SPIRIT: Spectral-Aware Pareto Iterative Refinement Optimization for Supervised High Level Synthesis

Sotiros Xydis Gianluca Palermo Vittorio Zaccaria Cristina Silvano

Abstract—A new class of design space exploration problems focusing on supervised High Level Synthesis (HLS) environments has been recently identified in literature. In this context, exploration strategies play the role of supervisor for tuning an HLS engine to generate design solutions that form optimal trade-off points for a set of target objectives. Additionally, the complexity of the problem is increased due to the large set of tunable parameters exposed by the "new wave" of HLS tools that include not only architectural alternatives but also on compiler transformations. In this paper, we developed a novel HLS exploration framework, called SPIRIT, that exploits response surface models (RSMs) for predicting the quality of the design points without resorting to costly architectural synthesis procedures. We show that the target solution space can be accurately modeled through RSMs, thus enabling a speedup of the overall exploration without compromising the quality of results. Furthermore, we introduce the usage of spectral techniques to guide iterative refinement. At each iteration, spectral analysis is used to characterize the randomness of different regions of the design space. Given a maximum number of samples per iteration, the proposed exploration strategy selectively refines the RSM with points found on the predicted Pareto set, thus moving towards higher quality solutions, and with points found in high variance regions of the design space, thus improving prediction accuracy. Comparative experimental results demonstrate the effectiveness of the proposed approach in terms of quality improvements of the design solutions and exploration runtime reductions. A case study has been used to describe how the proposed approach can be exploited for customizing the architecture when more than one application kernel should be hardware accelerated.

I. INTRODUCTION

The increasing market demand of consumer electronics imposes strict time-to-market constraints to system designers. High Level Synthesis (HLS) has been recognized as a key enabler for automated coprocessor synthesis within shortened design cycles. Being now in a phase of rapidly increasing its maturity [1], HLS is the driving force of design abstraction, offering an automated path from high level algorithmic specification down to circuit level implementation. Since there is a large number of explorable HLS parameters, designers have faced the problem of reasoning on different trade-offs regarding the generated architectural configurations. Efficient design space exploration (DSE) methodologies that supervise the invocation of HLS tools are of great importance for fast evaluation and pruning of the solution space [2].

The efficiency of the supervised HLS exploration is affected by two factors: (i) the core HLS optimization phases, namely operation scheduling, resource binding and register allocation and (ii) the RTL to gate-level synthesis tool. Iterative coprocessor customization implies the execution of two synthesis phases, namely (i) the C-to-RTL and (ii) the RTL-to-gates. Although the incorporation of gate-level synthesis during HLS exploration enables more accurate timing/area evaluation [3], it represents an extremely time consuming task. To reduce the exploration time, designers usually adopt a less time consuming exploration approach by shortening the exploration loop through the HLS design space only. Nowadays, it is common practice [4], [5], [6], [7], [8], [9] to adopt an iterative exploration approach at the level of HLS by evaluating the quality of the design solutions in a pre-synthesis step. In this way, only a reduced set of design configurations forming the Pareto-front [10] of the high level exploration is propagated down to gate-level and physical synthesis tools for further refinement. For example, let us consider the execution time required for a single run of the SPARK’s HLS engine [11] for the Inverse Discrete Cosine Transform kernel from the MPEG-2 application. When unrolling the inner and outer loops, it requires up to 8.5 minutes, while for a single gate-level synthesis evaluation, by using Synopsys Design Compiler [12], the time required is 23 minutes. The aggregated delay of iteratively using even only the core HLS engine imposes long and in many cases unaffordable exploration runtime. A straightforward approach would suggest to limit the design space, e.g. excluding from the exploration procedure the compiler level parameters, (i.e. the loop unrolling parameters). However, many researchers [13], [14], [9] have proven that this type of exploration simplifications inherently reduce the effectiveness of the DSE and thus the optimality of the approximated Pareto curve.

In this paper, we address the problem of supervised HLS for application-specific coprocessor synthesis by proposing for the first time the combination of spectral techniques with Response Surface Methods (RSMs) to efficiently approximate Pareto optimal design solutions. We show (i) how the multi-objective HLS solution space can be modelled as time-series, thus being amenable to signal processing techniques, (ii) how spectral analysis can be used for extracting valuable information of the structure of the design space and (iii) how the typical structure of Pareto Iterative Refinement exploration strategies should be extended to efficiently integrate and manage spectral analysis. Our main objective is not the complete replacement of the synthesis engine, but to introduce an abstraction meta-layer that enables the reduction of the number
of times that the proposed exploration framework has to resort to the costly architectural synthesis step. For efficiently exploring the HLS design space without compromising the quality of design solutions, we construct and validate delay and area predictive RSM models to analytically capture the behavior of the core synthesis optimization phases (namely operation scheduling, resource binding and register allocation). Validation results show that the HLS solution space can be accurately modeled by using RSM techniques. Furthermore, we introduce spectral analysis techniques to characterize the randomness of different regions of the solution space, and then we exploit the extracted knowledge to guide the iterative RSM refinement. Efficiently incorporating spectral analysis within design space exploration is a hard task, and several techniques have been developed: identification of hard-to-predict partitions in the solution space, adaptive sampling strategy based on both the Pareto solutions and the spectrum distribution over the solution space, extending the structure of existing meta-model assisted exploration frameworks. While existing exploration approaches adopt refinement strategies focused on decision metrics characterizing the optimization potential of the configurations to be evaluated in next exploration rounds [15–19], we propose to extent this concept by incorporating in the refinement process design configurations found in high variance regions of the design space. Considering configurations from both the predicted Pareto and the regions exhibiting high degree of randomness, we manage to refine the employed RSM towards higher quality solutions, therefore improving its prediction accuracy.

An additional contribution of this paper consists of the first comparative analysis among all the state-of-art techniques adopted so far in the field of supervised- HLS exploration. In Section V, we demonstrate how the proposed exploration framework has to resort to the costly architectural synthesis step. For efficiently exploring the HLS design space without compromising the quality of design solutions, we construct and validate delay and area predictive RSM models to analytically capture the behavior of the core synthesis optimization phases (namely operation scheduling, resource binding and register allocation). Validation results show that the HLS solution space can be accurately modeled by using RSM techniques. Furthermore, we introduce spectral analysis techniques to characterize the randomness of different regions of the solution space, and then we exploit the extracted knowledge to guide the iterative RSM refinement. Efficiently incorporating spectral analysis within design space exploration is a hard task, and several techniques have been developed: identification of hard-to-predict partitions in the solution space, adaptive sampling strategy based on both the Pareto solutions and the spectrum distribution over the solution space, extending the structure of existing meta-model assisted exploration frameworks. While existing exploration approaches adopt refinement strategies focused on decision metrics characterizing the optimization potential of the configurations to be evaluated in next exploration rounds [15–19], we propose to extent this concept by incorporating in the refinement process design configurations found in high variance regions of the design space. Considering configurations from both the predicted Pareto and the regions exhibiting high degree of randomness, we manage to refine the employed RSM towards higher quality solutions, therefore improving its prediction accuracy.

