Combining Evolutionary Algorithms and Deep Learning for Hardware/Software Interface Optimization

Lorenzo Servadei *
Johannes Kepler University Linz
Infineon Technologies AG
Munich, Germany
Lorenzo.Servadei@infineon.com

Edoardo Mosca *
Technical University of Munich
Infineon Technologies AG
Munich, Germany
Edoardo.Mosca@infineon.com

Michael Werner
Technical University of Munich
Infineon Technologies AG
Munich, Germany
Michael.Werner@infineon.com

Volkan Esen
Infineon Technologies AG
Munich, Germany
Volkan.Esen@infineon.com

Robert Wille
Johannes Kepler University Linz
Linz, Austria
Robert.Wille@jku.at

Wolfgang Ecker
Technical University of Munich
Infineon Technologies AG
Munich, Germany
Wolfgang.Ecker@infineon.com

***Abstract*—With the advancement of Internet of Things, the cost of System-on-Chips (in terms of area, performance, etc.) becomes increasingly relevant for realizing affordable as well as performant devices. Although System-on-Chips are very diverse with respect to specifications and requirements, some components are ubiquitous. One of them is the Hardware/Software Interface, which serves for controlling communication and interconnected functionalities between Hardware and Software. Motivated by their common use, the implementation of optimized interfaces towards certain costs (in terms of area, performance, etc.) becomes a central problem in the design of embedded systems. In this work we introduce a novel optimization method for minimizing the cost of Hardware/Software Interfaces using Convolutional Neural Networks coupled with Evolutionary Algorithms.**

***Index Terms*—Hardware/Software Interface Optimization, Machine Learning, Deep Learning, Evolutionary Algorithms**

I. Introduction and Motivation

*Hardware/Software Interface*s (HSIs) are particularly relevant for the *System-on-Chips* (SoCs) design process due to their role of regulating Hardware/Software interactions. These are mostly defined by the use of bitfields (data access), which regulate the intercommunication of Hardware and Software for peripheral devices of the SoC. For designing an HSI instance, the engineer is provided with several options concerning how to realize a system. In particular, the displacement of the bitfields offers many degrees of freedom in the HSI realization. The choice of a specific bitfields configuration results in a different HW/SW logic inside the HSI, which in turn changes the costs of different HSIs design configurations (concerning Hardware usage and Software metrics).

To determine an optimal bitfields position, methods for HSI cost optimization are frequently applied in industrial practice. However, currently used methods are manual, do not scale to the increasing complexity of modern SoCs, and do not take into account the cost of an HSI instance. In fact, they are mostly based on design experience and fixed-rules. In this work, we address the problem of finding an optimal position for a given set of bitfields, in order to minimize the cost of the corresponding HSI instance. To be more specific, we are interested in minimizing

- the area, denoted by *Logic Units Tables* (LUTs) and *Slice Registers* (SRs) which are needed to realize the HSI on an FPGA board
- the size of the compiled binary file *SW code size* (SS) as well as the number of cycles needed by the CPU to execute the SW program (SCs).

From now on, with the term *design cost*, we refer to the four objectives LUTs, SRs, SS, and SCs. With this in mind, our task can then be formalized as follows: given an HSI x containing L bitfields $\{b_i\}_{i=1}^{L}$, we are interested in optimizing (i.e. minimizing) the design cost, which highly depends on the spatial allocation of the b_i on x [6]. In other words, x can be seen as a board on which the b_is are placed as objects and the optimization aims to find a placement for the b_i on x such that the design cost $C(x)$ is minimized.

In [5], a similar problem (3D bin packing) has been successfully approached through *Machine Learning* (ML) optimization methods. The success of their approach on a similar allocation task is a strong motivation towards data-driven methods for nearing this problem class. Consequently, there is more room than just for greedy methods.

