Automatic Approximation of Computer Systems Through Multi-objective Optimization

Mario Barbareschi, Salvatore Barone, Alberto Bosio, Marcello Traiola

Automatic Approximation of Computer Systems through Multi-Objective Optimization

Mario Barbareschi, Salvatore Barone, Alberto Bosio, and Marcello Traiola *

Abstract In this chapter, we address the automatic approximation of computer systems through multi-objective optimization. Firstly, we present our automatic design methodology, i.e., how we model the approximate design space to be automatically explored. The exploration is achieved through multi-objective optimization to find good trade-offs between the system efficiency and accuracy. Then, we show how the methodology is applied to the systematic and application-independent design of generic combinational logic circuits, based on non-trivial local rewriting of and-inverter graphs (AIGs). Finally, to push forward the approximation limits, we showcase the design of approximate hardware accelerators for image-processing and for common machine-learning-based classification models.

Keywords Approximate Computing, Genetic Algorithm, Design Space Exploration, Multi-objective Optimization, Discrete Cosine Transform, Approximate Circuits, Approximate Neural Networks, Approximate Decision Trees

1 Introduction

In this chapter, we foster an application-independent, unified methodology able to automatically explore the impact of different approximation techniques on a given application, while resorting to the Approximate Computing (AxC) design paradigm

Mario Barbareschi, Salvatore Barone
Department of Electrical Engineering and Information Technologies, University of Naples Federico II e-mail: firstname.lastname@unina.it

Alberto Bosio
Univ Lyon, ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, France e-mail: alberto.bosio@ec-lyon.fr

Marcello Traiola
Univ Rennes, CNRS, Inria, IRISA - UMR6074, F-35000 Rennes France e-mail: marcello.traiola@inria.fr

* Authors are listed in alphabetical order

and Multi-objective Optimization Problem (MOP)-based Design-Space Exploration (DSE). We also devote particular relevance to all the phases and steps of the proposed methodology which can be automated. Our methodology is neither tailored to a specific application nor to an approximation technique, it does not require the designer to specify which part(s) of the application should be approximated and how, and it only requires the definition of the acceptable output degradation from the user. Moreover, it addresses the design problem as a MOP, which allows optimizing different figures of metrics, e.g., error and hardware-requirements, at the same time, providing the designer with a set of equally good Pareto-optimal solutions, leaving the designer free to choose the one that, according to his experience, best suits the context or the requirements of the application considered. We also discuss the application of the presented methodology to relevant applications in the scope of the AxC paradigm.

This chapter is organized as follows: Section 2 provides the reader with a brief introduction concerning the AxC design paradigm, including issues and challenges to be addressed to exploit the AxC full potential. Section 3 discusses the automatic design methodology exploiting AxC and MOP-based design methodology, including the main steps of the method that we propose, and how the method helps in addressing the challenges that the AxC paradigm poses to the designer. Section 4 applies the methodology to the design of combinational logic circuits, i.e., those that typically constitute building-blocks for larger, more complex, designs. Section 5 presents the design of hardware accelerators for image-processing, while Section 6 discusses the approximation of two of the most common classification models in the machine-learning domain, namely Deep Neural Networks (DNNs) and Decision Tree based Mutiple Classifier Systems (DT MCSs). These applications are even more challenging, since hardware-accelerators are utterly resource intensive, and reducing the amount of induced error is very critical, because machine-learning systems process a huge amount of data.

2 The Approximate Computing Design Paradigm and its Application

This section provides the reader with a brief introduction concerning the AxC design paradigm, including issues and challenges to be addressed to exploit the AxC full potential.