An additional contribution of this paper consists of the first comparative analysis among all the state-of-art techniques adopted so far in the field of supervised- HLS exploration. In Section V, we demonstrate how the proposed exploration strategy overcomes all the compared techniques, not only in terms of average accuracy (considering the same exploration runtime latency) but also in terms of robustness, by guaranteeing a fast convergence towards optimal solutions.

The remainder of the paper is organized as follows. Section II reviews on the related research on HLS and existing DSE techniques. In Section III, we provide a description on background material on the targeted problem and design space. Section IV introduces spectral analysis for DSE and describes the proposed SPIRIT methodology. In Section V, we provide experimental evaluation of the proposed methodology, while Section VI concludes the paper.

II. RELATED RESEARCH REVIEW

For more than 30 years, high level synthesis is an active field of research. During the evolution of the HLS research, we can recognize three main directions. The first direction focuses on the development of efficient scheduling algorithms [3], [6], [20], [21], [22] for generating optimized hardware from a behavioral description. This approach belongs to the historical roots of HLS, since its main focus is to provide algorithms that manage the internal representation of the application, to built efficient HLS tools.

The second direction is mainly driven by the advancements in the fields of micro-architecture and compiler technology. Several research works explore architectural optimizations for performance improvements, such as operation chaining [23], [24], CLK selection [25], timing-driven pipelining [26], memory hierarchy [27], off-chip communication management [28]. In addition, the impact of compiler-level transformations on the final datapath’s implementation has been examined in [29], [30], [11] [31].

The third direction is the most recent one and is guided by the increased maturity and the availability of the modern industrial HLS tools (such as CatapultC[4], AutoESL [32], CyberWorkBench [5], [33]) and academic tools (such as Legup [34], GAUT [1], SPARK [11]). This research path proposes the concept of supervised HLS, i.e. the usage of the HLS engine as a “black box” focusing on the multi-objective optimization of the design parameters for deriving optimal trade-offs of conflicting design metrics, such as delay, area. Adaptive simulated annealing has been proposed in [35], and later in [7], [36] combined with parameter clustering and kernels decomposition, to target the delay-area HLS optimization. The main limitations of these approaches are: first, they require to annotate each parameter with its typical contribution to the objective functions and second, they formulate the problem as minimization of an adaptive weighted sum of objectives [37]. Recently, Xydis et al. [9] proposed a compiler-in-the-loop HLS exploration approach guided by a gradient-based heuristic to manage an extended compiler/architectural instance of the HLS design space. This approach suffers from a lower efficiency, since it requires to evaluate a large fraction of the design space.

Motivated by the advances in research fields related to parametrized processor micro-architectures [38], [39], [40] and to platform-based design [15], [41], [42], machine learning techniques started gaining a lot of attention in the HLS community [43], [17], [44], [19] as an effective way to hide the complexity and to improve exploration’s time. In [43], the authors evaluate the efficiency of regression and decision-trees based models and propose an exploration strategy based on the incorporation of a genetic algorithm. Once trained, the model is embedded in the genetic optimizer, thus no refinement of the model is considered during the exploration. In [44], Xydis et al. proposed an RSM-based Pareto iterative refinement HLS exploration methodology for customizing the compiler/architectural parameters of hardware coprocessors. Zuluaga et al. [17] proposed a predictive design methodology that utilizes Gaussian regression for driving the sampling strategy. Uncertainty is used to drive model refinement, which however considers only design points found in the predicted Pareto curve. In [18], the authors extended the work in [17], by considering a Gaussian Process model that applies to the overall design space and uses the estimated uncertainty to iteratively classify the points as either non- or Pareto optimal. However, the evaluation of the technique has been performed on a limited size solution space, since despite the fact that Gaussian models enables to control the uncertainty bounds, they suffer from high runtime when the size of solution space scales up. Recently, Liu et al. [19] proposed an iterative refinement framework for supervised HLS that tries to a-priori minimize the uncertainty by utilizing Random Forests.
to model the targeted design space and it uses transductive experimental design (TED) for selecting representative configurations to train the predictive model. As noticed in [19], TED presents a poor scaling for extreme size design spaces and a randomized TED is proposed to alleviate this problem. However for large design spaces, the randomized version degenerates to an almost random sampling.

In existing approaches, iterative refinement is concentrated only on predicted Pareto configurations. Although refining the models with configurations found in the predicted Pareto set generates design solutions of improved quality [17], [18], [19], [44], there are no guarantees on biasing the model towards local optimality. In this paper, we differentiate from previous work by iteratively refining the model both with the predicted Pareto set, thus moving towards higher quality solutions, and with regions of the design space that exhibit high variance, thus enabling the model to capture the global behavior of the solution space. We extend [44] by introducing spectral analysis techniques for guiding the iterative refinement to account for configurations found in regions of the solution space that exhibit high variance/randomness. To demonstrate the efficiency of the proposed approach, we provide direct comparisons with respect to state-of-art exploration strategies tuned for the HLS-specific problems, considering very large solution spaces - orders of magnitude larger than those used in previous papers (e.g. [17], [18], [19]).

III. Problem Description

We focus on the fundamental architectural synthesis problem of discovering design solutions that derive Pareto optimal delay-area trade-offs. Specifically, given a design space \( D \), the target problem can be formulated as the following multi-objective optimization problem:

\[
\min_{x \in D} \left[ \begin{array}{c}
\text{Delay}(x) \\
\text{Area}(x)
\end{array} \right] \in \mathbb{R}^2
\]

subject to the following constraints:

\[
\left[ \begin{array}{c}
\text{Delay}(x) \\
\text{Area}(x)
\end{array} \right] \leq \left[ \begin{array}{c}
\text{Max_Delay} \\
\text{Max_Area}
\end{array} \right]
\]

The variable \( x \in D \) is a vector that refers to an instance of the HLS design space composed of the design parameters that are exposed to the designer. In our case, vector \( x \) includes both compiler and architectural parameters (Section III-B). HLS can be viewed as a function that maps each configuration vector, \( x \), to the 2-dimensional delay-area solution space. The delay metric is defined as \( \text{Delay}(x) = \#\text{Cycles}(x) \times \text{CLK}_{\text{period}}(x) \), thus referring to the overall execution latency of a hardware coprocessor synthesized according to the \( x \) configuration vector. Accordingly, the area metric, \( \text{Area}(x) \), refers to the overall area complexity, i.e. the area of the functional units, steering logic, registers, controller and memory interface, occupied by a hardware coprocessor synthesized according to the \( x \) configuration vector. The optimization goal is to find those configuration vectors, \( x \), that are mapped to Pareto (non-dominant) designs in the solution space. The optimization problem involves the minimization of multiple objectives (delay, area) making the definition of optimality not unique. As a matter of fact, a coprocessor which is the best from the performance point of view, can be the worst in terms of area complexity and vice-versa. The delay-area constraints incorporated in the problem formulation usually encode the specific requirements given by the target problem. For example, the constraints can refer to the maximum area available for including the coprocessor and/or to the maximum coprocessor latency sustainable by the system to satisfy application-level constraints. The exploration framework proposed in this paper supports the constraints on the objective functions, however as it will be shown in the rest of the paper it works also for the unconstrained case.

