A Meta-Model Assisted Coprocessor Synthesis Framework for Compiler/Architecture Parameters Customization

Sotirios Xydis, Gianluca Palermo, Vittorio Zaccaria, Cristina Silvano

Politecnico di Milano - Dipartimento di Elettronica e Informazione
E-mail: {xydis, gpalermo, zaccaria, silvano}@elet.polimi.it

***Abstract*—Hardware coprocessors are extensively used in modern heterogeneous systems-on-chip (SoC) designs to provide efficient implementation of application-specific functions. Customized coprocessor synthesis exploits design space exploration to derive Pareto optimal design configurations for a set of targeted metrics. Existing exploration strategies for coprocessor synthesis have been focused on either time consuming iterative scheduling approaches or ad-hoc sampling of the solution space guided by the designer's experience. In this paper, we introduce a meta-model assisted exploration framework that eliminates the aforementioned drawbacks by using response surface models (RSMs) for generating customized coprocessor architectures. The methodology is based on the construction of analytical delay and area models for predicting the quality of the design points without resorting to costly architectural synthesis procedures. Various RSM techniques are evaluated with respect to their accuracy and convergence. We show that the targeted solution space can be accurately modeled through RSMs, thus enabling a speedup of the overall exploration runtime without compromising the quality of results. Comparative experimental results, over a set of real-life benchmarks, prove the effectiveness of the proposed approach in terms of quality improvements of the design solutions and exploration runtime reductions. An MPEG-2 decoder case study describes how the proposed approach can be exploited for customizing the architecture of two hardware accelerated kernels.**

I. Introduction

The increasing market demands in the field of consumer electronics, impose 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 mature phase [1], HLS is the driving force of design abstraction, offering an automated path from high level algorithmic specifications down to circuit level implementations. Since a large number of architectural configurations became explorable, designers have faced the problem of reasoning on differing trade-offs. Thus, efficient design space exploration (DSE) methodologies accompanied with automated tools are of great importance for fast and accurate evaluation of the solution space [2].

The efficiency of the exploration phase for coprocessor synthesis is highly affected by two factors: (i) the core HLS optimization algorithms, 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. While the inclusion of gate-level synthesis information during exploration enables accuracy, it remains an extremely time consuming task. To reduce exploration delay, designers usually adopt a less time consuming exploration approach by shortening the exploration loop to iterate through the HLS design space. Nowadays, it is common practice [3]–[8] to adopt an iterative exploration approach at the level of HLS by evaluating the quality of the design solutions during the pre-synthesis phase. In this way, only a reduced set of design configurations forming the Pareto-front of the high level exploration is propagated down to gate-level synthesis tools for further refinement. For example, the execution time required for a single run of the SPARK's HLS engine [9] for the Inverse Discrete Cosine Transform kernel from the MPEG2 application (when unrolling the inner and outer loops) is approximately 8.5 minutes, while for a single gate-level synthesis evaluation, by using Synopsys Design Compiler [10], the time required is 23 minutes. Even in the case of a high level exploration loop, the aggregated delay of iteratively using the core HLS engine imposes large 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 [7], [8], [11], [12] have proven that this type of exploration simplifications inherently reduces the effectiveness of the DSE and thus the optimality of the approximated Pareto curve.

In this paper, we address the coprocessor synthesis and customization introducing Response Surface Methods. The innovativeness of the approach consists of analytically modeling and predicting the behavior of the synthesis engine, to efficiently explore the design space in terms of both architectural and compiler parameters. The main objective is not to target the complete replacement of the synthesis engine, but to introduce an abstract meta-layer that enables the reduction of times the proposed exploration framework has to resort to the costly architectural synthesis. More in detail, we propose (i) the construction of delay and area predictive RSM models to capture the behavior of the core synthesis optimization algorithms, namely operation scheduling, resource binding and register allocation and (ii) the development of an exploration framework for coprocessor customization, which exploits the predictive meta-models to iteratively refine and optimize towards the exact Pareto frontier, thus exploring the design space in a fast manner without compromising the quality of design solutions. By adopting the proposed approach, we manage to automatically build customized coprocessor architectures through combined exploration of both compiler- and architectural design parameters, and to accelerate the exploration by eliminating the iterative resorting to the costly architectural synthesis procedures. We show that RSM-based exploration will be proved to be an efficient approach, even including

978-3-9815370-0-0/DATE13/©2013 EDAA

the time required for training the RSM. Experimental results show that the HLS solution space can be accurately modeled by using RSM techniques.