2.1 Overview

The scientific literature demonstrated that inexact computation can be selectively exploited to enhance computing system performance, defining the AxC paradigm [84]. It is based on the intuitive observation that, while performing exact computation, or maintaining peak-level service performance, require a high number of resources, selective approximation or occasional violation of the specification can provide quite interesting gains in efficiency. In other words, the AxC paradigm exploits the gap between the level of accuracy required by the application or the end-users, and that provided by the computing system – with the former being often far lower than the latter – for achieving diverse optimizations. Thus, this design paradigm has the potential

to benefit a wide range of applications, including data analytic, scientific computing, multimedia and signal processing, machine learning, etc [55].

Anyway, exploiting AxC requires coping with (i) the characterization of parts of the considered software or hardware component, identifying those that are suitable to be approximate; (ii) the approach to introduce actual approximation; (iii) the selection of appropriate error metrics, which generally depend on the particular application; (iv) the actual error-assessment procedure, to guarantee output quality constraints are met [28], and, finally (v) the DSE, to select the best approximate configurations among those generated by a certain approximation technique. As for the first two of the aforementioned issues, pinpointing approximable code or data portions may require the designer to have profound insights into the application. Error-injection is quite common as an approach to find the data or operation that can be approximated with little impact on quality of result [69, 67]. Once portions, or data, to be approximate have been identified, being able to introduce approximation is not a straightforward matter, and may require coping with several technical challenges [19]. Indeed, a naive approach – e.g., using uniform approximation – is unlikely to be efficient. Moreover, no method can be universally applied to all approximable applications. Therefore, the approximation strategy needs to be determined on a per-application basis.

One of the most commonly adopted techniques to introduce approximation is *precision-scaling*, also referred to as *bit-width reduction*. Essentially, it reduces the number of bits used for representing input data and intermediate operands [75, 87]. Precision-scaling basically combines close values into a single value, which paves the way for the *memoization* technique [44]. Memoization is based on storing the results of functions for later reuse with similar inputs. Finally, another quite widespread approach is *loop-perforation*, that is based on skipping some iterations of a loop to reduce computational overhead. It has proven to be effective when applied to several computational patterns, such as the Monte Carlo simulation, iterative refinement, and search space enumeration [73].

Concerning the approximation of circuits, the scientific literature distinguishes between timing and functional techniques [68]. The former consists of forcing the circuit to operate on reduced voltage or higher frequency than nominal ones, while the latter includes altering the logic being implemented. Technology-independent functional approximation currently represents the most popular technique to introduce approximations within hardware components, and many libraries consisting of thousands of elementary approximate circuits have been proposed in the scientific literature, supplying hundreds of implementations of even a single arithmetic operation [57, 42].

As for error assessment, it typically requires the simulation of both exact and approximate applications. Nevertheless, Bayesian inference [76] or machine-learning based approaches [58] have been proposed to reduce computation-demanding simulations. However, these approaches can provide only an estimate of the error, meaning that they do not offer any guarantees on the maximum value that the error can yield. Hence, various analytical and formal approaches have been proposed and applied for exact quantification of the error. They do not make any assumption on the

structure of the approximate circuits, and, albeit potentially time-consuming, they can be applied to determine almost every error metric [78].

Finally, concerning DSE, initial approaches either combine multiple design objectives in a single-objective optimization problem or optimize a single parameter while keeping the others fixed. Therefore, the resulting solutions are centered around a few dominant design alternatives [31]. Recently published studies address the approximate design problem by using MOP to search for Pareto-optimal approximate circuit implementations [14]. Unfortunately, such approaches did not focus on complex systems, rather on arithmetic components, such as adders and multipliers, since they are building-blocks for more complex designs. Conversely, in the following section we foster an application-independent, unified methodology able to automatically explore the impact of different approximation techniques on a given application, while resorting to the AxC design paradigm and MOP-based DSE.