A. Target Coprocessor Architecture

In this section, we briefly introduce the target coprocessor architectural template (Figure 1) used in the proposed framework. The architectural template resembles a typical structure found in hardware accelerators and it is able to model architectures with various degrees of parallelism, including pipelining. It consists of (i) datapath components (ALUs and MULs), (ii) the register bank (scratch registers) for locally storing the intermediate results, (iii) the steering logic that moves data from the register bank to the datapath component and vice versa, (iv) the memory interface components (memory ports and Load/Store (LD/ST) units) to read/write data from/to the memory and (v) the control unit (FSM based) that generates control signals on a cycle-by-cycle basis.

We adopt the datapath area model, described in [9], which enables an accurate evaluation of the costs associated with each examined solution, unlike the simplified datapath models used in several previous exploration frameworks for coprocessor synthesis (usually based only on the number of ALU and multiplier, MUL, units) [6], [25], [29], [20]. The higher accuracy of the model is determined by the inclusion of components that can be derived by considering only decisions taken by the HLS engine, and thus not directly estimated from the configuration vector. In particular those components of the area model can be only derived after operation scheduling (e.g. controller complexity), register allocation (e.g. number of registers) and variable-to-register binding (e.g. number and dimension of multiplexers). Validation results of the adopted
analytical area model with respect to post-synthesis area estimation provided by Synopsys Design Compiler [12] tool can be found in [44].

B. Definition of the Design Space

In this paper, the design space considered for HLS exploration includes the complete set of tunable compiler- and architectural-parameters exposed to the designer in the SPARK-HLS tool [45]. However, the proposed methodology is quite general and it can be easily applied to alternative HLS tools that expose a different set of design parameters.

Compiler-level decisions (transformations) are imposed at the source code level of the behavioral description before the core HLS phases of operation scheduling and operation/variable binding take place. Compiler-level HLS decisions modify the structure of the intermediate representation that greatly affects the generated final datapath’s architecture. SPARK-HLS tool supports the following compiler transformations: (i) strength reduction, (ii) copy and constant propagation, (iii) common subexpression elimination and (iv) loop unrolling. Strength reduction, copy and constant propagation, and common sub-expression elimination are global scope transformations that target to the simplification of the overall intermediate representation. Loop unrolling decisions are applied locally to loop structures regarding which indexes (or index in case of single loop behavioral descriptions) and how many times (Loop Unrolling Factor, LUF) have to be unrolled. The LUF values depend on the number of loop indexes and their bounds found into the behavioral specification.

Architectural-level decisions directly impact on the final datapath structure. Design parameters concerning resource allocation and clock length selection are defined in this category. The designer provides the bounds of a range of interest in terms of minimum and maximum number of functional units and memory ports (i.e. Min./Max. #ALUs, Min./Max. #Muls, Min./Max. #MemPorts). Given the boundaries, allocation scenarios are derived from all the possible combinations inside that range. Each allocation scenario represents an instance of the resource constraints that the HLS operation scheduling procedure has to satisfy during delay minimization. On the other hand, CLK selection includes design decisions concerning the allocated clock frequency. We adopt the CLK selection methodology proposed by Blythe and Walker [25], which calculate a tight set of CLK candidates. The set of the candidate CLK lengths is derived by the ceiling of the integral divisors of the delay associated with each functional unit, while the minimum CLK length is determined by the minimum register-to-register delay of technology library. Operation chaining and operation multi-cycling are implicitly modeled by forcing the inclusion or the exclusion of specific CLK lengths from the candidate CLK set.

In this paper we are targeting the coprocessor synthesis problem, in which only the computationally intensive kernels are mapped as hardware coprocessors and not the overall application. For larger applications, with several functions to be mapped as hardware coprocessors, we suggest to adopt a strategy similar to the one proposed in [36] (see Section V-D) to limit the explosion of the design space size that would bring to an unaffordable exploration phase, even with efficient meta-heuristics.

IV. Spectral Analysis in Response-Surface Based Exploration

In this section, we introduce the proposed HLS exploration methodology called SPIRIT. First we describe how the HLS solution space can be represented as a signal and discuss why and how spectral analysis can be utilized to characterize the randomness of specific regions (Section IV-A). Then, we introduce the proposed iterative exploration framework for HLS design space (Section IV-B) and SPIRIT algorithm (Section IV-C), showing how RSM techniques and spectral analysis are exploited to efficiently approximate the Pareto design solutions. Finally, we conclude the section presenting the analysis of the SPIRIT parameters (Section IV-D) and RSM selection based on the model accuracy (Section IV-E).

A. Signal Representation of the HLS solution space

A signal is a sequence of data points, measured typically at successive points in time, spaced at uniform time intervals. We show that the targeted HLS solution space can be modeled as a discrete-time signal with finite duration, i.e. a time-series defined only at discrete points in time but not in magnitude. This will enable the application of well-known signal analysis techniques to be incorporated during exploration. Without loss of generality, we define the discrete-time signal $S$, with $N_D = 2^k$ data points\(^1\), as the signal representing a HLS solution. Let $s[i]$ be the magnitude of the signal representing a HLS solution after applying configuration $i$. We define $s[i]$ as single-dimension signal considering the area-delay product (ADP) that corresponds to configuration $i$, i.e.:

$$s[i] = A(i) \times D(i), \ i \in D_o \quad (3)$$

where $A(i)$, $D(i)$ are the area and delay of configuration $i$, respectively, and $D_o$ is an ordering of the design space $D$. The ordering of design space $D$, is an implicit\(^2\) user specified function referring to parameters’ order specified during the design space definition and corresponds to the traversal order of the HLS design space in case of a full-search evaluation of the solution space is applied.

We focus on spectral analysis for the HLS signal to estimate its frequency content and the related power spectrum. The power spectrum of a signal represents a characterization metric of its randomness. The more correlated or predictable is a signal, the more concentrated is its power spectrum. Conversely the more random or unpredictable is a signal, the more spreaded is its power spectrum. Thus, while for periodic signals, the power is concentrated in narrow bands of frequencies, indicating the existence of structure and the predictable character of the signal, for a purely random signal

\(^1\)In case that the cardinality of design space is not an integer multiple of 2, we can pad signal $S$ with virtual/dummy configurations equal to the last configuration in $S$.