The rest of the paper is organized as follows. Section II presents related work. Section III introduces the evaluated RSM techniques and describes the proposed methodology. Section IV provides experimental evaluation of the proposed methodology, while Section V concludes the paper.

II. Related Work

Coprocessor synthesis is strongly connected with the problems addressed by high level synthesis. During the evolution of the HLS, we can recognize three main directions. The first direction focuses on the development of efficient scheduling algorithms [13], [5] for generating optimized hardware from a behavioral description. This approach links with the historical roots of HLS, since its main focus is to provide algorithms to built efficient HLS tools. The second direction is mainly driven by the advances in the fields of micro-architecture and compiler technology. Specifically, it is focused on exploring architectural optimizations, i.e. operation chaining [14], clock selection [15] etc. for performance improvement, or exploiting code transformations for datapath optimization [9], [16], [17]. The third direction is the most recent one and is guided by the increased maturity and the availability of the modern industrial and academic HLS tools, i.e. CatapultC [3], AutoESL [18], CyberWorkBench [4], [19], Legup [20], GAUT (Chap. 9 in [1]), SPARK [9]. It proposes the usage of the HLS engine as a "black box" focusing on the tuning of the design parameters and targeting mainly to multi-objective optimization for deriving Pareto optimal trade-offs of conflicting design metrics i.e. delay, area, power. Within this context, Shafer and Wakabayashi [6] proposed a combination of parameter clustering along with an adaptive simulating annealer to accelerate exploration's runtime.

Response surface meta-models have recently gained a lot of attention in micro-architecture community as efficient performance prediction tools of parameterized microprocessor architectures [21]–[23]. The incorporation of predictive RSMs within DSE strategies targeting platform based design, has been also investigated in [24], [25]. In our paper, we differentiate from the aforementioned approaches, since we are applying Response Surface Methods to model and predict the behavior of high-level synthesis optimization algorithms rather than a specific processor parameters. Recently, Zuluaga et al [26] proposed a predictive design methodology for the resource sharing synthesis problem. However, the efficiency of both the exploration methodology (in terms of runtime) and the actual coprocessor architecture (in terms of area complexity, execution latency, power etc.) are highly affected from synthesis steps prior to resource sharing, i.e. operation scheduling, resource binding and register allocation. This paper is rather complementary with the work in [26], since we will show that with the proposed meta-model assisted methodology the aforementioned synthesis steps can be accurately predicted and furthermore the generated models can be used within the actual optimization procedure.

III. Meta-Model Assisted Coprocessor Synthesis

The key idea of this paper is to investigate the usage of analytical meta-models to predict the behavior of a synthesis engine. 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

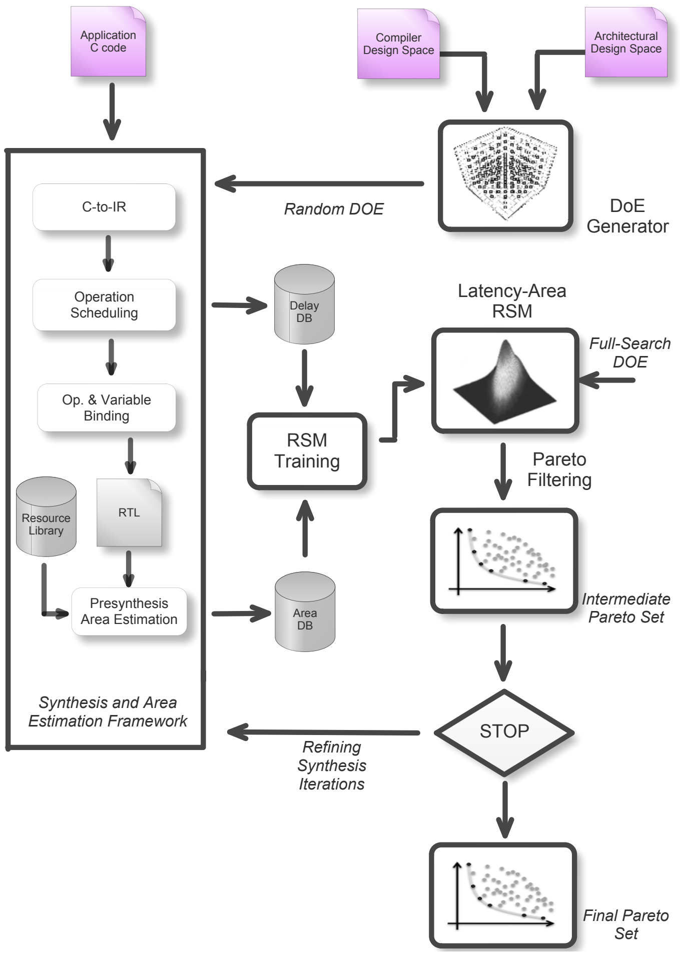


Fig. 1. The meta-model assisted coprocessor synthesis framework

$\boldsymbol{D}$, the targeted problem can be formulated as the following multi-objective optimization problem:

$$\min_{x\in\boldsymbol{D}} \begin{bmatrix} Delay(\mathbf{x}) \\ Area(\mathbf{x}) \end{bmatrix} \in R^2 \tag{1}$$

subject to:

$$\begin{bmatrix} Delay(\mathbf{x}) \\ Area(\mathbf{x}) \end{bmatrix} \leq \begin{bmatrix} Max\_Delay \\ Max\_Area \end{bmatrix} \tag{2}$$

The optimization goal is to find those configuration vectors, $\mathbf{x}$, that are mapped to Pareto (non-dominant) designs in the solution space. To address the aforementioned problem, we exploit Response Surface Modeling (RSM) techniques to capture in an analytical manner the relationships between several design parameters and one or more response variables. A typical RSM-based flow involves a *training phase*, in which known data (or *training set*), generated by Design of Experiments (DoE), are used to tune the RSM configuration, and a *prediction phase* in which the RSM is used to predict the unknown system response.

The proposed RSM-based framework, shown in Figure 1, adopts RESPIR [24] exploration strategy to tackle a new target problem. The left part of the design flow (namely coprocessor synthesis and area estimation framework) includes the typical HLS steps (intermediate representation (IR) generation, operation scheduling and operation/variable binding) for generating the coprocessor RTL description. The RTL description together with a gate-level pre-characterized resource library are used as inputs for estimating (at the pre-synthesis level) the area complexity of the architectural solution. The innovative feature of the proposed framework consists of extending the typical coprocessor synthesis flow with a set of new modules, i.e. the DoE generator, the RSM training function, the trained RSM model, Pareto analysis functions. These modules implement the proposed meta-model assisted exploration framework, which is composed of the following steps:

1) Setup an initial random design of experiments (DoE). The selected design points are synthesized through the coprocessor synthesis and area estimation framework to extract execution latency and area complexity of each point.
2) Train the selected RSM by using the current set of synthesized design points;
3) Compute the *intermediate* Pareto set by using a full-search DoE on the RSM trained in the step 2.
4) For all the points belonging to the intermediate Pareto set and not yet synthesized, proceed with synthesis through the coprocessor synthesis tool flow. This enables new promising points to be part of current set of synthesized design points.
5) If the new set of synthesized design points has at least one new point dominating the previous set, restart from step 2. Otherwise, the final Pareto set is derived from the latest set of synthesized points.

A. Pre-synthesis Area Estimation

In this section, we briefly introduce the targeted coprocessor architectural template (Figure 2) and we present the adopted pre-synthesis area model used in the proposed methodology to avoid the costly gate-level synthesis. We consider a realistic datapath model that enables more accurate evaluation of the costs, associated with each examined solution, unlike the simplified datapath models (number of Alu and Mul units) used in several previous exploration frameworks for coprocessor synthesis [5], [13], [15], [17].

The target architectural template (Figure 2) resembles a typical structure found in hardware accelerators and it is able to model architectures with various degrees of horizontal and/or vertical parallelism (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. The total area complexity ($\mathbf{A}$) for the adopted architectural template is described by the following model:

$$\mathbf{A} = \mathbf{A}_{DP} + \mathbf{A}_{MemIF} + \mathbf{A}_{Regs} + \mathbf{A}_{Steer} + \mathbf{A}_{Ctrl}$$

where the components are the following:

- Datapath ($\mathbf{A}_{DP}$) and Memory Interface ($\mathbf{A}_{MemIF}$) Area:

$$\mathbf{A}_{DP} = \#Alu \times \mathbf{A}_{Alu} + \#Mul \times \mathbf{A}_{Mul}$$

$$\mathbf{A}_{MemIF} = \#LD/ST \times \mathbf{A}_{LD/ST} + \#MemPort \times \mathbf{A}_{MemPort}$$

