SD\_list, QOR)$;
21 }
22 $PO_{List} = prune\_non\_optimal(PO_{candidate})$;
23 $return(PO_{List})$;

the pre-characterization library in case that this kernel is found in later designs to be explored (line 12). Encapsulating each kernel individually isolates them from the rest making them independent. This entices that the characterization process can be performed in parallel for all micro-kernels in $KL_{nomatch}$. This is accomplished by making use of multi-threading capabilities of the modern computers. In particular using pthreads, further speeding-up the HLS DSE process. The results of this step is a list of fully characterized micro-kernels that compose the new behavioral description.

Step 4: HLS DSE of complete description: This last step takes as input all the pre-characterized micro-kernels and uses a compositional search technique to find the combination of synthesis directives that lead to the dominating designs (lines 15 to 21). Compositional methods have shown to lead to good results in the context of system-level DSE [19], [20]. In order to fully search the design space a cost function if initially set to minimize either area or latency or to equally balance the area and latency. Based on this the kernels implementations that minimize this cost function are selected and grouped together, with the area and latency estimated additively.

Finally, the dominating configurations are synthesized in order to get accurate area and latency estimates and the synthesis directive combinations that lead to dominating micro-architectures reported (lines 22 and 23).

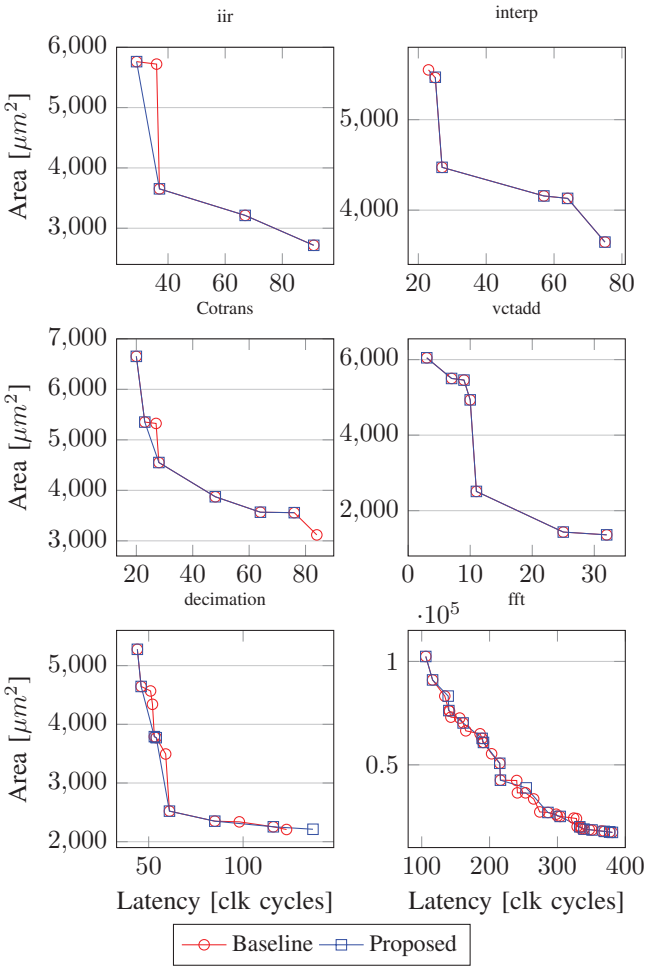

Fig. 4: Experimental results comparison. Synthesis vs. Proposed method trade-offs.

V. Experimental Results

Twenty different applications taken from the open source S2CBench [21] and CHstone [22] benchmark suites were initially used to build the library of micro-kernels. Out of these benchmarks our method extracted and pre-characterized 31 micro-kernels of different complexities. To test the accuracy of our proposed method we chose six benchmarks not used in the pre-characterization process and compare the exploration result against an exhaustive search synthesis based method which guarantees to find the optimal solutions. We call this the baseline method (Baseline). The HLS tool used is CyberWorkBench v.6.1 from NEC. The target synthesis frequency is 100MHz in all cases and the target technology Nangate 45nm.

To measure the quality of the exploration result we use Average Distance to Reference Set (ADRS), which is a widely used metric to compare multi-objective optimization problems like this one. ADRS indicates the average distance between the reference Pareto-front (PO_{ref}) and the approximate Pareto-set, *i.e.*, tells how close a Pareto front (Ω) is to the reference front, where the reference front (Γ) is obtained in this case from the exhaustive search (Baseline). ADRS is

TABLE II: Experimental Results summary

	Proposed Method				Baseline	
Bench	Total kernels	Kernels found	ADRS [%]	Run [s]	ADRS [%]	Run [s]
iir	4	2	0.07	1,710	0	15,179
Interp	4	4	0.14	2,782	0	24,837
Cotrans	3	3	0.91	504	0	11,124
Vctadd	6	3	0	609	0	27,506
decim	4	4	0.79	464	0	15,188
fft	9	5	1.05	3,177	0	65,195
Avg.	5	3.5	0.49		0	
Geom.				1,136		22,027

computed as shown in (1).

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

Fig. 4 shows the HLS DSE results. It can be observed visually that our proposed method leads to very good results. Table II summarizes the results, where columns 2 and 3 indicate the total number of kernels extracted from each benchmark and the number of kernel found in the pre-characterized kernel library. Columns 4 and 5 show the ADRS achieved by our proposed method and the total HLS DSE runtime. Finally, the last two columns show the ADRS and runtime of the exhaustive search baseline method. The ADRS is 0 in all cases as the exhaustive search is able to find the actual Pareto-optimal designs.