\(^2\)Since conventional optimization heuristics do not evaluate exhaustively the overall design space, the user is usually not aware of how the exploration tool represents internally the design/parameter space.
the signal's power is spread equally in the frequency domain, indicating the lack of structure in the signal [46]. For the discrete-time signal \( s[i] \) of length \( N_D \) samples, the total energy is finite, \( \sum_{i=-\infty}^{+\infty} |s[k]|^2 < +\infty \). Thus, the discrete Fourier transform (DFT) is defined as \( N \) uniformly spaced spectral samples:

\[
S[f] = \sum_{0}^{N_D-1} s[i] \times \exp(-j2\pi fm)
\]  

(4)

The power spectrum analysis is concerned with the distribution of the signal power in the frequency domain. For the non-parametric HLS signal, \( S \), the power-spectral density is defined as:

\[
|P[f]|^2 = \frac{1}{N_D} \left| \sum_{0}^{N_D-1} s[i] \times \exp(-j2\pi fm) \right|^2 = \frac{1}{N_D} |S[f]|^2
\]  

(5)

Spectral analysis can be used to provide more localized information of the HLS signal. As in typical signal processing, the overall HLS signal can be partitioned in smaller windows, where each window refers to a specific region of the solution space. The power spectrum of each window characterizes the randomness of each region of the HLS solution space.

Figure 2(a) shows the ADP signal representation of two distinct regions (Signal A and Signal B) of the HLS solution space considering the DCT-2D kernel. We can observe that ADP Signal A exhibits higher variance/randomness than ADP Signal B. As shown in Figure 2(b), this higher randomness is efficiently captured by the power-spectrum of the two signals. For exploring the HLS solution space defined by the aforementioned regions and given that the behavior of region B is less random than the one of region A, exploration should focus its sampling strategy more in region A (rather than in region B) since more points are required to capture its behavior. Based on this observation, we propose a new RSM based exploration strategy (presented in Section IV-B) that, under a given number of maximum evaluation runs, it utilizes spectral analysis to characterize the randomness of different regions of the solution space and drives accordingly the iterative refinement procedure.

B. Proposed Exploration Strategy

The proposed exploration strategy relies on iterative HLS-based refinements for approximating the Pareto design solutions. It exploits two major concepts: (i) the usage of RSM techniques to analytically capture the relations between the design parameters and the area-delay response variables, thus speeding up the process associated with a HLS synthesis evaluation, and (ii) the introduction of spectral analysis techniques for guiding the iterative refinement procedure also including configurations found in "hard to predict" regions, i.e. regions of the solution space exhibiting high variance, to improve the knowledge on the design space.

Figure 3 shows the proposed HLS exploration framework. The upper-left part of the design flow includes the typical HLS phases. Given a behavioral specification in terms of C code, the phases of intermediate representation (IR) generation, operation scheduling, operation and variable binding are performed for generating the coprocessor's RTL description. Then, during the pre-synthesis area estimation phase, the generated RTL description together with a gate-level pre-characterized resource library are used as inputs for estimating the area complexity of the architectural solution. The design configurations passed to the HLS phases have been selected by means of (i) a Design of Experiments (DoE) module (used only in the initial phase) and (ii) a cooperative RSM-based Pareto- and spectral-aware refinement technique (bottom part of the flow). This combined approach, representing the feedback loop of the framework, is the core of the proposed exploration strategy.

The analytical approximation of the solution space, that is the entry point for the lower part of the framework, is obtained by means of a RSM trained by using area and delay results of previously explored solutions. The RSM-based Pareto-aware refinement, bottom-left part of the flow, enables to selectively refine the solution set with points predicted as Pareto over the entire design space, thus moving towards higher quality solutions. The spectral-aware refinement, bottom-right part of the flow, is also applied to the approximated solution space. However, the design space is incrementally partitioned at each
iteration of the exploration strategy to identify high variance regions. Then, design points located in high-variance regions of the design space are added to the set of points to be evaluated by the HLS tool to improve the prediction accuracy of the approximated solution space on the next iteration.

### C. The SPIRIT Algorithm

The iterative exploration strategy introduced in the previous section to solve the supervised HLS problem is based on the SPIRIT algorithm. SPIRIT is described with more details in Algorithm 1. It accepts as inputs (i) the application to be synthesized, \( A \), (ii) the definition of the design space, \( D \), (iii) the number of iterations, \( it_{\text{max}} \), that correspond to the number of refinement rounds to be invoked, (iv) the maximum number of HLS evaluations per iteration, \( nsynth_{\text{max}} \), (v) the initial partition window size, \( w_{\text{size max}} \), for signal representation of the HLS solution space, and (vi) the response-surface model, \( RSM \). The product \( it_{\text{max}} \times nsynth_{\text{max}} \) defines the budget constraint regarding the maximum number of HLS evaluations.

First, the overall design space, \( D \), is sampled with a uniform random distribution (line 2). A set of initial design configurations, \( R \), is extracted from \( D \). Each configuration is synthesized through HLS and the actual performance and area metrics are derived, providing an initial coarse view of the target design space (line 3). \( F \) represents the database containing all the information (configurations, area/delay metrics) about the configurations synthesized through HLS. Using \( F \) as training vectors, the RSM is trained by generating a prediction archive \( M \) of the solution space, considering all the possible configurations annotated with the predicted area and delay metrics (line 4). \( M \) corresponds both to the predicted solution space and to the overall HLS signal. Signal \( M \) is then partitioned in smaller windows, \( M_i \), according to the partition window size parameter, \( w_{\text{size}} \), for enabling the characterization of regions of high randomness (lines 5-6). At each iteration of the exploration strategy, the size of the partition window is incrementally refined towards smaller granularities, to enable more localized spectral analysis as the exploration proceeds (lines 27-28). Without loss of generality, the proposed exploration algorithm implements the case that \( w_{\text{size}} \) is reduced to half between two successive iterations, i.e. doubling the number of windows that the HLS signal is partitioned.

After acquiring a first prediction of the overall solution space, the exploration strategy enters to the iterative refinement phase. As mentioned, the refinement is done on the predicted Pareto design points and on sample points from regions of the solution space that exhibit higher randomness. Regarding the Pareto-aware iterative refinement, we adopt an approach similar to [15], [44]. First, the approximate Pareto configurations, \( L \), are extracted by applying a full search on the predicted solution space, \( M \) (line 8). The predicted Pareto configurations are evaluated through HLS to acquire the actual metrics, \( F_L \) (line 9) and then inserted in the training database \( F \) for the next refinement of the predictive model \( M \) (line 10).

