Cluster-Based Heuristic for High Level Synthesis Design Space Exploration

Lorenzo Ferretti
Università della Svizzera Italiana
Lugano, Switzerland
Email: lorenzo.ferretti@usi.ch

Giovanni Ansaloni
Università della Svizzera Italiana
Lugano, Switzerland
Email: giovanni.ansaloni@usi.ch

Laura Pozzi
Università della Svizzera Italiana
Lugano, Switzerland
Email: laura.pozzi@usi.ch

Abstract—High Level Synthesis (HLS) frameworks allow to describe hardware designs in a high-level language (C/C++), without burdening developers with the error-prone task of specifying their implementation in detail. The HLS process is usually controlled by user-specified directives, which influence the implementation area and latency. Nonetheless, the correlation between directives and performance is often difficult to foresee and to quantify. Addressing this challenge, we herein propose a heuristic that, by only exploring a subset of possible configurations for an HLS design, is able to retrieve a close approximation of its Pareto Frontier of non-dominated implementations. Our framework identifies regions of interest in the design space, and iteratively searches for new solutions within such regions, or in their combinations. Experimental evidence across multiple benchmarks showcases that our approach to HLS design space exploration reaches better Pareto approximations, and with less required synthesis runs, with respect to State of the Art alternatives.

I. INTRODUCTION

The common practice for the design of digital Integrated Circuits (ICs) relies on Hardware Description Languages (HDLs), such as VHDL or Verilog. HDL-based. In addition, these approaches require the concurrent specification of a circuit functionality and of its implementation. These two issues compound, making the exploration of different design variants, having diverse costs and performances, an error-prone and time-consuming process.

High Level Synthesis (HLS) tackles these issues by raising the abstraction level at which hardware circuits are defined, allowing an automated translation of e.g. C/C++ code to hardware. Moreover, HLS frameworks enable, through the use of synthesis directives, the rapid generation of different versions of the same ICs, without requiring any further source code modification. HLS is particularly useful for the development of Application Specific Integrated Processors (ASIPs), which combine software-programmable processors and dedicated accelerators, and are widely used in the embedded systems domain. By sharing a common high-level language (C/C++) across the hardware/software divide, HLS workflows enable a fast exploration of ASIPs systems having accelerators with different characteristics, as well as a direct path to their implementation.

Different directive settings can lead to designs with widely different area and latency. On the other hand, typically only few of the resulting implementations are Pareto-optimal. HLS tools give little guidance on which directive settings lead to Pareto solutions. This task is far from obvious, since the rich portfolio of possible synthesis optimizations causes an exponential growth of feasible configurations for a given design. Moreover, directives tend to be inter-dependent according to complex patterns, and their effectiveness is highly influenced by low-level application characteristics such as data dependencies. In this light, an approach based on trial-and-error is clearly sub-optimal, while an exhaustive search is unfeasible beyond very simple cases, because of the explosion in the number of possible configurations.

Herein, we present a novel heuristic, able to well approximate the area/latency Pareto frontier of hardware designs developed with HLS, while synthesizing only a small subset of the possible directives configurations. Our framework combines a) a smart choice of the initial exploration points

Fig. 1: Example of an exhaustive exploration compared with the cluster-based heuristic. The × are the result of exhaustive exploration, while the dots (dark and light) are the design points explored by our heuristic (13% of the total). Dark dots represent the retrieved Pareto solution.
and b) an iterative cluster-based refinement of the estimated Pareto front. Our approach is agnostic both to the set of directives offered by an employed HLS tool, and to the underlying system characteristics. Different HLS frameworks can therefore be interfaced with no required adaptations.

Our contribution is two-fold:
- We propose the use of probabilistic distribution to favour the selection of extreme values for each directive. The proposed distribution allows to start the design space exploration closer to the Pareto frontier.
- We introduce a cluster-based heuristic, able to effectively divide the design space into regions, and focus the exploration only on promising ones. The heuristic is composed of two stages: an intra-cluster exploration stage which aims to refine the search within a region, followed by an inter-cluster exploration stage which allows to discover new, unexplored areas of the design space.