- Register Bank Area, $\mathbf{A}_{Regs}$:

$$\mathbf{A}_{Regs} = \#Regs \times \mathbf{A}_{Reg}^{Nbit}$$

- Steering Logic Area, $\mathbf{A}_{Steer}$:

$$\mathbf{A}_{Steer} = \sum_{i=0}^{\#Mux} \mathbf{A}_{Mux_i}$$

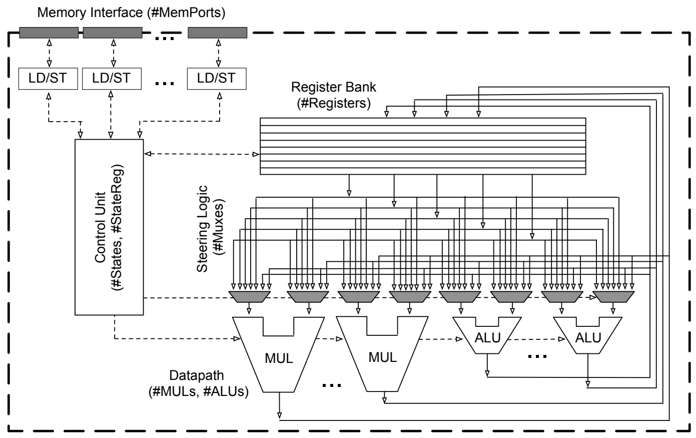


Fig. 2. Target architectural template.

$$\mathbf{A}_{Muxi} = (\#Operands_{Mux_i} - 1) \times \mathbf{A}_{Mux2to1}$$

- Controller Area, $\mathbf{A}_{Ctrl}$:

$$\mathbf{A}_{Ctrl} = \mathbf{A}_{States} + \mathbf{A}_{Mux}^{Sel.Bits} + \mathbf{A}_{Alu}^{Sel.Bits}$$

$$\mathbf{A}_{States} = (\lceil log_2(\#States_{FSM}) \rceil) \times \mathbf{A}_{Reg}^{1bit}$$

$$\mathbf{A}_{Mux}^{Sel.Bits} = (\#Mux \times (\#Operands_{PerMux} - 1)) \times \mathbf{A}_{Reg}^{1bit}$$

$$\mathbf{A}_{Alu}^{Sel.Bits} = (\#Alu \times (\lceil log_2 \#Operation_{Alu} \rceil) \times \mathbf{A}_{Reg}^{1bit}$$

In the previous model, $\mathbf{A}_{States}$ is the area of FSM registers that encode the state space, $\mathbf{A}_{Mux}^{Sel.Bits}$ and $\mathbf{A}_{Alu}^{Sel.Bits}$ are the area of FSM registers for the Mux selection signals and for the ALUs operation selection, while obviously $\mathbf{A}_{Alu}$, $\mathbf{A}_{Mul}$, $\mathbf{A}_{LD/ST}$, $\mathbf{A}_{MemPort}$, $\mathbf{A}_{Reg}^{Nbit}$ and $\mathbf{A}_{Mux2to1}$ are respectively the area for a single ALU, Multiplier, Load/Store unit, Memory Port, N-bit Register and 2 to 1 Multiplexer.

From the previous equations, $\mathbf{A}_{DP}$ and the $\mathbf{A}_{MemIF}$ can be directly estimated from the configuration vector that drives the DSE. However, this is not the case for the other area coefficients ($\mathbf{A}_{Regs}$, $\mathbf{A}_{Steer}$ and $\mathbf{A}_{Ctrl}$) since their area complexity depends on the decision made by the HLS engine. Specifically, the aforementioned coefficients include some terms determined only after (i) operation scheduling, i.e. $\#States_{FSM}$, (ii) register allocation, i.e. $\#Regs$, (iii) variable-to-register binding, i.e. $\#Operands_{Mux_i}$. Due to their strong dependence with the core synthesis optimization algorithms, these terms form the area coefficients that RSM techniques are required to predict their impact on area complexity.

The presented area model is based on the measurements of the following primitive components, (i) $\mathbf{A}_{Alu}$, (ii) $\mathbf{A}_{Mul}$, (iii) $\mathbf{A}_{LD/ST}$, (iv) $\mathbf{A}_{MemPort}$, (v) $\mathbf{A}_{Reg}^{Nbit}$, and (vi) $\mathbf{A}_{Mux2to1}$. For these components we derived area estimation through post-synthesis characterization by using Synopsys Design Compiler [10] and TSMC 0.13 um standard cell library [27]. To validate the area model, we synthesize the HDL descriptions that derived as output of the SPARK HLS tool for several design configurations of our benchmark suite. Validation data in terms of gate-level synthesized area with respect to the pre-synthesis area estimation for the aforementioned designs show that the minimum, average and maximum errors are 0.3%, 11% and 25.6%, respectively.