Given the maximum number of HLS evaluations per iteration, \( nsynth_{\text{max}} \), and assuming |\( L | \) configurations to be included in the approximate Pareto set, the maximum number of configurations to be used for spectral-aware refinement, \( nsynth_{\text{spectral}} \), is calculated (line 12). At each iteration, the

---

**Algorithm 1: Pareto and Spectral Aware Iterative Refinement Exploration Strategy**

Input: Application source code: \( A \)

Input: Design-space definition: \( D \)

Input: Number of iterations: \( it_{\text{max}} \)

Input: Number of HLS evaluations per iteration: \( nsynth_{\text{max}} \)

Input: Initial partition window size: \( w_{\text{size max}} \)

Input: Response-surface model: \( RSM \)

Output: Approximate Pareto set: \( L \)

1. \( nsynth \leftarrow N \)
2. \( R \leftarrow \text{RandomDoE}(D, nsynth) \)
3. \( F \leftarrow \text{HLS}(A, R) \)
4. \( M \leftarrow \text{Train}(RSM, F) \)
5. \( w_{\text{size}} \leftarrow w_{\text{size max}} \)
6. \( \{M_i\} \leftarrow \text{Windowing}(M, w_{\text{size}}), i \in \{0, \ldots, \lceil \frac{w_{\text{size}}}{w_{\text{size}} - 1} \rceil \} \)
7. For \( i = 1 \) to \( it_{\text{max}} \) do
8. \( L \leftarrow \text{ParetoFilter}(M) \)
9. \( F_L \leftarrow \text{HLS}(L) \)
10. \( F \leftarrow F \cup F_L \)
11. \( nsynth_{\text{spectral}} \leftarrow |L| \)
12. \( nsynth_{\text{spectral}} \leftarrow \text{nsynth}_{\text{max}} \times nsynth_{\text{parareto}} \)
13. Foreach \( M_i, i \in \{0, \ldots, \lceil \frac{w_{\text{size}}}{w_{\text{size}} - 1} \rceil \} \) do
14. \( \{P_k\} \leftarrow \text{CalculatePowerSpectrum}(M_i), k \in 0 \ldots \lceil \frac{w_{\text{size}}}{w_{\text{size}} - 1} \rceil \)
15. \( \text{Spectrum}^{\text{pow}} \leftarrow \sum_{k=0}^{\lceil \frac{w_{\text{size}}}{w_{\text{size}} - 1} \rceil} P_k \)
16. \( \text{TotalSpectrum}^{\text{pow}} \leftarrow \text{TotalSpectrum}^{\text{pow}} + \text{Spectrum}^{\text{pow}} \)
17. End
18. Foreach \( i \in \{0, \ldots, \lceil \frac{w_{\text{size}}}{w_{\text{size}} - 1} \rceil \} \) do
19. \( nsynth_i \leftarrow \text{nsynth}_{\text{spectral}} \times \frac{\text{TotalSpectrum}^{\text{pow}}}{\text{Spectrum}^{\text{pow}}} \)
20. \( R_i \leftarrow \text{RandomDoE}(M_i, nsynth_i) \)
21. \( R_i \leftarrow \text{EliminateDuplicate}(R, R_i) \)
22. \( R \leftarrow R \cup R_i \)
23. \( F \leftarrow \text{HLS}(R) \)
24. \( F \leftarrow F \cup F_L \)
25. End
26. \( M \leftarrow \text{Train}(RSM, F) \)
27. \( w_{\text{size}} \leftarrow \frac{w_{\text{size}}}{2} \)
28. \( \{M_i\} \leftarrow \text{Windowing}(M, w_{\text{size}}), i \in \{0, \ldots, \lceil \frac{w_{\text{size}}}{w_{\text{size}} - 1} \rceil \} \)
29. End
30. \( M \leftarrow \text{ParetoFilter}(M) \)
$nsynth_{spectral}$ represents the overall budget of configuration samples to be allocated to the instantiated partition windows of the HLS solution space, $M_i$, that exhibit the higher randomness, thus being hard to be accurately predicted. For each partition window, $M_i$, the power spectrum coefficients, $P_k$, $k \in 0 \ldots \frac{M_i}{2}$, of the corresponding HLS signal are calculated (line 14). The computational cost of the spectrum calculation depends only by the design space size and not from the partition window size. The power spectrum of each region, $Spectrum_{pow}$, is calculated as the sum of the spectrum’s coefficients (line 15), while the total power spectrum of the overall HLS signal is the summation of the power spectra that characterize each HLS partition window (line 16).

The metric $\frac{Spectrum_{pow}}{TotalSpectrum_{pow}}$ reflects the contribution that the randomness of each HLS partition window $M_i$ implies to the overall randomness of the HLS solution space. Since the higher the contribution the harder to predict is the HLS region, we use the aforementioned metric to guide the allocation of configuration samples, $nsynth_i$, in each HLS region (line 19). Each partition window $M_i$ is randomly sampled with $nsynth_i$ configurations (line 20). The newer configurations (those never evaluated before) are synthesized through the HLS engine and, after acquiring their actual area-delay measurements, they are inserted into the training database, $F$ (lines 21-22). The Pareto- and spectral-aware training points of $F$ are used for refining the RSM and generating the new prediction of the HLS solution space, $M$ (line 26). The refined $M$ is incrementally partitioned in the number of windows that corresponds to the new value of partition size, $w_{size}$, and forms the input for the next iteration (line 28). At the end of the exploration, the final approximated Pareto set is derived for the designer (line 30).

D. Parameters Analysis

The SPIRIT algorithm presents a set of input parameters that require to be analyzed before fixing their values. Figure 4 shows the contour maps representing the impact of the input parameters: (i) number of iterations $it_{max}$ and (ii) initial partition window size $w_{size_{max}}$ with respect to the targeted problem size. We consider various problem sizes based on down-scaled instances of the original design space defined for the JPEG 2D-DCT kernel, and we examine how the aforementioned parameters affect the exploration’s efficiency in terms of the ADRS [47] metric (Average Distance from Reference Set, see Section V-A). We report average ADRS values considering 100 HLS evaluations per iteration. From Figure 4(a), it is observed that for problem sizes up to 200000 solutions, allocating either a low or a high number of iterations deliver high ADRS values. Moving towards larger problem sizes, there is a benefit regarding ADRS when allocating higher number of iterations, due to the fact that a better refinement is performed. The same trend it is also observed in Figure 4(b) for the initial $w_{size_{max}}$, i.e. the exploration of large design spaces is benefited by larger initial partitions. A large initial partition window size enables a uniform sampling of the design space to be performed during the first iterations of the algorithm, together with a more concentrated one during the last iterations. On the other hand, small initial partition sizes concentrate from the very beginning the distribution of HLS evaluations in specific regions of the solution space, increasing the possibility to leave large regions of the design space unexplored. From Figure 4(b), it can be observed that for $w_{size_{max}}$ around 2048-4096, there is a region exhibiting high ADRS accuracy for design space sizes of $6 \times 10^5$. However, the specific $w_{size_{max}}$ is not robust enough to be considered as a general trend, since moving towards bigger design space sizes considering the specific $w_{size_{max}}$ parameter values, the ADRS accuracy is worsened.

E. RSM selection

In the proposed methodology, RSMs are used to forecast the unknown response of the HLS system. In this section, we analyze the behavior of several RSMs for selecting the most suitable analytical description for the targeted design space and HLS engine.