3 Automatic Application-driven, Multi-Objective Approximate Design Methodology

This section addresses the automatic approximation of computer systems through multi-objective optimization. We describe the main steps of the methodology, and how the method helps in addressing the challenges that the AxC paradigm poses to the designer. As discussed in Section 2.1, there are several challenges to be addressed to effectively exploit the AxC design paradigm. Although diverse research articles in the scientific literature proposed well-founded approaches addressing the above-mentioned challenges, there are still plenty of open ones holding AxC back from wider employment. In particular, one of the key points is the lack of a general and automatic DSE methodology. Indeed, existing AxC design tools consider specific transformations and domains, and they are not fully automatic, providing only a guided approach for approximation. Therefore, in the following we discuss a generic, MOP-based and fully automatic methodology to design hardware accelerators for error-resilient applications.

In particular, we break down our methodology into different phases: (i) how to identify which part of the application is amenable for approximation and (ii) a suitable approximation technique, (iii) how MOP-based DSE can be defined, and, finally, (iv) how to pinpoint suitable fitness-functions for error assessment and performance estimation to effectively drive the DSE toward Pareto-optimal approximate configurations.

3.1 Identifying approximable portions and suitable approximation techniques

The first challenge to be addressed when dealing with the AxC is identifying error-resilient – i.e., approximable – data or portions of a given algorithm/application, and, consequently, a suitable approximation technique. Although it seems trivial, this step of the methodology is rather quite crucial. Indeed, as we discuss in the following, an improper design choice concerning either parts to be approximate, or the technique to be adopted, impacts all the subsequent phases.

Despite many of the methods from the scientific literature claiming to be generic, they actually require the designer to have in-depth knowledge of the target application to choose a suitable approximation technique. Unfortunately, this may not be trivial, or even not possible: there are plenty of applications for which, albeit conceptually simple, having profound understanding is very difficult indeed, e.g, DNNs and DT MCSs. Furthermore, once portions to be approximated have been correctly pinpointed, and a suitable approximate technique selected, the actual approximation has to be performed. However, manual introduction of approximation within applications is definitely inconvenient, due to their complexity or due to the amount of data/operations amenable for approximation.

Conversely, the methodology we are bound to discuss requires only minimal knowledge of the target application and provides the designer with a systematic approach to automatically generate approximate variants. An approximation variant is an implementation of a given application where approximable parts are implemented by approximate components. In general, the goal is to automatically generate approximate variants while having control on the error. Therefore, it is needed to collect information on the operations suitable for approximation. The gathering process can be automated by analyzing the Abstract Syntax Tree (AST) of the given an algorithm implementation. Then, AST manipulation using *mutators* [17] allows automatic generation of approximate variants.

Mutators are defined as a set of search-and-modify rules on the AST; the rule definition is generally application-independent, and does not require the designer to know the algorithm or its specific implementation. Furthermore, mutators do not depend on the specific approximation technique being adopted, and they effectively allow introducing a suitable tuning knobs for approximation, replacing exact operations within the AST using their approximate counterparts. Consider, for instance, an approximate multiplier designed using the precision-scaling technique: let the Number of Approximate Bit (NAB) be the parameter for such approximation and suppose the approximate operation truncates the least nab significant bits of operands, with nab being configurable. Setting a value for the nab parameters tunes the approximation degree, resulting in an *approximate configuration* of the algorithm. Mutators can be exploited, for instance, to implement the *inexact-component* technique. Indeed, exact multiplications can be automatically replaced using a mutator that allows selecting which implementation to be adopted among those provided by a given library, e.g., the EvoApproxLib library [57]. In this case, the configuration parameter would allow selection of an optimal multiplier implementation regarding a given error metric, required silicon area, and power dissipation.

3.2 Optimization and design-space exploration

The number of approximate variants and, consequently, the number of approximate configurations, grows quickly with the number of parts suitable for approximation. Consider, for instance, an algorithm implementation with n approximable operations, each allowing k different degrees of approximation: $\binom{n}{j}$ different approximate variants can be defined by simultaneously approximating j operations, and k^j different approximate configurations can be defined for each of the variants. There-

fore, the total number of approximate configurations is $\sum_{i=1}^{n} k^i \times \binom{n}{i}$. At this point, the main challenge is to find values for the approximation parameters leading to the Pareto-optimal trade-offs between performance gains and accuracy losses.