Our strategy outperforms state-of-the-art methodologies in terms of the number of required synthesis, the closeness of the retrieved (estimated) Pareto frontier with respect to the true one, and the algorithm run-time. In the example presented in Figure 1, referring to the Autocorr function from the gsm CHStone benchmark [1], an Average Distance from Reference Set (ADRS) of 0.0128 was achieved with only 234 synthesis runs (over 1728 possible configuration), and an average algorithm run-time of ~201 seconds.

The paper continues as follows: in Section II we compare our contribution with respect to related efforts in the field. Section III provides a high-level description of the main features characterizing our strategy. Section IV introduces the mathematical foundations and the notation used in this paper and Section V details the various steps composing our cluster-based heuristic. Section VI and VII provide the experimental evidence highlighting the performance of our approach. Concluding remarks are exposed in Section VIII.

The source code of the framework implementation, as well as a database of the area-latency values derived from the exhaustive explorations for the considered benchmarks, are available at http://www.inf.usi.ch/phd/ferretti/cluster-based-DSE.html [2].

II. STATE OF THE ART

A number of academic and industrial High-level Synthesis frameworks exist, including LegUp [3], ROCCC [4], SPARK [5] and Xilinx VivadoHLS [6]. All of them allow users to guide the synthesis process through specific synthesis directives.

Proposed strategies to automate the ensuing design space exploration can be broadly divided in two categories. A first approach, investigated in [7], [8], [9], [10] and [11], is to derive a high-level model of a target application, and estimate the effect of changing the values of its synthesis directives. Since they strive to approximate the behaviour of HLS tools, these works require a small number of synthesis runs (which are only used to fine-tune the results), but are also tightly linked to specific HLS frameworks and to their employed architectural assumptions. Moreover, they usually take into account only a limited set of directives, as the modelling effort steeply increases with each added degree of freedom.

An alternative strategy, which we also adopt, is to treat HLS suites as black boxes, only observing their inputs (directives values) and outputs (area, latency of a design). Machine learning techniques can then be used to infer the behaviour of a synthesis tool, and generate a model for the effect of directives. To this end, the authors of [12] introduced a framework based on spectral analysis, while [13] proposes an algorithm based on Simulated Annealing and [14], [15] propose algorithms based on Response Surface Modelling. Shafer et al. [11] propose instead to use learning-based methods to implement local search techniques. The paper of Liu et al. of [16] investigates a methodology based on Random Forest [17] and its results favourably compare with other black-box alternatives mentioned above. Therefore, we adopt this methodology as a baseline for our experimental evaluation in Section VII.

III. OVERVIEW

Our exploration strategy is motivated by the observation that some combinations of values, when assigned to directives, result in high-quality implementations, while others are sub-optimal, leading to designs with high cost and low performance. Starting from a small initial set of area/latency points in a design space, we therefore explore it by clustering solutions with a high degree of similarity, and discarding clusters which are distant from the Pareto curve.

Clustering allows to decompose the Design Space Exploration (DSE) problem in many smaller sub-problems, effectively lowering its complexity. Solutions are clustered considering their similarity both according to the synthesis output and in the values of the input directives, as shown in Figure 2. Two points (e.g. $\phi_3$ and $\phi_4$ in Figure 2) may be assigned to the same cluster if they present similar design parameters, even if they have quite different area and delay. The grouping of points into clusters is performed each time a new solution is synthesized, letting the correlation between values of different

\[ \phi_3 = [200, 960, 4, 0, 16, 1] \]
\[ \phi_4 = [120, 1500, 4, 0, 4, 1] \]
Initial sampling
A C
B
Cluster selection
Intra-cluster expl.
B
A
D
Inter-cluster expl.

Fig. 3: Overview of the Clustering-Based DSE framework.

directives to naturally emerge. Therefore, our strategy waives the need for a model of the effect of each directive on a target design.