The RSM selection is based on the minimization of the accuracy error and on the maximization of the convergence behavior. Specifically, we considered six RSM models (four regression-based and two interpolation-based): (i) Linear Regression [48], (ii) Spline-based Regression [49], (iii) Artificial Neural Networks (ANNs) [50], (iv) Random Forests (RFs) [51] (v) Shepard-based Interpolation [52] and (vi) Radial Basis
Fig. 5. Validation results for the selected RSM methodologies considering different Box-Cox transformations

Functions (RBFs) [53]. The RSMs have been validated by means of the Monte Carlo cross-validation procedures [54]. We trained the selected set of RSMs by using as training-set (known points) a variable number of random configurations, from 200 to 1800 (Figure 5), and we evaluated the average normalized error, on both area and delay metrics, computed by the RSMs over the remaining configurations of the entire design space. In this context, to further improve the prediction accuracy, we introduced a Box-Cox power transform on each sample $y_{k}$ of the observed data [48] in order to reduce the data variation and improve the correlation between parameters and response functions. The power transformation is defined as a continuously varying function, with respect to the power parameter $\xi$:

$$y_{k}^{(c)} = \begin{cases} 
(y_{k}^{\xi} - 1)/\xi, & \text{if } \xi \neq 0 \\
\log y_{k}, & \text{if } \xi = 0 
\end{cases}$$

(6)

We examined a family of transformations as potential candidates $\xi \in \{1, 0.5, 0\}$. For the case under analysis, we found that the log case outperforms all the selected Box-Cox transformations in terms of approximation error for the considered design space for almost all the RSMs and training set size.

The analysis showed that the accuracy of linear and spline-based regression models remains almost constant in the selected range of training set size. In particular, for the targeted set of Box-Cox transformations, the average normalized error remains between 25% and 13.5% for the linear regression (Figure 5(a)), while between 21% and 11% for the spline-based regression (Figure 5(b)). Concerning linear regression, we considered also a second order model without interaction that presented an average prediction error ranging from 2% to 4% better than the first order model including interaction (considering the same Box-Cox transformations). For the remaining RSMs, we observed an estimation error decreasing from 26-20% to 16-14% when considering the Shepard’s interpolation (Figure 5(e)), from 20-15% to 11-8.5% when considering the ANN (Figure 5(c)) from 20-17% to 9.5-8% when considering the Random Forest (Figure 5(d)) and from 16-13% to 9-8% for the radial basis functions (Figure 5(f)), depending on the adopted Box-Cox transformation.

Figure 6 shows the validation results considering the best configuration from each examined RSM with respect to estimation accuracy. For the given set of data, we can see that RBF, RF and ANN present the best estimation accuracy when increasing the size of the training data. However, radial basis functions, that always outperform the Random Forest, also present a very consistent decrement of the validation error, making its behavior more stable with respect to the artificial neural network. Thus, our analysis can conclude that the RBF model is the most suitable RSM to be used in the proposed RSM-based exploration strategy for the target problem, and it has been selected to be used in the following Experimental Results section. The reason behind this result can be found on the peculiar characteristic of the RBFs in performing an accurate multivariate interpolation even when the data are distributed in a non-uniform manner [55].

V. EXPERIMENTAL RESULTS

In this section, we evaluate the efficiency of the proposed HLS exploration strategy. First, we compare the proposed approach to the Zero-Gradient exploration [9], a state-of-art exploration strategy tailored to the same problem of co-

For readability issues, only the configurations presenting the best performance for each RSM has been plot in Figure 5.

For the case under analysis, we found that the log case outperforms all the selected Box-Cox transformations in terms of approximation error for the considered design space for almost all the RSMs and training set size. In particular, for the targeted set of Box-Cox transformations, the average normalized error remains between 25% and 13.5% for the linear regression (Figure 5(a)), while between 21% and 11% for the spline-based regression (Figure 5(b)). Concerning linear regression, we considered also a second order model without interaction that presented an average prediction error ranging from 2% to 4% better than the first order model including interaction (considering the same Box-Cox transformations). For the remaining RSMs, we observed an estimation error decreasing from 26-20% to 16-14% when considering the Shepard’s interpolation (Figure 5(e)), from 20-15% to 11-8.5% when considering the ANN (Figure 5(c)) from 20-17% to 9.5-8% when considering the Random Forest (Figure 5(d)) and from 16-13% to 9-8% for the radial basis functions (Figure 5(f)), depending on the adopted Box-Cox transformation.

Figure 6 shows the validation results considering the best configuration from each examined RSM with respect to estimation accuracy. For the given set of data, we can see that RBF, RF and ANN present the best estimation accuracy when increasing the size of the training data. However, radial basis functions, that always outperform the Random Forest, also present a very consistent decrement of the validation error, making its behavior more stable with respect to the artificial neural network. Thus, our analysis can conclude that the RBF model is the most suitable RSM to be used in the proposed RSM-based exploration strategy for the target problem, and it has been selected to be used in the following Experimental Results section. The reason behind this result can be found on the peculiar characteristic of the RBFs in performing an accurate multivariate interpolation even when the data are distributed in a non-uniform manner [55].

V. EXPERIMENTAL RESULTS

In this section, we evaluate the efficiency of the proposed HLS exploration strategy. First, we compare the proposed approach to the Zero-Gradient exploration [9], a state-of-art exploration strategy tailored to the same problem of co-
We selected a representative set of benchmarks found into real-life DSP and multimedia applications. The benchmark suite consists of 8 computationally intensive kernels of various sizes and complexities: (i) the least-mean-square (LMS) adaptive filter, (ii) the 1D DCT, (iii) the YUV to RGBA filter, (iv) the MESA 4×4 Matrix Multiplication algorithm, (v) the the 2D DCT kernel of JPEG, (vi) the 2D Inverse DCT kernel of MPEG-2, (vii) the Gauss Blur image transformation from Cavity Detector algorithms, (viii) the 8×8 Sobel edge detection image filter. In Table I, we report the complexity of each kernel, in terms of number of basic blocks and operations (respectively #BB and #Op), ranges of the exploration parameters considered for each case, as well as size of the overall solution space.

To assess multi-objective optimization heuristics a measure of the distance between the exact and the approximate Pareto sets should be introduced. In this paper we used the average distance from reference set (ADRS) [47]. The ADRS is used to measure the distance between the exact Pareto set⁵, Π, and the approximate Pareto set, A:

$$\text{ADRS}(\Pi, A) = \frac{1}{|\Pi|} \sum_{x_R \in \Pi} \left( \min_{x_A \in A} \{\delta(x_R, x_A)\} \right)$$  \hspace{1cm} (7)

where $\delta$ is a measure of the normalized distance in the objective function space of two configurations:

$$\delta(x_R, x_A) = \max_{j=1,\ldots,m} \left\{ 0, \frac{f_j(x_A) - f_j(x_R)}{f_j(x_R)} \right\}$$  \hspace{1cm} (8)

where $f_j$ represents the objective functions to be minimized (area and delay). ADRS is measured in terms of percentage. The higher the ADRS, the worse is $A$ with respect to $\Pi$.

B. Comparative Results: Proposed vs. Zero-Gradient Exploration Strategy

In this section, we compare the proposed spectral-aware approach to the Zero-Gradient exploration strategy [9], which forms an HLS specific optimization tailored to the unified compiler and architectural HLS design space. Zero-gradient HLS exploration is controlled by two parameters, (i) the gradient-depth parameter, $Depth_k$, indicating how many design points the search algorithm should look ahead and (ii) the amortization parameter $a^T$ identifying at which design point the exploration of a new region should start. As shown in [9], the larger the $Depth_k$ and the $a^T$, the more accurate the approximated Pareto with respect to the ADRS metric, at the expense of an increased exploration’s runtime.