In the same spirit, we borrow and re-adapt solutions based on ML, in particular, *Deep Learning* (DL) and *Evolutionary Algorithms* (EA). Based on these, an approach is proposed which outperforms existing methods for cost optimization. Experimental evaluations within an industrial context show that the proposed method can optimize on average $9,83\%$

* These Authors contributed equally to this paper

on the LUTs objective, $7,82\%$ on the SS and $8,85\%$ on the SCs values w.r.t. existing methods. In addition, we argue that our approach outperforms existing methods for decrease in computational time and manual effort.

As for the structure of this work, in section II we briefly review related works in the fields concerning our proposed solution. In section III we describe the proposed method for combining EA and DL in order to optimize HSIs. The implementation details and experimental results are then presented in section IV. Finally, in section V, we enlist the outcomes and introduce ideas for future work. Our contribution could be summarized as follows:

- We propose a novel method that combines DL and EA for the purpose of HSI optimization towards HW and SW metrics. To the best of our knowledge, this is the first non-greedy/manual method for HSI optimization.
- We re-adapt novel Neural Networks architectures popular in Computer Vision towards the estimation of HW and SW metrics.
- We apply the proposed method in an industrial context, providing therefore experimentation details on real-world data.

II. Related Work

In this section, we briefly summarize previous approaches for the design process and the related flow in our industrial environment. Successively, we refer to the existing work in the fields that relate to HSI optimization and the proposed method.

A. Hardware Software Interfaces Design and Optimization

The MetaRTL framework, originally proposed in [8], provides an environment for Hardware design generation (in this specific case, HSIs). MetaRTL allows to define abstractions and properties for each Hardware component and generates the design based on a chosen configuration. The automation and abstraction of the design process leads to an increase in productivity and to a rapid workflow. For this reason, such model-based generation approach (also thoroughly described in [4], [17]), is adopted in our industrial environment. Thanks to this, the designer instantiates a particular configuration of the component through an instance of the *meta-model* following the design requirements. In particular, the process of specification for an HSI meta-model, shown in Fig. 1, is inspired by [6]. This configuration would then trigger the generation of RTL as well as Software code. Finally, the cost concerning Hardware and Software metrics is obtained from the synthesis/compilation of the previously generated code. The obtained cost would then guide the designer towards a suitable implementation of Hardware together with Software on an FPGA board. This same procedure applies to a set of Hardware components, including HSIs. A complete description of this meta-model is available in an online appendix [7].

Given the HSI meta-model, appropriate configurations have to be chosen such that the intended behavior is realized (e.g. enabling proper access to peripheral devices) while satisfying specified cost constraints. This motivated existing work, such as [11], [14], [16], [21], to find faster design cost estimation methods that support the designer decisions. Furthermore, these recent methods of cost estimation have been paving the road to an ML-aided design optimization. In fact, ML-driven optimization methods tasks have been proposed in different fields such as placement and routing [13] and logic optimization [10]. However, to the best of our knowledge, this is the first work that optimizes HSI configurations with an ML-based method.