In facts, each one of the introduced approximation parameters impacts both accuracy and performance. Hence, the automated design of approximate applications is inherently a MOP in which variants satisfying user-defined constraints and showing the desired trade-off between the quality and other performance-related parameters is sought within all possible implementations [79]. As we mentioned, most of the approximation approaches either combine multiple design objectives in a single-objective optimization problem or optimize a single parameter while keeping the others fixed. Therefore, the resulting solutions are centered around a few dominant design alternatives [31]. We propose to find Pareto optimal configurations for approximation parameters through an automatic MOP-based DSE, which is not only tailored to the target application, yet it considers the latter target as a whole. Indeed, recent works addressing the circuit design problem as MOP, e.g., [72], did not focus on complex systems, rather on arithmetic components, such as adders and multipliers, since they are building-blocks for more complex designs.

In the following, we provide the reader with the required knowledge concerning MOP.

3.2.1 Multi-objective Optimization Problems

Basically, a MOP is an optimization problem involving multiple objectives. More formally, given the set of *fitness-functions* (1), or *objective-functions*, and the set of *constraints* (2), a MOP can be formulated as in Equation (3). While the functions of the former set assume values in $\mathbb{R}$, or its subset, the constraint functions assume either the value *1* or *0* to indicate that the constraint is or is not met, respectively.

$$\Gamma = \{\gamma_i : X \rightarrow \mathbb{R},\ i = 1 \cdots k\},\ X \subseteq \mathbb{R}^n \tag{1}$$

$$\Psi = \{\psi_j : A \rightarrow \{0,1\},\ j = 1 \cdots l\ \},\ A \subseteq \mathbb{R}^n \tag{2}$$

$$\begin{aligned} &min/max\ \{\gamma \in \Gamma\} \\ &s.t.\ \psi \in \Psi \end{aligned} \tag{3}$$

The $X \subseteq \mathbb{R}^n$, which is defined by constraints (2), is the set of feasible solutions to the MOP, or the *feasible set*. An element $x \in X$ is a *feasible-solution*, while its image through Γ, i.e. $z = \{\gamma(x), \gamma \in \Gamma\}$, is called the *outcome* of x.

Let us consider two solutions, $x, y \in X : x \neq y$, x is said to *dominate* y i.f.f. $x \prec y \iff \gamma_i(x) \leq \gamma_i(y)\ \forall i \in [1,k]\ \wedge\ \exists j \in [1,k] : \gamma_j(x) < \gamma_j(y)$ holds, i.e., x shows better or equally good objective values than y in all objectives and at least better in one objective. If a solution is not dominated by any others, it is called a *Pareto-optimal* solution.

Due to the rapid growth of the size of the solution space as the number of decision variables, fitness-functions and constraints increases, using exact solving algorithms for MOPs turns out to be very computation-intensive and time-consuming. Consequently, a variety of (meta-)heuristics aiming at producing an approximation

of the Pareto-front have been proposed in the scientific literature: Genetic Algorithm (GA) [52], Simulated Annealing (SA) [45] and particle swarm [30] are just some of the most commonly adopted ones, and among the heuristics belonging to the mentioned families, the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [32] and Archived Multi-Objective Simulated Annealing (AMOSA) [8] are the most common ones.

3.2.2 MOP modeling: identifying decision-variables and suitable fitness functions

In the context of AxC, modeling a specific optimization problem is not trivial, and no general rules exist. Anyway, considering the technique used to generate the approximate variants definitely helps in at least in identifying the decision variables of the problem. Indeed, the latter find natural correspondence in the configuration parameters introduced to govern the degree of approximation.