From the table we can observe that the ADRS is on average only 0.49% larger, which implies that our proposed method can in most cases either find the synthesis configurations that lead to Pareto-optimal designs or configurations that leads to a design close to all of the Pareto-optimal design (a large ADRS implies that a large exploration region is missing). In terms of running time, our proposed method is on average 19.39 $\times$ faster, where in this case the geometric mean is used to account for the benchmark size differences. It should be noted that the core of our method is extremely fast, taking less than 1 minute, and that the most time consuming part is the synthesis of all the Pareto-optimal configuration candidates, which requires a full synthesis (step 4 in the previously described exploration process).

From the results, it can be concluded that our proposed method works well, finding in most cases the synthesis directive configurations that lead to Pareto-optimal designs, while being much faster.

most of the synthesis directives that lead to Pareto-optimal

VI. Summary and Conclusions

This work presents a method to accelerate the HLS design space exploration by pre-characterizing micro-kernels offline and then using these for new unseen behavioral descriptions. Two key contributions are the use of predictive models to quickly estimate the area and latency of these kernels as well as the use of perceptual hashing to identify similar kernels. Experimental results have shown that using the pre-characterized results of similar kernels still leads to finding micro-architectures. Experimental results show that our proposed method is very effective, while being over 19 $\times$ faster than an exhaustive synthesis-based method.

References

[1] P. Coussy and A. Morawiec, *High-Level Synthesis: From Algorithm to Digital Circuit*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[2] B. C. Schafer and K. Wakabayashi, "Design space exploration acceleration through operation clustering," *IEEE TCAD.*, vol. 29, no. 1, pp. 153–157, Jan. 2010.

[3] S. Xydis, C. Skouroumounis, K. Pekmestzi, D. Soudris, and G. Economakos, "Efficient High Level Synthesis Exploration Methodology Combining Exhaustive and Gradient-Based Pruned Searching," in *ISVLSI*, July 2010, pp. 104–109.

[4] N. K. Pham, A. K. Singh, A. Kumar, and M. M. A. Khin, "Exploiting loop-array dependencies to accelerate the design space exploration with high level synthesis," in *DATE*. New York, NY, USA: ACM, 2014.

[5] L. Ferretti, G. Ansaloni, and L. Pozzi, "Cluster-based heuristic for high level synthesis design space exploration," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–9, 2018.

[6] C. H. Yu *et al.*, "S2fa: An accelerator automation framework for heterogeneous computing in datacenters," in *DAC*. New York, NY, USA: ACM, 2018, pp. 153:1–153:6.

[7] H.-Y. Liu and L. Carloni, "On learning-based methods for design-space exploration with high-level synthesis," in *DAC*, May 2013, pp. 1–7.

[8] M. Zuluaga, A. Krause, P. Milder, and M. Püschel, ""smart" design space sampling to predict pareto-optimal solutions," *SIGPLAN Not.*, vol. 47, no. 5, pp. 119–128, Jun. 2012.

[9] B. C. Schafer and K. Wakabayashi, "Machine learning predictive modelling high-level synthesis design space exploration," *IET Computers Digital Techniques*, vol. 6, no. 3, pp. 153–159, May 2012.

[10] G. Zhong, A. Prakash, Y. Liang, T. Mitra, and S. Niar, "Lin-analyzer: A high-level performance analysis tool for fpga-based accelerators," in *DAC*, June 2016, pp. 1–6.

[11] 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 *ICCAD*, Nov 2017, pp. 430–437.

[12] B. C. Schafer, T. Takenaka, and K. Wakabayashi, "Adaptive simulated annealer for high level synthesis design space exploration," in *VLSI-DAT*, April 2009, pp. 106–109.

[13] B. Carrion Schafer, "Parallel high-level synthesis design space exploration for behavioral ips of exact latencies," *TODAES*, vol. 22, no. 4, pp. 65:1–65:20, May 2017.

[14] A. Cilardo and L. Gallo, "Interplay of loop unrolling and multidimensional memory partitioning in hls," in *DATE*, 2015, pp. 163–168.

[15] G. A. Lorenzo Ferretti and L. Pozzi, "Lattice-traversing design space exploration for high level synthesis," in *ICCD*, Nov 2018, pp. 509–512.

[16] P. Meng, A. Althoff, Q. Gautier, and R. Kastner, "Adaptive threshold non-pareto elimination: Re-thinking machine learning for system level design space exploration on fpgas," in *DATE*, March 2016, pp. 918–923.

[17] B. Carrion Schafer and Z. Wang, "High-level synthesis design space exploration: Past, present and future," *IEEE TCAD*, pp. 1–12, 2019.

[18] J. Fridrich and M. Goljan, "Robust hash functions for digital watermarking," in *Proceedings International Conference on Information Technology*, 2000, pp. 178–183, exported from https://app.dimensions.ai on 2019/05/17.

[19] G. Di Guglielmo, C. Pilato, and L. P. Carloni, "A design methodology for compositional high-level synthesis of communication-centric socs," in *DAC*, 2014, pp. 128:1–128:6.

[20] L. Piccolboni, P. Mantovani, G. D. Guglielmo, and L. P. Carloni, "Cosmos: Coordination of high-level synthesis and memory optimization for hardware accelerators," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 150:1–150:22, Sep. 2017.

[21] S2cbench, "S2CBench: Synthesizable SystemC Benchmark Suite for High-Level Synthesis," https://sourceforge.net/projects/s2cbench/, 2018.

[22] Y. Hara, H. Tomiyama, S. Honda, H. Takada, and K. Ishii, "Chstone: A benchmark program suite for practical c-based high-level synthesis," in *ISCAS*, May 2008, pp. 1192–1195.