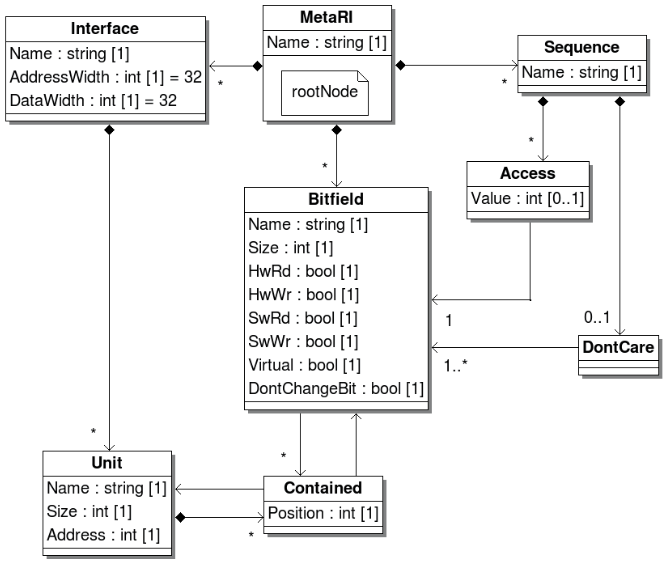


Fig. 1. Example of HSI meta-model

B. Deep Learning for Computer Vision

Computer Vision (CV, [1]) is a discipline which focuses on the analysis and interpretation of images. CV is able to perform several tasks (e.g. object detection, segmentation, classification) by an automated processing *features* contained in the images. Before the advent of Deep Learning (DL, [9]), the state-of-the-art algorithms would use manually retrieved features for applying processing algorithms [15].

Current DL algorithms can instead utilize pixel-based representations as an input of the ML algorithms [2], [9]. This happens through the implementation of *Neural Networks* (NNs, [2], [9]) (i.e. ML architectures which are motivated by the functioning of biological neurons), which can exploit efficiently raw features in form of pixel data, reaching the state-of-the-art in several CV tasks. To be able to elaborate efficiently raw features and utilize many consecutive processing layers, some specific architectures encourage features sharing and gradient propagation. One of the most famous NN with these characteristics, which is of particular relevance for this work, is DenseNet [12].

C. Evolutionary Algorithms for Optimization

EA optimization methods are inspired by biological selection and evolution. They became recently popular and have been applied in several fields, such as NASA optimal Antenna

Design and Optimal Floor-Plan Design, as described in [19]. One of their main characteristics, besides the biological link, is that they are a gradient-free method, i.e. they do not require to compute derivatives. This feature distinguishes them from other more complex recent approaches such as Reinforcement Learning [20]. Generally, in EA, a set $G = \{x_i\}_{i=1}^N$ of individuals, i.e. possible solutions for the considered optimization problem, is created. We call this set a *generation*. The elements are evaluated through a *fitness function* $Fit(x)$ that assess how "fit" these individuals are, that is how good they are as a solution for the given optimization problem. The fittest individuals are then picked and used to generate the next generation G' through a mutation process. That means that every new individual $x'_i \in G'$ has been generated by changing and/or combining some of the fittest individuals in G. This process is then repeated several times and aims to select fitter and fitter individuals as generations go by and to get as close as possible to the ideal individual $x^* = \arg\max_x Fit(x)$. In this work, this process is used to mimic the selection of the designers among many possible HSIs configurations, searching for the one that minimizes the design cost.

III. Proposed Method

In this section, we sketch conceptually the proposed method. Our approach could be simply described as an alternation between one *evaluation step* and one *mutation step*. In III-A we describe, as a preliminary, the used data representation. Afterward, in III-B, our DL-based estimation method choice for the evaluation step is explained. Finally, in III-C, the two main steps are both described first singularly and later as parts of the combined flow.

A. Data Representation

In order to efficiently combine DL and EA approaches in an unique flow, we use a data representation that is appropriate for both methods. While the implemented DL method needs to estimate each configuration fast and accurately, the EA has to generate new sets of candidate solutions. For this reason, on the one hand the DL method takes advantage of a data representation in image format. Such notation uses a 2D image for Hardware's cost prediction and a 3D image for Software cost metrics, in a similar fashion to what presented in [18]. On the other hand, the EA component uses an HSI compact notation for performing each mutation step. This notation condenses in one small dictionary the information required to retrieve the 2D and 3D images that are necessary to perform the estimation step. An illustrative example of the two different notations can be seen in Fig. 2.