In this section, we consider two instances of the Zero-Gradient algorithm: the first one tuned for ADRS accuracy (Zero-Gradient Accurate - $a^T = 80\%$ and $Depth_k = 4$), while the second one tuned to reduced the exploration runtime (Zero-Gradient Fast - $a^T = 20\%$ and $Depth_k = 2$). Regarding the proposed SPIRIT exploration strategy, we consider a single instance across all the benchmarks according to the following parameter values: (i) number of iterations: 4, (ii) number of

³The exact Pareto curve has been extracted from exhaustive exploration based on an actual HLS campaign over the design space.

---

**TABLE I**

<table>
<thead>
<tr>
<th>Kernel Characterization and Exploration Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kernel</strong></td>
</tr>
<tr>
<td>LMS</td>
</tr>
<tr>
<td>JPEG 1D DCT</td>
</tr>
<tr>
<td>YUB2RGBA</td>
</tr>
<tr>
<td>MatMul</td>
</tr>
<tr>
<td>JPEG 2D DCT</td>
</tr>
<tr>
<td>MPEG 2D IDCT</td>
</tr>
<tr>
<td>GAUSS BLUR 2D</td>
</tr>
<tr>
<td>SOBEL 2D</td>
</tr>
</tbody>
</table>

---

**Fig. 6.** Comparative prediction accuracy for the best configuration of the selected RSM methodologies
samples per iteration: 100, (iii) initial partition size: 8192, (iv) RSM: Radial Basis Function.

Figure 7 shows the comparative results between SPIRIT and Zero-Gradient (both Accurate and Fast version) exploration strategies regarding the ADRS metric. On average, the proposed approach delivers an ADRS of 1.7%, clearly outperforming the Zero-Gradient strategy that reaches only an ADRS of 3.5% and 8.5% respectively for Accurate and Fast versions. More specifically, while SPIRIT outperforms the Fast version of the Zero-Gradient in all analyzed kernels, with respect to the Accurate instance it reports significant ADRS reduction, close to 80%, for four benchmarks (LMS, MPEG IDCT, Gauss Blur and Sobel). Similar levels of ADRS efficiency are reported for the DCT-1D and YUV2RGB kernels, while Zero-Gradient Accurate outperforms the proposed approach for the MatMul and JPEG 2D-DCT kernels.

However, these results should be evaluated by taking also into consideration the exploration time required by the methods. In fact, while the Accurate version of the Zero-Gradient approach evaluates on average more than 35% of the solution space, the proposed SPIRIT approach evaluates only 0.4% demonstrating a reduced exploration effort.

In particular, Figure 8 shows the exploration speedup achieved by the SPIRIT strategy over the two Zero-Gradient versions for the entire kernel set. SPIRIT outperforms in all cases the Zero-Gradient runtime, except for the LMS kernel and MatMul (with respect to the Fast Zero Gradient only). Additionally, for 5 out of 8 kernels, the proposed approach shows a speedup larger than 10 and for the JPEG 2D-DCT case it is larger than 100. JPEG-2D-DCT is one of the two kernels where SPIRIT was outperformed in terms of ADRS by the Zero-Gradient Accurate. Looking at the 3 cases where the speedup is less than 10, including LMS where the proposed approach demonstrates a slow-down, we notice that they refer to the simpler kernels in the benchmark set (see Table I). The reason behind the reduced speedup can be found in the additional overhead introduced by the RSM training that becomes significant when the complexity of the kernel is reduced and thus the actual time of the HLS process is not so high.

Concluding, to give also a practical view of the comparison between SPIRIT and Zero-Gradient approaches, we add some data about the exploration runtime\(^6\) to better understand the behavior shown in Figure 8. The Accurate version of the Zero-Gradient strategy, comparable in terms of ADRS with respect to the proposed methodology, takes on average 2.4 days to conclude the exploration. SPIRIT requires only 21 minutes on average for the same task. The difference is even more evident for the case of JPEG 2D-DCT where the exploration runtime are 16.1 days for the Zero-Gradient Accurate and 1.2 hours for SPIRIT.

\(^6\)The exploration runtime includes (i) all the HLS steps, excluding the front-end and back-end time to read the input file and writing the output HDL file, for each configuration and (ii) the RSM training, if needed by the strategy.

\(^7\)The need of having a pure multi-objective search instead of minimizing a weighted sums of the objectives is well explained in [37].

C. Comparative Results: Proposed vs. Meta-Heuristic Exploration Strategies

In order to evaluate the efficiency of the proposed SPIRIT strategy, we compare it with respect to state-of-art meta-heuristics for multi-objective optimization used for the HLS exploration problem, namely: (i) the Multi-Objective Simulating Annealing ([57] (MOSA) that is a pure\(^7\) multi-objective version resembling the strategy used in [35], (ii) the Response-Surface Pareto Iterative Refinement ([15] (RESPIR) [44], (iii) the Pareto-Active Learning algorithm ([18] (PAL) as implemented in [58] and (iv) the learning-base method presented in [19], where a Randomized Transductive Experimental Design ([59] is coupled with Random-Forest [60] as predictive model (RTED-RF). For a fair comparison, we run the corresponding explorations several times (at least 25) by varying, where available, the parameter configurations and repeating the initial sampling to avoid the possibility of a biased behavior. MOSA has been configured as follows: \#epochs \(\in \{2, 18, 35\}\), epoch length \(\in \{10, 35, 60\}\), annealing coefficient \(0.75\). The Gaussian Process models in PAL has been tuned by varying the epsilon parameter \(\epsilon \in \{0.8...0.04\}\) with a step of 0.05 and initial training size of 50 points. SPIRIT and RESPIR have been configured by plugging in the RBF model selected from the results presented in Section IV-E, combined with a Random DoE strategy for selecting the initial set of 50 points (which roughly corresponds to the sum of all the levels of the parameters of the design space). Finally, the initial
Transductive Experimental Design of RTED-RF has been applied to a selection composed of 5000 randomly selected designs to obtain the same initial set of 50 points as for the proposed method. Concluding, the initial sampling has been repeated 5 times for those strategies including configurable parameters (MOSA and PAL) and 25 times for the others (SPIRIT, RESPIR and RTED-RF).

We present results in an aggregated manner for the selected benchmarks. For this, we considered a unified design space across applications shown in Table II composed of 51840 design alternatives. We compare the five exploration strategies in terms of: (i) accuracy of results given by the Average Distance from Reference Set (ADRS) metric and (ii) number of synthesized solutions.

Figure 9 shows the ADRS (up to 10%) with respect to the exact Pareto-front versus the number of synthesized configurations (up to 520, corresponding to 1% of the target design space) for all instances of the exploration strategies. The proposed methodology dominates almost completely all the other exploration strategies. For the same or even smaller number of synthesized configurations, SPIRIT delivers design solutions that are closer to the exact Pareto set with respect to the other DSE strategies. Additionally, Figure 9 shows how SPIRIT is capable in all the cases (except for one) to reach an ADRS value within 2% (and close to 0.5% on average).