An approach only relying on intra-cluster exploration may never reach DSE regions which do not include any point in the initial set. To avoid this pitfall, clusters are combined to generate new ones. This inter-cluster step enables the exploration of points whose characteristics are in-between two previously considered design space regions. Since it searches for intermediate solutions, such strategy is most effective when points with extreme features values (high or low) are included in the DSE as part of the initial set. To ensure this condition, the features in the initial sampling set are generated according to a U-shaped probabilistic distribution (detailed in Section V-A).

Figure 3 summarises the clustering-based heuristic presented in this work. Starting from the initial sampling of the design space, the Clustering, Cluster selection, Intra-cluster exploration (expansion of each cluster) and Inter-cluster exploration steps (generation of new clusters) are iteratively performed until either no new solution is found, or a user-defined budget of synthesis runs expires.

A detailed description of the DSE framework steps is provided in the following sections.

IV. PROBLEM FORMULATION AND NOTATION

We treat the DSE of possible implementations of an HLS design as a multi-objective optimisation, with area (\(A\)) and latency (\(L\)) as objective functions. In fact, the DSE process can be expressed as:

\[
HLS : \text{Code}, D \rightarrow S.
\]

The synthesis tool, in fact, receives in input the code describing a functionality to be implemented in hardware, and a set of HLS directives values \(\vec{f} \in D\), generating as output a pair of real numbers representing area and latency: \((a, l) \in S \subseteq \mathbb{R} \times \mathbb{R}\). \(S\) and \(D\) are termed the Synthesis space and the Design space, respectively.

We aim at finding an approximate Pareto frontier \(\vec{P}\) of the best performing implementations, as close as possible to the one deriving from exhaustive search (\(P\)), while minimizing the number of synthesis runs. The Pareto frontier is defined as the set of non-dominated solutions in \(S\):

\[
p \in P \iff \nexists q \in S, q \neq p \mid A(q) \leq A(p) \land L(q) \leq L(p).
\]

In other words, if \(p \in P\), then no other solutions in \(S\) have simultaneously less area and less latency than \(p\). The initial sampling step defines the starting set of points belonging to \(\vec{P}\), which is then iteratively refined, resulting in the final output of our framework.

The Design space \(D\) contains the set of all legal combinations of directives, and it is composed of the union of all feature vectors \(\vec{f}\). We name the set of admissible values for a directive as its feature set \(F\). A feature vector is then an element of the cartesian product among all feature sets:

\[
\vec{f} \in F_1 \times F_2 \times \ldots \times F_n
\]

\[
\vec{f} = [f_1, f_2, \ldots, f_n]
\]

To obtain an equally distributed representation of the features in \(F\), we represent these in a discretised form \(\vec{f}^*\), in which each element \(f_i\) has the same distance from its previous and following elements in \(F\) with \(F \in [0, 1]\). For the synthesis space, instead, a normalized representation is adopted and can be straightforwardly derived by dividing each element in \(S\) by the maximum area and latency values among the explored configurations.

Example. Figure 4 shows a simple HLS code snippet and three possible HLS directives that can be applied: function inlining, input bundling and loop unrolling. Inlining can either be performed or not, so the inline directive has two legal values. Similarly, the two arrays can either be bundled on the same port or each assigned to a different one. Finally, for this example we assume that the legal loop unrolling factors are \([1, 2, 5, 10]\), corresponding to the discretized values of \([0, 1/2, 2/2, 1]\). A possible discretized feature vector for this application is \(\vec{f}^* = [0, 1/2, 1]\), which corresponds to an implementation with no port bundling, inlining of function \(\text{bar}\), and the for loop unrolled by 2.