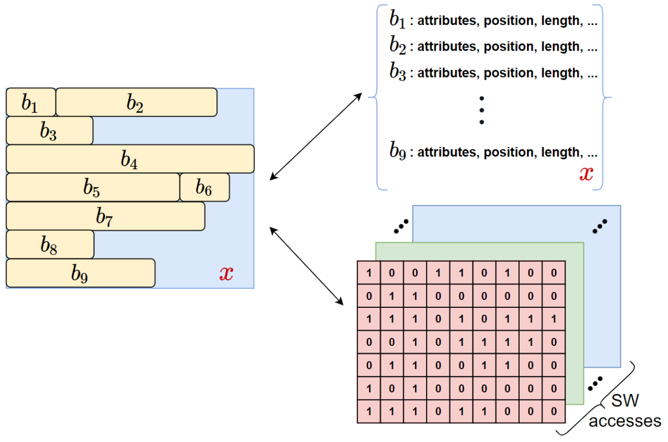


Fig. 2. Two equivalent ways of representing an HSI: Compact Notation (up) and Image Notation (down)

B. Estimating the Configuration Cost with Deep Learning

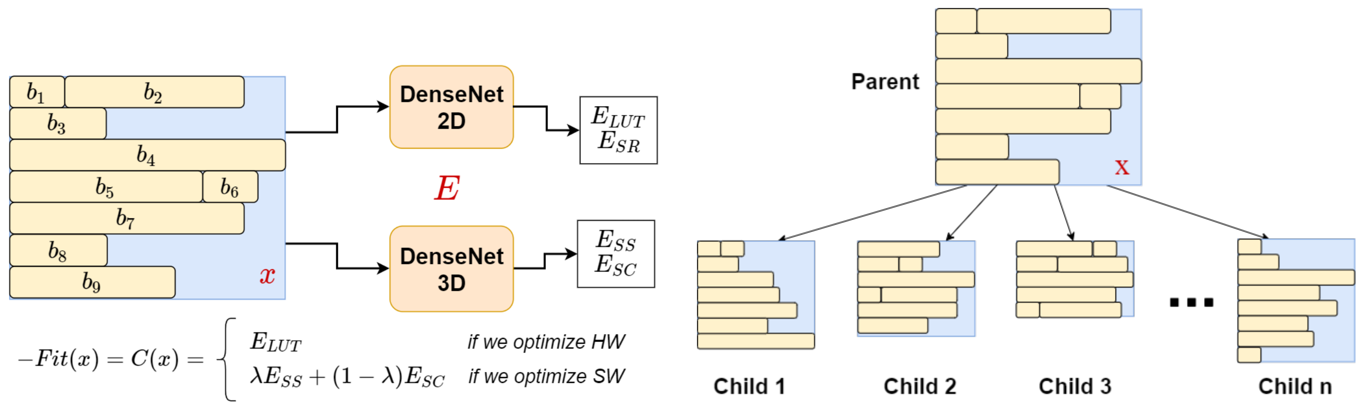
During the optimization process, the algorithm needs to estimate iteratively each different HSI configuration x. This requires a very efficient method E to estimate the cost $C(x)$ independently from which configuration x has been given as input. Tipically, the cost is calculated through a synthesis process (using Xilinx Vivado). Unfortunately, the current synthesis method is time-expensive and, if several iterations are necessary for retrieving the optimal one, the use of the synthesis tool becomes soon infeasible. For this reason, our estimation method E for our evaluation step needs to be some order of magnitude faster in order to be coupled with our mutation step.

Convolutional Neural Networks (CNNs) have been proven to provide very accurate predictions for data structures which present spatial positions [9]. At the same time, in [18], the image format has been proven to be a good representation for predicting HSI's design cost. Motivated by this, we use convolutional layers for estimating the cost $C(x)$ for a certain HSI x structured as 2D and 3D images. This allows to compute $C(x)$ in a very fast and accurate manner. Hence, we adopt them as method E for our evaluation step.

To increase training efficiency and spatial features sharing, we assemble our convolutional layers with the DenseNet architecture [12]. DenseNet extends a CNN with skip-connections that allow an easier backpropagation of the gradient. This additional skip-connections contributed to the big success that this method has in the field of CV. More in detail, we build two architectures (2D and 3D) that are used as estimation method E respectively for Hardware and Software metrics optimization. Both nets possess two convolutional dense blocks and are connected with skip-connections. Together with a faster convergence in training, we consistently report more accurate and robust predictions after the implementation of the additional connections.