As already discussed, the identification of suitable decision-variables is only the first step to complete in order to define a MOP-based DSE. Indeed, we need to also define fitness-function driving the DSE. In particular, we must assess the error entailed by the approximations. Hence, we need to pinpoint an appropriate error metric to define a suitable error fitness-function to minimize. Unfortunately, when using the AxC paradigm, defining an appropriate error metric is of major concern, and it is usually not a trivial task. Therefore, the error-metric is usually selected case-by-case. Anyway, for some applications, the choice of error metric is obvious, if not outright forced, by the application-domain. The classification accuracy-loss, for instance, is a meaningful error metric for either DNNs and DT MCSs applications, while the Peak Signal-to-Noise Ratio (PSNR) or the Structural SIMilarity (SSIM) are common error metrics in the image-processing field [82].

For what pertains to performances in terms of either computational time, power consumption or hardware overhead, to accurately consider the resource savings in the DSE, we should measure area, power-consumption and maximum operating frequency of the explored approximate configurations. Unfortunately, this would require the synthesis and simulation of each approximate configuration explored during the DSE, which is definitely a time-consuming process. Therefore, we propose a model-based estimation of performance increases to drive the DSE. This has to consider the impact of the selected approximation technique on the final hardware implementation, to provide a faithful estimation, albeit not accurate. Defining such a model is not straightforward. Although removing some parts of an arithmetic circuit, for instance, undoubtedly leads to specific gains in terms of area/energy, model-based hardware-requirement estimation becomes trickier when the approximation has to be tailored to the application, and performance to be evaluated in the application's context, since they depend on the specific implementation.

3.3 Summary

For the reader's convenience, Figure 1 summarizes the proposed methodology. Starting from the model of the application to be approximated, an automatic approximation engine generates configurable approximate variants. Variants may either be generated from scratch starting from the model, or result from alterations

of the model itself. Furthermore, for each approximate portions, approximate variants allow to selectively adjust the degree of introduced approximation, through the use of convenient configuration parameters. The value for such parameters leading to optimal trade-offs between quality of results and performance gains is searched through a MOP-based DSE. The latter is performed using a suitable heuristic – e.g., either NSGA-II or AMOSA – minimizing the error entailed by the approximation and, at the same time, a figure of merit that correlates to performances, e.g. computational time, power consumption or hardware overhead. At the end of the DSE, resulting non-dominated approximate configurations are adopted to suitably shape a configurable implementation of the target application.

In the next sections we apply the proposed methodology to several applications from different domains, including logic, image-processing and machine-learning applications.

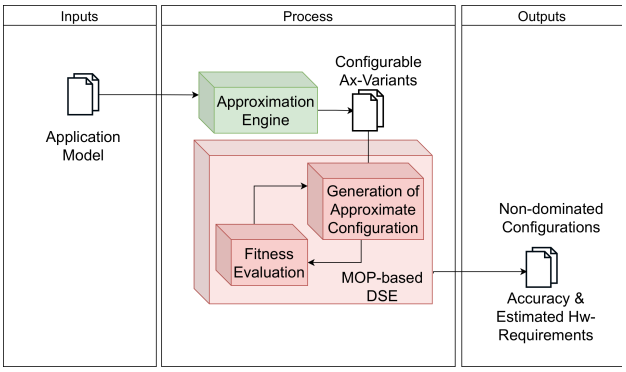


Fig. 1: Workflow of the discussed automatic design methodology

4 Automatic approximation of combinational circuits

In this section we discuss the design of combinational logic circuits, that is particularly relevant since combinational circuits typically constitute building-blocks for larger, more complex, designs. We resort to the catalog-based And-Inverter Graph (AIG)-rewriting technique from [13], and to the *pyALS* framework [16] that implements it. Anyway, the method we discuss is not constrained to a specific approximation technique.

Figure 2 sketches the overall flow of the *pyALS* framework [16]: starting from HDL source code describing the design under study, the approach goes through a first phase of circuit analysis and synthesis, then, it deals with the DSE by defining a MOP to obtain Pareto-optimal configurations in terms of error and hardware requirements.