B. Accuracy analysis for RSM selection

This section describes the procedure for selecting the most suitable analytical description for the (i) targeted design space, (ii) synthesis engine and (iii) benchmark kernels.

The set of parameters composing the design space are of great importance regarding the quality of the approximated Pareto designs. For example, recent studies [7], [8], [11], [12] proved that the combination of compiler- with architectural-level design parameters, enables exploration in a more global manner and leads to higher quality design solutions. We extend the compiler-architecture parameter space targeted in [7], [8], [12], by introducing architectural design decisions regarding the clock frequency following [15] and the memory interface configuration of the coprocessor architecture. The targeted design space (in terms of both compiler and architecture parameters and their ranges) is shown in Table I. The SPARK HLS tool [9] has been used within our DSE framework. Among the HLS tools that are now available to the community, such as Legup [20], we selected SPARK due to its advanced core HLS algorithms and its configurability on both compiler- and architectural level parameters. However, the proposed methodology is general enough and can be easily retargeted to other HLS tools. Finally, we formed a representative set of benchmarks found into real-life DSP and multimedia applications. The benchmark suite consists of 7 computationally intensive kernels of various sizes and complexities: (i) the 1D DCT, (ii) the YUV to RGBA filter, (iii) the MESA 44 Matrix Multiplication algorithm, (iv) the 2D DCT kernel of JPEG, (v) the 2D Inverse DCT kernel of MPEG-2, (vi) the Gauss Blur image transformation from Cavity Detector algorithms, (vii) the 8×8 Sobel edge detection image filter. Although the benchmarks would expose a variety of design spaces, we target the unified design space shown in Table I.

The RSM selection is based on the minimization of the accuracy error and maximization of convergence behavior. Specifically, we considered five RSM models (three regression-based and two interpolation-based): (i) Linear Regression, (ii) Spline-based Regression [28], (iii) Artificial Neural Networks (ANNs) [29], (iv) Shepard-based Interpolation and (v) Radial Basis Functions [30]. The RSMs have been trained considering the aforementioned setup. After training, we validated the average normalized error (on both area and delay metrics) computed by the selected set of RSMs over the entire design space by varying the number of random configurations used as a training-set (known points) from 200 to 1800. In this context, to further improve the prediction accuracy, we introduced a Box-Cox power transform on each sample of the observed data [22] to reduce the data variation and improve the correlation between parameters and response functions. We examined three power parameters $\xi \in \{1, 0.5, 0\}$. For the cases under analysis, we found that the *log* power parameter ($\xi = 0$) 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.

Detailed analysis showed that the accuracy of linear and spline-based regression models remains almost stable and 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, while between 21% and 11% for the spline-based regression. Concerning linear regression, we considered also a second order model without interaction that presented an average prediction error that is 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, from 20-15% to 11-8.5% when considering the ANN and from 16-13% to 9-8% for the radial basis functions (depending on the adopted Box-Cox transformation).

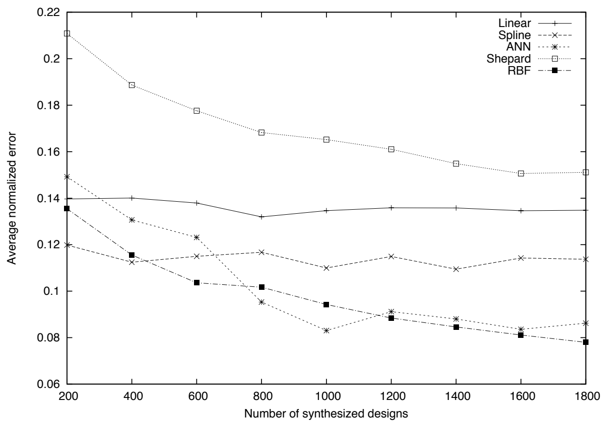


Fig. 3. Validation results for the adopted RSM methodologies,with a training data set corresponding to less than 3.4% of the solution space.

TABLE I
UNIFIED COMPILER/ARCHITECTURE DESIGN SPACE.

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

Figure 3 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 and ANN present the best estimation accuracy when increasing the size of the training data. However, radial basis functions also present a very consistent decreasing 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 within a meta-model enhanced exploration strategy for the target problem. For this reason, we select the RBF to be used as the estimation model in the proposed meta-model assisted methodology to carry out the experimental campaign.

IV. EXPERIMENTAL RESULTS

In this section, we evaluate the efficiency of the proposed meta-model assisted exploration strategy in comparison to state-of-art multi-objective optimizers. In addition, we further prove the effectiveness of the proposed approach by applying it to a more complex and realistic design problem that considers the synthesis of MPEG-2 coprocessors.