Figure 10 highlights this phenomena showing the ADRS distribution by means of a boxplot graph. The considered solutions are filtered in the area between 130 and 260 synthesized configurations (resulting between 0.25% and 0.5% of the entire design space) thus capturing most of the exploration instances. It is evident that while RESPIR, RTED-RF and SPIRIT are close to the Pareto curve, the proposed strategy exhibits higher robustness in respect to ADRS. This is the result of the enhancement introduced by Spectral-Aware Refinement over the basic iterative refinement strategies within both RESPIR and RTED-RF. On average, RESPIR results to be very accurate in finding the Pareto curve (average ADRS close to 1%), however it is strongly dependent on the initial random sampling that sometime can bring to local minima. This phenomenon is a bit less evident on RTED-RF since it is supported by the initial Randomized Transductive Experimental Design that helps to generate a better distribution of the initial configurations. On the other side RTED-RF suffers of a less accurate predictive model (Random Forests) especially for a very low number of training configurations (see Figure 6). MOSA and PAL are far from the accuracy levels demonstrated by the other methods in the considered interval of synthesized solutions. The main reason is that both methods require more design points to be synthesized to converge. While MOSA is not supported by any predictive model to discover at least the global trends of the objective functions, PAL requires several design evaluations for enabling the Gaussian Processes to predict an accurate point classification in large design spaces.

D. The MPEG-2 Decoder Case Study

To further show the effectiveness of the proposed methodology, in this section we present the results obtained for an MPEG-2 decoder case study. We show how the SPIRIT exploration strategy can be utilized to address the coprocessor design for complex applications when more than one kernels can be mapped to hardware. We adopt a decomposition-based strategy, utilizing the concept of divide and conquer exploration introduced in [36], in which each code block (i.e. a function defining a coprocessor in our case) is first explored separately to extract the respective Pareto curves that in a second step are further combined/merged to a reduced solution space defined only by combinations of per kernel Pareto design configurations. The final optimal configurations of each explored coprocessor are derived through a Pareto filtering on

---

TABLE II

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Min.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compiler Level:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strength Reduction</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Copy and Constant Propagation</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Sub-expression Elimination</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Outer loop unrolling factor</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Inner loop unrolling factor</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td><strong>Architectural Level:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of ALU</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>No. of MUL</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>No. of Mem. ports</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Clock period (ns)</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Operation Chaining</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Operation Multicycling</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

---

It is the maximum size of the randomized set without running out-of-memory on a machine with 8GB of memory to compute the DoE.
TABLE III

<table>
<thead>
<tr>
<th></th>
<th>2D-IDCT</th>
<th>YUV2RGBA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compiler Parameters:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Strength Reduction</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Copy and Constant Propagation</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sub-expression Elimination</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Outer loop unrolling factor</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Inner loop unrolling factor</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Architectural Parameters:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of ALU</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>No. of MUL</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>No. of Mem. ports</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Clock period (ns)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Operation Chaining</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Operation Multicycling</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Metrics:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Area [um2]</td>
<td>81264</td>
<td>68324</td>
</tr>
<tr>
<td>Delay [ns]</td>
<td>6178</td>
<td>85</td>
</tr>
</tbody>
</table>

This reduced solution space. This strategy represents the most efficient way to deal with the problem when using commercial HLS tools such as Vivado [56] or CyberWorkBench [5], which adopt per kernel parameter annotation.

Specifically, the main goal for the case study is the minimization of the average time needed for elaborating each pixel to minimize application delay (i.e. maximize the application throughput). The minimization problem has been constrained by an area value available for the coprocessor synthesis that should be less than 150000 um^2. The MPEG-2 decoder is composed of several kernels, however we focus our attention on the design of two of them that are eligible as HW coprocessors: the 2D inverse DCT (2D-IDCT) and the YUV to RGBA (YUV2RGBA) space converter. Following this decomposition, the average pixel elaboration time for the algorithm is composed of the sum of three components: the software part and the two accelerators. Since the software part of the application is not impacted by the HW design of the coprocessors (once the HW/SW interfaces have been decided), the minimization of the average pixel elaboration time can be seen as the minimization of the sum of the average elaboration time for the two accelerators.

According to [36], we solve the problem by applying separately the SPIRIT methodology to the two accelerators to find the Pareto curves. In particular, Figures 11(a) and 11(b) show the approximate Pareto curves for 2D-IDCT and YUV2RGBA compared to the exact Pareto curves, considering the design space of Table II. Despite of the small number of evaluations, respectively after 565 and 568 evaluations that globally required around 35 minutes, in both cases the proposed methodology is very close to the exact Pareto curve.

Then, we merged through a Cartesian product operator and filtered the results of the two explorations , into a single Pareto curve representing the combination of 2D-DCT and YUV2RGBA coprocessors in the area/delay (average latency per pixel) space, Figure 11(c). The delay values for the 2D-IDCT are divided by 64 since the accelerator works on a 8x8 matrix of pixels.

Finally, we selected the design configurations (one for each kernel) that combined together minimize the delay while respecting the area constraint. The selected solution is highlighted in Figure 11(c) and the configurations of the 2D-IDCT and YUV2RGBA accelerators are reported in Table III. In particular, although both kernels include multiplications with constant coefficients, the selected configurations do not include any multiplier, thus saving area. Moreover, we can notice that the strength reduction and constant propagation play a significant role for the synthesis of the analyzed coprocessors since they are included in both cases. Finally, we can note the total area is not equally split between the two accelerators, thus underlying the need of the final combined analysis for determine the optimal configuration. Despite of including a lower number of ALU and memory ports with respect to the YUV2RGBA, the 2D-IDCT coprocessors results in larger area mainly due to the higher complexity of the control part.

VI. CONCLUSION

In this paper, we investigated the problem of supervised HLS for coprocessor customization. A meta-model assisted exploration framework has been presented leveraging Response Surface Models to analytically represent the behavior of a HLS engine for predicting the quality of the design points without resorting to costly architectural synthesis. We proposed, for the first time, the usage of spectral analysis for characterizing the randomness of the solution space. We show that spectral analysis can be effectively incorporated within an HLS exploration framework enabling identification of hard-to-predict regions of the solution space. A novel exploration strategy has been developed that effectively exploits spectral analysis through an adaptive sampling strategy taking into...
consideration both the Pareto and the spectrum distribution over the solution space, to iteratively refine the accuracy of the approximate Pareto curve towards the exact Pareto frontier. Extensive experimental evaluation, in terms of Pareto approximation accuracy and exploration runtime, showed that SPIRIT outperforms all the recently introduced HLS-specific exploration heuristics. Finally, we also presented a case study on MPEG-2 decoder to outline how the proposed exploration technique can be incorporated in HLS tool-chains supporting per kernel parameter annotation for more complex applications.

REFERENCES

[34] Legup HLS, http://legup.epfl.ch.