Since the methodology is based on non-trivial local rewriting of AIGs and MOP, in the following we provide the reader with essentials concerning these building blocks.

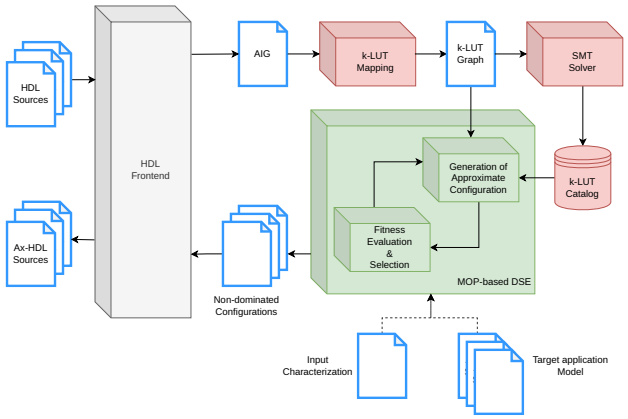


Fig. 2: Overview of the automatic workflow.

4.1 Approximate variants generation

The method from [13] exploits the AIG representation of digital circuits to introduce approximation. An AIG is a structural representation, based on directed acyclic graphs, for Boolean networks. In an AIG, nodes can be either Primary Input (PI) nodes that have no *fan-in*, meaning that they have no incoming edges and hence, there is no node driving them, or logic-AND nodes, which have two incoming edges. Furthermore, nodes having no *fan-out*, i.e, nodes that do not drive any other node, are called Primary Outputs (POs). For what pertains to edges, they represent physical connections between nodes, and they can be indicated as complemented or not. Conventionally, the polarity of complemented edges is 0. The AIG is non-canonical as a representation, which means that the same Boolean function can be realized by multiple, different AIGs. Briefly, given the AIG of a combinational circuit, the approach from [13] first enumerates k-feasible cuts, and then introduces approximation by superseding k-cuts with approximate ones. A k-cut of a node n, called *root*, is a set of at most K nodes of the AIG such that each path from a PI to n passes through at least one node of the set. Approximate k-cuts are generated by exploiting the Exact Synthesis (ES). Being f the Boolean function implemented by a k-cut, ES searches for those k-cuts implementing a function $f' \neq f$ that requires less resources w.r.t. f, and yet complies with error constraints. The latter are expressed in terms of the Hamming distance between f and f'. The main challenge is hence to find replacements leading to Pareto-optimal trade-offs between the quality of results and hardware requirements. This requires coping with two major concerns: (i) the number of approximate configurations grows exponentially with the size of the concerned Boolean functions; (ii) preserving the quality of results while pursuing a reduction in hardware requirements are conflicting design goals.

4.2 Design-space exploration

Once the catalog has been built, the main challenge is to find the combination of cut-replacements leading to Pareto-optimal trade-offs between error and performance, i.e., to perform a DSE.

The *pyALS* framework adopts the AMOSA [8] searching algorithm to orchestrate the DSE: k-Look-Up Table (LUT) nodes constitute the set of decision variables of the MOP, their indexes are assigned according to the topological ordering defined by the underlying graph, and their domain is given by catalog entries. Starting from a randomly chosen archived solution, the AMOSA selects a random LUTs and replaces it using a suitable element taken from the catalog, then fitness-functions are computed to state the Pareto-dominance relationship between the altered configuration and archived solutions.

As we mentioned, fitness-functions driving the DSE are error and silicon-area minimization. As far as silicon-area is concerned, we resort to a model-based gain estimation to drive the DSE. In particular, we estimate the silicon-area requirements from the number of AIG nodes, since the relationship between the latter and hardware requirements – in terms of critical path and LUTs or standard cells – has been empirically proven in [53]. We evaluate both the number of nodes and the depth for a given approximate configuration on Functionally-Reduced And-Inverter Graphs (FRAIGs) [54].