Each data point in the DSE problem is completely characterized by its values in \(S\) and \(D\). We call the concatenation of these two spaces the Clustering space \(\Phi\), and its elements \(\vec{\phi}\):

\[
\vec{\phi} \in \Phi = A \times L \times F_1 \times F_2 \times \ldots \times F_n
\]

\[
\vec{\phi} = [a, l, f_1, f_2, \ldots, f_n]
\]

During the exploration, only a part of \(\Phi\) is visited. We

foo(int in[10], int out[10]){
    bar(in);
    for(i = 0; i < 10; ++i){
        out[i] = in[i] * in[i];
    }
}
f =
Bundling
Function inlining
Loop unrolling

Fig. 4: Example of a C code and the regions that may be addressed with HLS directives.
name $\tilde{\Phi}$ the space containing only the solutions evaluated by our heuristic. Similarly we refer to $S$ and $D$ as to the entire Synthesis and Design spaces and with $\tilde{S}$ and $\tilde{D}$ to their subsets for which, during the algorithm execution, a synthesis has been performed.

The example in Figure 2 shows four different synthesized solutions $\tilde{\phi}_1, \tilde{\phi}_2, \tilde{\phi}_3, \tilde{\phi}_4$, plotted in the Synthesis space $\tilde{S}$. It also reports their value in the Clustering space $\Phi$ as vectors containing the area, latency (first two components) and feature values. The Pareto front for this example $\bar{P}$ contains solutions $\tilde{\phi}_2, \tilde{\phi}_3$ and $\tilde{\phi}_4$. However, $\tilde{\phi}_1$ is not part of $\bar{P}$, since $\tilde{\phi}_2$ has both smaller area and lower latency.

V. EXPLORATION METHODOLOGY

As summarized in Figure 3, the clustering-based exploration process is divided in five steps: a) Initial sampling, b) Clustering, c) Cluster selection, d) Intra-cluster exploration, e) Inter-cluster exploration. After initial sampling, steps (b) to (e) are repeated until no new solution is discovered, or a user-defined budget of synthesis runs is reached.

A. Initial sampling

This step generates the initial set of configurations $\tilde{D}$, and derives the first approximation of the design space Pareto curve $\bar{P}$. For an initial sampling size $n$, the $\tilde{D}$ space is composed of $n$ unique feature vectors $\tilde{f}$. The elements of each $\tilde{f}$ originate from the probabilistic sampling of a symmetric Beta distribution, which is characterized by a density function, defined over an interval $0 \leq x \leq 1$, as:

$$f(x) = \frac{x^{\alpha-1}(1-x)^{\alpha-1}}{B(\alpha)}$$

Where $B$ is the Beta function:

$$B(\alpha) = \int_0^1 x^{\alpha-1}(1-x)^{\alpha-1}dx$$

As shown in Figure 5, this U-shaped distribution has, with a value of $\alpha$ lower than 1, a higher probability associated to the boundary values of $x$. By adopting it, the initial explored set $\tilde{F}$ will contain elements whose features have, with high probability, extreme values from the respective set. This is a desired property in our framework, so that Pareto solutions with in-between feature values will be explored during the refinement steps.

To derive the set of initial exploration points $\tilde{F}$, which are synthesized by the HLS tool, the probabilistic samples are converted to feature values in the $\tilde{F}$ domain, approximating them to the nearest legal configuration. The retrieved area and latency numbers generate the first instance of $\tilde{S}$. Finally, $\tilde{S}$ and $\tilde{F}$ are concatenated to generate $\tilde{\Phi}$.

B. Clustering

Once the initial set of $\tilde{\Phi}$ is generated, its elements ($\phi$) having common characteristics are aggregated into clusters. To this end, we relied on the Hierarchical Clustering algorithm [18][19]. As a similarity metric, we considered the squared euclidean distance among the points in the $\tilde{\Phi}$ space.

To ensure a good balance between intra- and inter-cluster exploration, multiple clusters should be present, while most of them should aggregate multiple points. We governed this trade-off via a clustering factor, which sets the number of clusters to a percentage of the number of the explored designed points. An exploration of different settings for this parameter is presented in Section VII.

The clustering operation partitions the $\tilde{\Phi}$ space in multiple clusters $\tilde{C}_i$, and defines $\tilde{C}$ as:

$$\tilde{C} = \bigcup \tilde{C}_i$$

Each cluster is characterised by its centroid $\tilde{c}$, which is, for each $(a, l, f_1, f_2, ...)$ component, the average value among the points belonging to the clusters. Moreover, clusters possess a boundary in the $S$ space, corresponding to a 4 elements tuple containing the maximum and minimum values of area and latency of the cluster points.