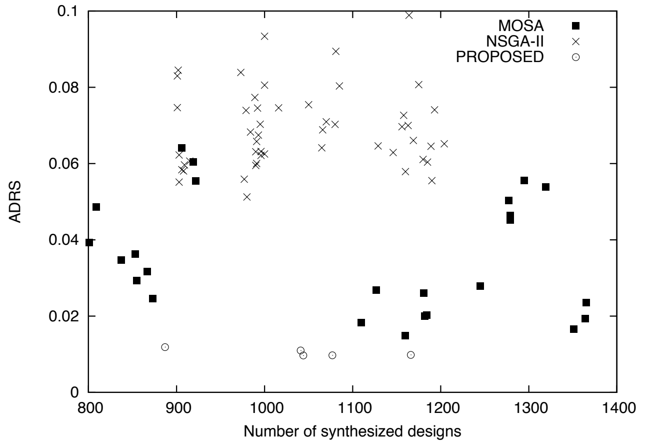


Fig. 4. Application of the MOSA, NSGA-II and Meta-Model assisted design space exploration strategies to the targeted benchmark suite: Accuracy vs. Efficiency.

A. Comparison Results

We have ran an experimental campaign by plugging in the RBF models selected from the validation results presented in the previous section, combined with a Random DoE strategy for selecting the initial set of points[1]. The Random DoE has been ran 5 times to account for the associated unpredictability of the designs. In order to evaluate the efficiency of the proposed RSM-based strategy, we compared with two state-of-art *non RSM-based* meta-heuristics for multi-objective optimization: (i) the Multi-Objective Simulating Annealing [31] (MOSA) and the (ii) NSGA-II multi-objective genetic algorithm [32]. For a fair comparison with state-of-art optimizers, we run the explorations by varying the MOSA and NSGA-II parameter configurations (such as, number of generations, population size and annealing coefficient), to avoid the possibility of a biased behavior.

We present results in an aggregate manner for all benchmarks. For the design space denoted in Table I, we compare the three exploration strategies in terms of (i) optimality of results through the Average Distance from Reference Set (ADRS) [33] metric (the lower the better), with respect to the exact Pareto curve and (ii) number of synthesized solutions that are proportionally correlated with the exploration's runtime efficiency. In case of proposed exploration strategy, synthesized solutions refer to the number of solutions that needed during runtime for the initial Design of Experiments and the refinement training of the RBF model, while in case of MOSA and NSGA-II, they refer to the number of examined solutions by the algorithms. Figure 4 shows the ADRS of the approximate Pareto front with respect to the exact Pareto front, by varying the number of synthesized designs for each exploration strategy (in the range between 800 and 1400, corresponding to a range of 1.5% - 2.7% of the examined design space). The efficiency of each exploration strategy is layered in different ranging zones. It is clear that the zone of the proposed methodology dominates almost completely both MOSA and NSGA-II variants. For the same or smaller number of synthesized configurations, the proposed RSM-based methodology delivers design solutions that are closer to the exact Pareto set with respect to the other DSE strategies. Specifically, it approximates the exact Pareto frontier with an average error (ADRS) close to 1%. The ADRS values for MOSA range between 2%-6%, while for the NSGA-II between 5%-10%, respectively. We notice that all the 5 runs of the proposed strategy belong to the Pareto frontier of the ADRS vs. Number of synthesized designs solutions space, showing also its robustness.

B. 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. The goal of this case study is the minimization of the average time needed for elaborating each pixel in order to maximize the application throughput. We focus our attention on the design of two HW coprocessors used by the algorithm: 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 by the minimization of the sum of the average elaboration time for the two accelerators. Moreover, the minimization problem has been constrained with an area value that should be less than 230000 um^2 considering both coprocessors. To solve this problem, we applied the proposed methodology to the two accelerators in order to find the two Pareto curves. Then, we combined the results of the two explorations into a single Pareto curve representing the solution space in the area and average latency per pixel space. The design space used for each accelerator is the same presented in Table I.

More in detail, Figure 5 shows the approximate Pareto curves for the two accelerators, found by using the proposed methodology, compared to the exact Pareto curves [2]. In particular, Figures 5(a) and 5(b) present the results for the 2D-IDCT and YUV2RGBA obtained respectively after 154 and 263 evaluations. Despite of the small number of evaluations, it can be noticed that in both cases the proposed methodology is very close to the exact Pareto curve.