For what pertains to error-metrics, the one to be used strictly depends on the final target application. In the following, we discuss experimental results while targeting generic combinational logic circuits and arithmetic circuits. In the former case, we adopt the Error Probability (EP) metric, while in the latter case we resort to the Absolute Worst-Case Error (AWCE) metric.

4.3 Experimental results

To evaluate the proposed methodology, we first considered a subset of the LGSynt91 benchmark [85], including both logic and arithmetic circuits. We also considered arithmetic circuits from [39] for further evaluation. At the end of the DSE, the AMOSA heuristic provided several approximate configurations for each of the considered benchmark circuits. We synthesized resulting Hardware Description Language (HDL) implementations targeting the 45 nm standard-cell library, to measure actual hardware requirements.

Figure 3 plots silicon-area against EP for circuits from the LGSynth91 benchmark: the red star denotes the exact circuits, while blue dots denote approximate configurations resulting from our workflow. As the reader can observe, a general decreasing trend for silicon area can be observed while the error increase, and, depending on the specific error-resiliency of the circuit being considered, our technique allows achieving significant savings. Furthermore, although we do not report plots for brevity's sake, a similar trend can be observed for power consumption.

Figure 4, instead, plots the silicon-area against AWCE for various 8-bits and 16-bits arithmetic circuits, including array-tree multipler (ATM), Dadda-tree multipler (DTM), Wallace-tree multipler (WTM), carry-skip adder (CSkA), ripple-carry adder (RCA) Han-Carlson adder (HCA) and carry-lookahead adder (CLA). Note that the x-axis for Figure 4 is in semilogarithmic scale. Once again, the red star is the reference – i.e., the exact circuits – while the blue dots denote approximate configurations resulting from our workflow. A general decreasing trend for silicon area can be observed in these results too, denoting the overall approach allows effectively intro-

ducing approximation also within arithmetic circuits, resulting in significant savings as the allowed error increases.

5 Approximation of image-processing applications

In this section, we discuss the application of our methodology to the design of hardware accelerators for image processing. Image processing is one of the main fields of application for AxC, since imperceptible reduction of image quality can lead to important computational resources savings [28]. Specifically, we address the design of an accelerator for Discrete Cosine Transform (DCT), which is the most resource-demanding step of the JPEG, one of the most commonly adopted lossy image and video compression algorithms.

Before discussing the mentioned case-studies, we briefly describe the implementation of the methodology from Section 3 as a state-of-the-art approximation framework, i.e., we present the Evolutionary-IIDEAA Is a Design Exploration tool for Approximate Algorithm (E-𝕀DE𝔸) framework [17].

5.1 The E-𝕀DE𝔸 framework

Evolutionary-𝕀DE𝔸 (IIDEAA Is a Design Exploration tool for Approximate Algorithm), which is an approximation framework for C/C++ applications. Figure 5 sketches the overall flow of E-𝕀DE𝔸. It consists of two main components, i.e., (i) a source-to-source manipulation tool, *the clang-Chimera*, and (ii) an evolutionary search engine, *the Bellerophon*. E-𝕀DE𝔸 requires:

1. the original application, described as C/C++ code,
2. the set of approximate operators, i.e., mutators, and
3. the fitness-functions to select the appropriate approximation outcomes.