Example: a cluster $C_1 = \{\phi_1, \phi_2, \phi_3\}$ with $\phi_1 = [0.1, 0.8, 0.1, 0.9, 1]$, $\phi_2 = [0.15, 0.7, 0.2, 0.8, 0]$ and $\phi_3 = [0.5, 0.5, 0.4, 0.4, 1]$ has the boundary: $b_{c_1} = (0.1, 0.5, 0.5, 0.8)$.

C. Clusters selection

This step selects which clusters will be considered for the generation of new points in the Synthesis space.

To perform it, as shown in Figure 6a, we consider: the Pareto frontier of the explored design space $\bar{P}$ (pictured as diamonds), the Pareto frontier of the centroids of the clusters $\bar{P}_C$ (pictured as a line among centroids) and the cluster boundaries (pictured as dashed rectangles). This data is employed to select candidate clusters corresponding to design space regions which contain promising solutions. Only the points belonging to these clusters are considered in the following intra- and inter-cluster exploration steps, while the rest are discarded.

Candidate clusters are selected according to three criteria:

1) if $\tilde{s} \in C_i \land \tilde{s} \notin \bar{P}$, then $C_i$ is a candidate cluster.
2) if a cluster $C_i$ belongs to the Pareto frontier of centroids $\bar{P}_C$, then $C_i$ is a candidate cluster.
3) for each $\tilde{c} \in \tilde{C}\backslash\bar{P}_C$, if $A(\tilde{c})$ and $L(\tilde{c})$ are inside the boundaries of an element of $\bar{P}_C$, then $C_i$ is a candidate cluster.

Fig. 5: Beta distributions for different values of $\alpha$, with $\alpha \leq 1$. 
Example: Figure 6 exemplifies the application of the cluster selection criteria. In Figure 6b, A, B and C are all selected because each of them contains elements which belong to $P$. Figure 6c shows an example where the application of the second criterion leads to the selection of clusters A and B. Finally, Figure 6d shows an example where the third criterion is applied, since the centroid of cluster C is within the boundary of cluster B.

D. Intra-cluster exploration

Intra-cluster exploration identifies new solutions by examining candidate clusters individually. For each cluster, the algorithm considers the points belonging to its local Pareto frontier $P_C$. These points are pair-wise combined in the $\Phi$ space by performing a vector addition relative to the cluster centroid, generating new estimated solutions $\hat{\phi}$ (Figure 7). Combinations which do not improve the global Pareto frontier are discarded without performing a synthesis run.

Estimated $\hat{\phi}$ elements may have features values which do not correspond to valid settings for the HLS directives (e.g.: they are not integer numbers). Up to three valid configurations are therefore derived according to the following rules:

1) each component is casted to the closest feature value.
2) all components are upcasted to a valid feature value.
3) all components are downcasted to a valid value.

After this pass, estimated points which have already been synthesized are also discarded. The resulting set of $f$ feature vectors are then used to invoke the HLS synthesis tool to retrieve the corresponding (non-estimated) area and latency. Finally, the new obtained $\phi$ are added to the $\Phi$ space and an updated Pareto frontier is retrieved.

Example: Figure 7 shows an example of intra-cluster exploration. The dark filled dots are the points of the cluster before the exploration, while the dashed empty dots are the estimated Pareto configurations. In this example only the combination of $\phi_1 + \phi_2$ and $\phi_2 + \phi_3$ generate new estimated design points, since $\phi_1 + \phi_3$ does not improve the current Pareto curve.

E. Inter-cluster exploration

In order to discover unexplored regions of the design space, this stage combines design points belonging to different clusters. This step considers all pairings of the clusters in $P_C$, merging them and calculating their common centroid.