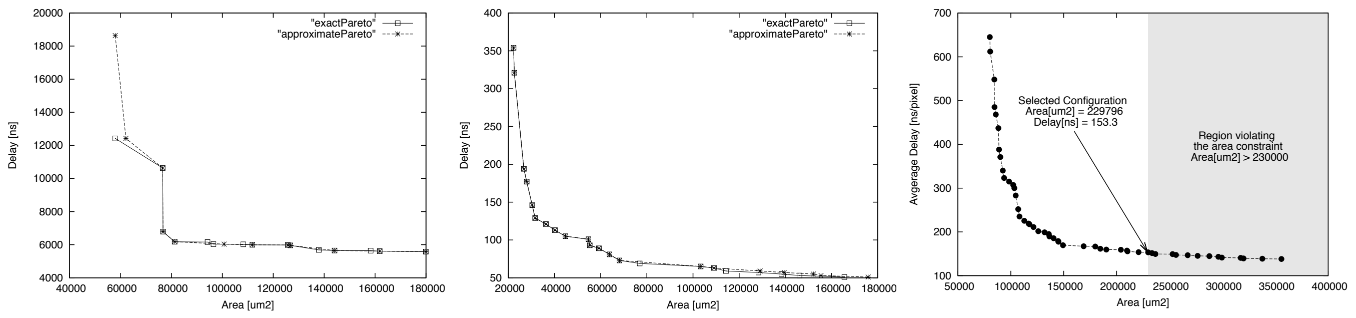
Once we found the two approximate Pareto curves, we combined the two sets of solutions in terms of delay per pixel (the delay values for the 2D-IDCT should be divided by 64 since the accelerator works on a 8x8 matrix of pixels). After this step, we found a new set of points that have been Pareto filtered considering the area and average delay per pixel objectives (see Figure 5(c)). Finally, among all the Pareto points we select the combined solution that reduces the average delay per pixel while respecting the area constraint.

V. CONCLUSION

In this paper, a meta-model assisted coprocessor customization framework has been introduced leveraging Response Surface Models to analytically represent the behavior of a coprocessor synthesis engine. The approach is based on the construction of analytical delay and area models (RSMs) for predicting the quality of the design points without resorting to costly architectural synthesis. The proposed meta-model

[1]The initial DoE is made up with 50 points, which roughly corresponds to the sum of all the levels of the parameters of the design space.

[2]For validation purposes, we have run exhaustive search, even if the size of the targeted design space is approximately 52K configurations.



(a) 2D-IDCT: Exact vs. Approximate Pareto (b) YUB2RGB: Exact vs. Approximate Pareto (c) Combined Pareto curve for the selected MPEG-2 decoder accelerators.

Fig. 5. Pareto curves for the MPEG-2 use case.

customization strategy iteratively refines the accuracy of the approximate Pareto curve towards the exact Pareto frontier.

To evaluate the efficiency of the proposed RSM-based strategy, we compare it with respect to two state-of-art non RSM-based meta-heuristics for multi-objective optimization: Multi-Objective Simulating Annealing (MOSA) and NSGA-II multi-objective genetic algorithm. For the given set of benchmarks, the proposed methodology outperforms both MOSA and NSGA-II improving the solutions quality by 3% and 7% respectively for the same number of evaluated designs.

References

[1] P. Coussy and A. Morawiec. *High-Level Synthesis: From Algorithm to Digital Circuit*. Springer, Berlin, Germany, 2008.

[2] A. Gerstlauer, C. Haubelt, A.D. Pimentel, T.P. Stefanov, D.D. Gajski, and J. Teich. Electronic system-level synthesis methodologies. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(10):1517 –1530, oct. 2009.

[3] Catapult, www.mentor.com/esl/catapult.

[4] CyberWorkBench, www.nec.com/global/prod/cwb/.

[5] Gang Wang, Wenrui Gong, Brian DeRenzi, and Ryan Kastner. Exploring Time/Resource Trade-offs by Solving Dual Scheduling Problems with the Ant Colony Optimization. *ACM Trans. Design Autom. Electr. Syst.*, 12(4), 2007.

[6] Benjamin Carrión Schäfer and Kazutoshi Wakabayashi. Design space exploration acceleration through operation clustering. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 29(1):153–157, 2010.

[7] Sotirios Xydis, Christos Skouroumounis, Kiamal Pekmestzi, Dimitrios Soudris, and George Economakos. Designing Efficient DSP Datapaths Through Compiler-in-the-Loop Exploration Methodology. In *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2598–2601, 2010.