C. Optimizing the Cost with Evolutionary Algorithms

In order to use a common terminology in the EA field, from now on, when we say individual x, we refer respectively to an HSI x. As already mentioned in II-C, EAs are gradient-free optimization methods that aim to find the fittest individual $x^* = \arg\max_x Fit(x)$. To do so, they simulate an arbitrary number M of generations G. In our case, for every generation G, we have two steps: evaluation and mutation.


b_1
b_2
b_3
b_4
b_5
b_6
b_7
b_8
b_9
x
DenseNet 2D
E
DenseNet 3D
E_{LUT} E_{SR}
E_{SS} E_{SC}
$-Fit(x) = C(x) = \begin{cases} E_{LUT} & \text{if we optimize HW} \\ \lambda E_{SS} + (1-\lambda) E_{SC} & \text{if we optimize SW} \end{cases}$
Parent
x
Child 1
Child 2
Child 3
Child n

(a) Evaluation step (b) Mutation step

Fig. 3. The two main steps of our method considered singularly. (a) evaluates an HSI configuration depending on what we are trying to optimize, (b) mutates the HSI configuration to generate new candidates.

The evaluation step consists in establishing how fit our current individuals are. To do so, we use an estimation method E (in our case we picked the DL method described in III-B) that takes as input an individual x and gives as output the result for the four design objectives E_{LUT}, E_{SR}, E_{SS} and E_{SC}. We define the fitness of an individual x as the opposite of its cost, i.e $Fit(x) = -C(x)$, where

$$C(x) = \delta E_{LUT} + (1-\delta) E_{SR} \tag{1}$$

in case we are interested in optimizing the area objectives (LUTs and SRs), and

$$C(x) = \lambda E_{SS} + (1-\lambda) E_{SC} \tag{2}$$

in case we are interested in optimizing the SW metrics (i.e. SS and SCs). Here, the hyper-parameters $\delta \in [0, 1]$ and $\lambda \in [0, 1]$ enables the designer to specify which the contribution to the final cost of E_{LUT}, E_{SR}, E_{SS} and E_{SC}.

Although SRs are important for evaluating the design cost, they are independent w.r.t. allocation of the b_i on x and cannot be optimized. For this reason, we remove them from (1), obtaining now

$$-Fit(X) = \begin{cases} E_{LUT}, & \text{when optimizing HW} \\ \lambda E_{SS} + (1-\lambda) E_{SC}, & \text{when optimizing SW} \end{cases} \tag{3}$$

as overall fitness definition.

As a consequence, only the hyper-parameter λ regulates the optimization step, and enables the designer to specify which SW design objective is more relevant for the optimization at the moment. For example, in (3), increasing λ will make E_{SS} more important while decreasing it puts the focus of the optimization on E_{SC}. As default, we propose $\lambda = 1/2$, which can then be changed depending on the designer's needs. With the given notation, maximizing the fitness function is equivalent to minimizing the design cost. A high-level sketch of this can be also seen in Fig. 3(a).

The mutation step creates new individuals by mutating the fittest individuals from the previous generation, i.e. creates a new individual x' by changing the spatial position of some b_i inside a previous HSI x. This is done by choosing how many bitfields of the old configurations x should be randomly moved, we call this ratio $r \in [0, 1]$ *mutation rate*. Specifically, $r = 1$ is equivalent to move all the bitfields b_i in a random way and therefore creating a totally random HSI while $r = 0$ is equivalent to not move any of the b_i and therefore having no mutation, i.e. $x = x'$. Hence, the adopted mutation rate can be seen as an arbitrary hyper-parameter to trade-off exploration and exploitation when generating new possible configurations.