Vector addition, relative to the centroid, is applied to the Pareto points of the merged cluster. The resulting set of estimated $\hat{\phi}$ vectors are casted to valid configuration as in the intra-clustering stage. Finally, configurations which are not yet part of $\hat{\Phi}$ are synthesized, and the Pareto frontier is updated.

Example: Figure 8 shows an example of an inter-cluster exploration. The Pareto points of cluster A (dark filled dots) and B (light filled dots) are combined generating new candidate solutions (dashed empty dots).

VI. Experimental setup

The HLS exploration framework presented in the previous sections has been implemented in Matlab, which, as part of its Statistics and Machine Learning Toolbox [19], provides an implementation of the Hierarchical-Clustering algorithm.
TABLE I: Pragmas applied to the different CHStone function explored. (|D| = size of the design space)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Function</th>
<th>Pragma</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>jpeg</td>
<td>ChenIDct</td>
<td>unroll loop1</td>
<td>0.2,4.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop2</td>
<td>0.2,4.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop3</td>
<td>0.2,4,8,16,32,64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bundle</td>
<td>0.1</td>
</tr>
<tr>
<td>Encode</td>
<td></td>
<td>unroll loop1</td>
<td>0.5,10,25,50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop2</td>
<td>0.2,5,10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop3</td>
<td>0.2,4,10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop4</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inline all</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bundle</td>
<td>0.1</td>
</tr>
<tr>
<td>Decode</td>
<td></td>
<td>unroll loop1</td>
<td>0.5,10,25,50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop2</td>
<td>0.2,5,10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop3</td>
<td>0.2,4,10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop4</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inline all</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bundle</td>
<td>0.1</td>
</tr>
<tr>
<td>Autocorr</td>
<td></td>
<td>unroll loop1</td>
<td>0.4,16,40,80,160</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop2</td>
<td>0.4,16,40,80,160</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop3</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop4</td>
<td>0.4,19,38,76,152</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop5</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bundle</td>
<td>0.1</td>
</tr>
<tr>
<td>Reflection</td>
<td></td>
<td>unroll loop1</td>
<td>0.5,9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop2</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop3</td>
<td>0.3,9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unroll loop4</td>
<td>0.2,4,8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inline func</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inline func2</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inline func3</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inline func4</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inline func5</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bundle</td>
<td>0.1</td>
</tr>
</tbody>
</table>

A low value of ADRS indicates that $\hat{P}$ well approximates $P$, while a high one reports a low-quality approximation.

For all experiments, $P$ is derived from an exhaustive search of all possible directive configurations. Such brute-force exploration required multiple days of computation on each of the considered benchmarks, highlighting the importance of exploration heuristics targeting HLS. Since both our methodology and that of Liu et al. have a probabilistic component governed by a seed value, for each experimental setting we ran the algorithms 100 times, averaging the results.

VII. EXPERIMENTAL RESULTS

**Tuning.** We have evaluated the effect of varying the parameters required by our framework: the clustering factor (governing the size and number of clusters generated at runtime) and the number of points evaluated in the initial sampling phase.

Figure 9 shows the impact of adopting different clustering factors, for the considered benchmarks, on the achieved ADRS. The data reported in the figure considers a number of clusters equal to 5%, 10%, 15% and 20% of the number of explored design points. It highlights that this parameter has a small impact on the quality of the results, with a value of 15% leading, on average, to marginally better results.

The size of the initial sampled set plays instead a more important role, and this is seen in Figure 10. The figure shows the performance of our heuristic, in terms of mean ADRS achieved, for different initial sampling sizes (5%, 10%, 15% and 20% of the design space) for the Decode function from the gsm benchmark. It can be observed that, the higher the initial sampling size, the better approximation the algorithm finally converges to — i.e., it converges to a lower ADRS value. On the other hand, if we consider an a-priori limited number of synthesis, then lower initial sampling sizes can outperform higher values. For example, for a budget of 100 synthesis, an initial sampling of 10% reaches a lower ADRS than an initial sampling of 15%. Indeed, if the number of synthesis is limited, there is a tradeoff between how many synthesis should be spent initially, and how many are then left for exploration. We show results for an initial sampling size of 10% and a clustering factor of 15%, but we further experimented with all initial sampling sizes and clustering factors, obtaining consistent results.