[8] Sotirios Xydis, Christos Skouroumounis, Kiamal Pekmestzi, Dimitrios Soudris, and George Economakos. Efficient High Level Synthesis Exploration Methodology Combining Exhaustive and Gradient-Based Pruned Searching. In *Proc. of the IEEE Annual Symposium on VLSI (ISVLSI)*, pages 104–109, 2010.

[9] Sumit Gupta, Nikil Dutt, Rajesh Gupta, and Alex Nicolau. Coordinated Parallelizing Compiler Optimizations and High-Level Synthesis. *ACM Trans. Des. Autom. Electron. Syst*, 9:2004, 2002.

[10] Synopsys Inc. www.synopsys.com/products/. 2009.

[11] Alastair Colin Murray, Richard Vincent Bennett, Björn Franke, and Nigel P. Topham. Code Transformation and Instruction Set Extension. *ACM Trans. Embedded Comput. Syst.*, 8(4), 2009.

[12] Paolo Bonzini and Laura Pozzi. Code transformation strategies for extensible embedded processors. In *CASES '06: Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pages 242–252, New York, NY, USA, 2006. ACM.

[13] Pierre G. Paulin and John P. Knight. Force-directed scheduling for the behavioral synthesis of asics. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 8(6):661–679, 1989.

[14] P. Marwedel, B. Landwehr, and R. Domer. Built-in Chaining: Introducing Complex Components into Architectural Synthesis. In *Proc. of the ASP-DAC*, pages 599–605, 1997.

[15] Stephen A. Blythe and Robert A. Walker. Efficient optimal design space characterization methodologies. *ACM Trans. Des. Autom. Electron. Syst.*, 5(3):322–336, 2000.

[16] Srikanth Kurra, Neeraj Kumar Singh, and Preeti Ranjan Panda. The impact of Loop Unrolling on Controller Delay in High Level Synthesis. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 391–396, 2007.

[17] Joachim Gerlach and Wolfgang Rosenstiel. A Methodology and Tool for Automated Transformational High-Level Design Space Exploration. In *ICCD*, pages 545–548, 2000.

[18] AutoESL, www.autoesl.com.

[19] ROCCC 2.0, http://www.jacquardcomputing.com/.

[20] Legup HLS, http://legup.eecg.utoronto.ca/.

[21] P. J. Joseph, Kapil Vaswani, and Matthew J. Thazhuthaveetil. A predictive performance model for superscalar processors. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 161–170, Washington, DC, USA, 2006. IEEE Computer Society.

[22] P.J. Joseph, Kapil Vaswani, and M.J. Thazhuthaveetil. Construction and use of linear regression models for processor performance analysis. In *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, pages 99 – 108, feb. 2006.

[23] Tejas S. Karkhanis and James E. Smith. A first-order superscalar processor model. In *Proceedings of the 31st annual international symposium on Computer architecture*, ISCA '04, pages 338–, Washington, DC, USA, 2004. IEEE Computer Society.

[24] Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. Respir: a response surface-based pareto iterative refinement for application-specific design space exploration. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 28:1816–1829, December 2009.

[25] E. İpek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz. Efficiently exploring architectural design spaces via predictive modeling. *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, 40(5):195–206, 2006.

[26] Marcela Zuluaga, Edwin Bonilla, and Nigel Topham. Predicting Best Design Trade-offs: A Case Study in Processor Customization. In *DATE '12: Proceedings of the conference on Design, automation and test in Europe*, 2012.

[27] Artisan Components. Tsmc 0.13 library databook.

[28] Benjamin C. Lee and David M. Brooks. Spatial sampling and regression strategies. *Micro, IEEE*, 27(3):74 –93, may-june 2007.

[29] S. E. Fahlman, D. Baker, and J. Boyan. *The cascade 2 learning architecture, Technical Report CMU-CS-TR-96-184*. Carnegie Mellon University, 1996.

[30] M. J. D. Powell. The theory of radial basis functions. In *Advances in Numerical Analysis II: Wavelets, Subdivision, and Radial Basis Functions, W. Light (ed)*, pages 105–210. University Press, 1992.

[31] K.I. Smith, R.M. Everson, J.E. Fieldsend, C. Murphy, and R. Misra. Dominance-based multiobjective simulated annealing. *Evolutionary Computation, IEEE Transactions on*, 12(3):323 –342, june 2008.

[32] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182 –197, apr 2002.

[33] Tatsuya Okabe, Yaochu Jin, and Bernhard Sendhoff. A critical survey of performance indices for multi-objective optimisation. In *Proc. of 2003 Congress on Evolutionary Computation*, pages 878–885. IEEE Press, 2003.