The movement of the selected b_i is completely random, with the constraint of not overlapping other bitfields and not exiting the limits of the HSI size. A simple sketch of this step is shown in Fig. 3(b).

The two steps are combined in a unique pipeline. Before each step, due to the difference in the input's data representation, we perform a mapping from one notation of every HSI x to the other. For example, before we run the evaluation step, we will map our compact notation for x used by the mutation step to the image representation. The image can successively be fed to our DL method for the estimation. Analogously, before the mutation step, we will perform the inverse mapping on every HSI x image picked from the previous generation.

All the described steps are combined together as a cycle, shown in Fig. 4, that is iterated several times until M generations have been processed. Every cycle aims to provide fitter, i.e. more optimized, individuals.

IV. Experimental Results

In this section, we describe some relevant details about data generation, real cost retrieval, method implementation and training. After that, we report the obtained results in our industrial environment and discuss them.

A. Dataset and Setup

For the experimental phase we use two datasets, both generated in the same fashion as [18]. Each sample of the

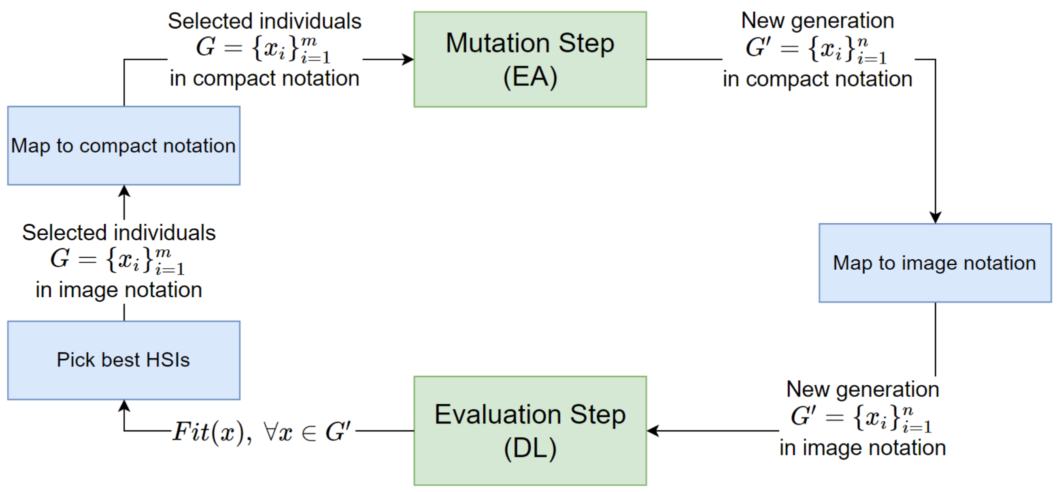


Fig. 4. The two main steps, mutation and evaluation, combined in one unique flow together with intermediate steps.

dataset belongs to a predefined design space (in terms of No. of Bitfields, where No. of Bitflieds $\in [15, 30]$) and contains variations over spatial distribution of bitfields.

We first use a dataset consisting of 4532 samples to train our DL model that is implemented for estimation purposes. On this dataset, we perform a train/validation/test split in the proportions 70/15/15 following the common practice. The difference in accuracy between the train and test set is negligible (i.e. $< 1\%$), therefore we do not report overfitting for our estimation method.

To evaluate our optimization method, we optimize configurations belonging to a much smaller dataset containing 100 samples and we observe the average improvement, i.e. the average decrease in the design cost. Unlike most ML algorithm, this second dataset does not need training/test splits. In fact, our proposed method does not require training for the two steps once we have a pre-trained estimation method.

After generating the design configurations, we retrieve the objectives measurements for each one of the design instances. We first ran a synthesis (through Xilinx Vivado) on the instances to obtain the real cost for the objectives LUTs and SRs of each design. After that, we ran on each one of the HSI a Software program, where the operations of the program are specified in the design configuration. From this, we retrieved the real cost for the SS and SCs objectives.