**ADRS comparisons.** We now perform a comparative evaluation of our methodology with respect to the one proposed by Liu et al. of [16]. The comparison is illustrated in Figures
11 through 15. These figures report the ADRS achieved by our framework (Clust-Beta), with a maximum budget of synthesis equal to 40% of the total design space and an initial sampling budget equal to 10%. Results are compared with five other combinations of initial sampling and refinement exploration strategies: the intra- and inter-cluster exploration combined with random or TED initial sampling, and Random Forest (RF) exploration of a Beta, random or TED initial sampling. Across all benchmarks, Clust-Beta consistently outperforms alternative methodologies, both when a low or a high number of synthesis are considered, with the only exception of Autocorr for a high synthesis budget. We believe that the competitive advantage of Clust-Beta qualitatively lays in our design space decomposition, together with the intra- and inter-cluster exploration and the use of a Beta-distribution for the initial sampling. The combination of these factors enables to focus the exploration only on the most promising regions of the design space.

We further experimented with all initial sampling sizes reported in Figure 10, sweeping the synthesis budget up until both our method and the best performing alternative converge or reach 40% of the design space size. Results are consistent with the one shown in Figures 11—15: Clust-Beta outperforms the other considered methodologies most (87%) of the times, resulting in smaller ADRS.

Run-time comparisons. Lastly, we have evaluated the algorithm run-time (without considering the time required for the synthesis) of Clust-Beta with respect to RF-TED. For the run-time comparison we have considered the execution time of our methodology and RF-TED running both algorithm until no new synthesizable configuration are generated, or 40% of the design space is explored. The run-time is then divided, in both cases, by the number of synthesis effectively run, so that we could divide the time spent in the exploration engine itself. Results are reported in Table II. Note that, besides reaching a better approximation of the Pareto curves, our methodology is also quicker compared to the one proposed by Liu et al. of [16]. The highest speedup is achieved in the Reflection case, which is the largest among the considered applications, hinting at a better scalability of our methodology.

VIII. CONCLUSIONS

The High Level Synthesis exploration methodology illustrated in this paper allows to discover high-quality implementations of a hardware functionality, while only requiring the synthesis of a small percentage of the design space. The paper addresses a major challenge posed by HLS tools, which usually offer a rich collection of configuration options, but provide little guidance on how these options in turn affect the area and latency of a resulting design.

Our methodology can effectively discover favourable combinations of synthesis directives, by clustering solutions which have a high degree of similarity. By then combining only the most effective directives characterising each cluster, the methodology is able to focus the search towards global Pareto-optimal solution.

The proposed methodology is agnostic to the low-level characteristics of the explored design and to the employed HLS tool. In fact, it does not require an analysis of the application data/control dependencies (as in [11], [7]), nor a model of the considered directives set (as in [10]). It advances the state of the art in black-box HLS exploration frameworks [16] by providing a better Pareto approximation of best-performing implementation of digital designs, while requiring less synthesis runs.

TABLE II: Run-time comparisons: Clust-Beta vs RF-TED

<table>
<thead>
<tr>
<th>Method</th>
<th>Run-time comparisons</th>
<th>ChenIIRF</th>
<th>Encode</th>
<th>Decode</th>
<th>Autocorr</th>
<th>Reflection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clust-Beta</td>
<td>1.76x</td>
<td>1.9x</td>
<td>1.81x</td>
<td>1.31x</td>
<td>20.17x</td>
<td></td>
</tr>
</tbody>
</table>

REFERENCES

Fig. 11: Methodologies comparison, ChenIDCt benchmark.

Fig. 12: Methodologies comparison, Encode benchmark.

Fig. 13: Methodologies comparison, Decode benchmark.

Fig. 14: Methodologies comparison, Autocorr benchmark.

Fig. 15: Methodologies comparison, Reflection benchmark.