As implementation environment, we use Python 3.6 with Tensorflow-GPU 1.14.0 and Keras 2.2.4. As for Hardware setup, we use an Nvidia Tesla P100 GPU, an Intel Core i7-8700K CPU and DIMM 16GB DDR4-3000 module of RAM. In the next section, we report timing experiment both with and without GPU acceleration.

B. Results

Intending to evaluate the effectiveness of the proposed method, we implement it in an industrial environment with generated HSI use-cases. To measure the performance, we observe the average improvement on the three design objectives of the optimized configurations that our method outputs and compare it with the results given by the greedy method currently adopted in the industrial setup, namely the *First Fit Search* [3]. The results are expressed in terms of percentage improvement within the variational range of each objective. The quantitative results of this comparison can be seen in Table I. When compared to First Fit Search [3], we report an average improvement of $9,83\%$ when optimizing for the area. When optimizing for the Software metrics with a trade-off value of $\lambda = 1/2$, we report $7,82\%$ for the SS and $8,85\%$ for SCs. In all our experiments we used a mutation rate $r = 0.1$.

TABLE I
OBJECTIVES IMPROVEMENTS EVALUATION

Objective			**Average Improvement**
Name	***Type***	***Variational Range***	**w.r.t to First Fit [3]**
LUTs	Area	[353, 546]	9,83%
SS	Software	[722b, 1645b]	7,82%
SCs	Software	[224, 423]	8,85%

For SW optimization we utilized the default value $\lambda = 1/2$.
For SS, values are expressed in bits (b).

As one would expect, the time effort necessary to run the described method depends on several factors, e.g. the number of steps that the designer wants to perform. Although in our experimentation we cannot determine an ”ideal minimum” number of generations for which the proposed method works, we observe that no optimal candidate is found after 100 generations. Therefore, we propose this number as upper-bound of generations for the optimization process. In Table II we report the necessary time to run the proposed upper-bound number of generations.

We believe that the conducted experimentation shows the advantage of the proposed method when compared to greedy algorithms in terms of optimization performance. The timing experiment results confirm that the usage of a fast estimation method, as described in III-B, allows the proposed method to run in a reasonable amount of time. Furthermore, we

TABLE II
TIMING EVALUATION

Optimization type		**Running time**
Target	***Hardware***	***(seconds)***
Area	with GPU	45,7
Area	only CPU	77,1
Software	with GPU	1298,8
Software	only CPU	2763,4

Results are for 100 generations and are independent w.r.t. λ and r.

also report that the usage of GPU acceleration during the optimization, although not necessary, can further reduce the time needed to obtain optimized HSIs.

V. Conclusions and Future Work

In this work, we aim to solve the problem of optimizing Hardware/Software Interfaces, i.e. finding an optimal allocation for a set of bitfields inside the HSI configuration. To this end, we proposed a novel gradient-free optimization algorithm based on Evolutionary Algorithms and Deep Learning composed of two steps: mutation and evaluation. We define a fitness function that is linked directly to our design cost and therefore to the objectives that are crucial for the design process. Furthermore, we propose the usage of s-o-t-a Deep Learning algorithms to estimate the fitness function. This combination between an Evolutionary Algorithm and a Deep Learning method makes the evaluation step fast. Without this coupling, by using classical methods such as Xilinx Vivado synthesis, the whole pipeline would be extremely time-expensive and therefore not feasible. Experimental evaluations within an industrial context show that our method consistently optimizes an HSI in a fast as well as automatic manner, and outperforms currently used greedy/manual methods in terms of cost optimization.

As for future work, we believe that even more accurate methods for the evaluation step could lead to an improvement of our optimization results. This will be the focus of our future research together with other optimization methods, including gradient-based ones.

VI. Acknowledgement

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

References

[1] Dana Harry Ballard and Christopher M. Brown. *Computer Vision*. Prentice Hall Professional Technical Reference, 1st edition, 1982.

[2] Roberto Cipolla, Sebastiano Battiato, Giovanni Maria Farinella, et al. *Machine Learning for Computer Vision*, volume 5. Springer, 2013.

[3] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[4] Keerthikumara Devarajegowda, Johannes Schreiner, Rainer Findenig, and Wolfgang Ecker. Python based framework for hdsls with an underlying formal semantics: (invited paper). *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1019–1025, 2017.

[5] Lu Duan, Haoyuan Hu, Yu Qian, Yu Gong, Xiaodong Zhang, Jiangwen Wei, and Yinghui Xu. A multi-task selected learning approach for solving 3d flexible bin packing problem. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1386–1394. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[6] Wolfgang Ecker, Wolfgang Mueller, and Rainer Doemer. *Hardware-dependent Software: Principles and Practice*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[7] Wolfgang Ecker and J Schreiner. Metamodeling and code generation in the hardware/software interface domain. In *Handbook of Hardware/Software Codesign*, pages 1051–1091, 11 2017.

[8] Wolfgang Ecker and Johannes Schreiner. Introducing model-of-things (mot) and model-of-design (mod) for simpler and more efficient hardware generators. In *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6. IEEE, 2016.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[10] W. Haaswijk, E. Collins, B. Seguin, M. Soeken, F. Kaplan, S. Süsstrunk, and G. De Micheli. Deep learning for logic optimization algorithms. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, May 2018.

[11] Hsuan Hsiao and Jason H Anderson. Sensei: An area-reduction advisor for fpga high-level synthesis. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 25–30. IEEE, 2018.

[12] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[13] Yu-Hung Huang, Zhiyao Xie, Guan-Qi Fang, Tao-Chun Yu, Haoxing Ren, Shao-Yun Fang, Yiran Chen, and Jiang Hu. Routability-driven macro placement with embedded cnn-based prediction model. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 180–185. IEEE, 2019.

[14] Pingfan Meng, Alric Althoff, Quentin Gautier, and Ryan Kastner. Adaptive threshold non-pareto elimination: Re-thinking machine learning for system level design space exploration on FPGAs. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, pages 918–923. EDA Consortium, 2016.

[15] Mark Nixon and Alberto S. Aguado. *Feature Extraction & Image Processing for Computer Vision, Third Edition*. Academic Press, Inc., Orlando, FL, USA, 3rd edition, 2012.

[16] Adam Powell, Christos Savvas-Bouganis, and Peter Y. K. Cheung. High-level Power and Performance Estimation of FPGA-based Soft Processors and Its Application to Design Space Exploration. *J. Syst. Archit.*, 59(10):1144–1156, November 2013.

[17] Johannes Schreiner and Wolfgang Ecker. Digital hardware design based on metamodels and model transformations. In *VLSI-SoC: System-on-Chip in the Nanoscale Era - Design, Verification and Reliability - 24th IFIP WG 10.5/IEEE International Conference on Very Large Scale Integration, VLSI-SoC 2016, Tallinn, Estonia, September 26-28, 2016, Revised Selected Papers*, pages 83–107, 2016.

[18] L. Servadei, E. Zennaro, K. Devarajegowda, M. Manzinger, W. Ecker, and R. Wille. Accurate cost estimation of memory systems inspired by machine learning for computer vision. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1277–1280, March 2019.

[19] Andrew N. Sloss and Steven Gustafson. 2019 evolutionary algorithms review. *CoRR*, abs/1906.08870, 2019.

[20] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

[21] Elena Zennaro, Lorenzo Servadei, Keerthikumara Devarajegowda, and Wolfgang Ecker. A Machine Learning Approach for Area Prediction of Hardware Designs from Abstract Specifications. In *Proceedings of the 21st Euromicro Conference on Digital System Design*, 2018.