The green dotted box in Figure 5 highlights Clang-Chimera flow. Clang-Chimera is a mutation engine for C/C++ code. It is based on the Clang compiler [7], used to rapidly develop source-to-source C/C++ compilers. Clang-Chimera applies the set of mutators (i.e., AxC Operators) to the input application code. It exploits Low Level Virtual Machine (LLVM)/Clang facilities – e.g., *ASTMatcher* and *Rewriter* – to apply a given mutator and to make systematic modifications to the code. This process generates a set of mutated files that are the configurable approximate variants of the input application code. Clang-Chimera analyzes and manipulates the input application source code through its AST, which is a tree-based representation of the application code, where each node of the tree denotes a language construct of the analyzed code. A set of AST nodes defines an AST pattern, which corresponds to a specific structure of the code. Altering the AST allows introducing constructs that enable the approximation degree tuning. Clang-Chimera is already provided with a set of mutators implementing common approximation techniques, such as (i) two loop-perforation mutators, namely LOOP1 and LOOP2, (ii) two precision-scaling mutators for the floating-point arithmetic, namely Variable Precision Arithmetic (VPA) and FLexible Arithmetic Precision (FLAP), (iii) a precision scaling mutator for the integer arithmetic, namely TRUNC, and (iv) a mutator supporting approximate arithmetic operator models of circuits being part of the EvoApproxLib [57] and EvoApproxLibLite [61] libraries.

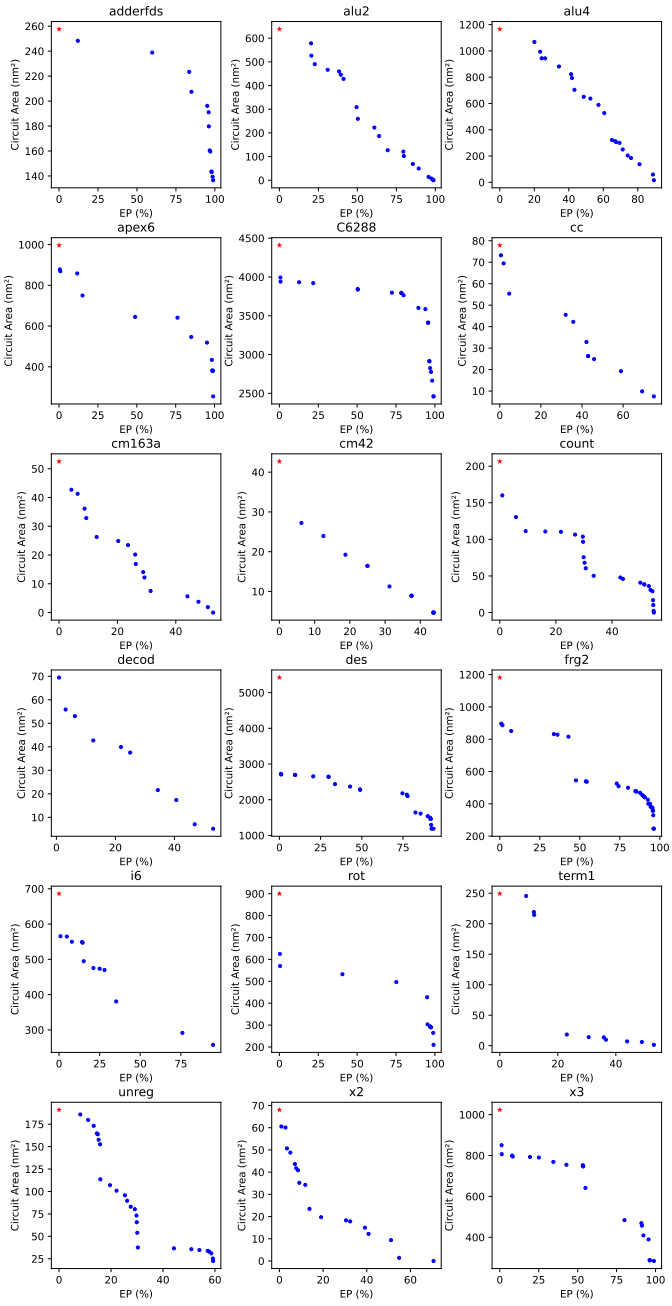


Fig. 3: Experimental results for LGSynt91 benchmark circuits.

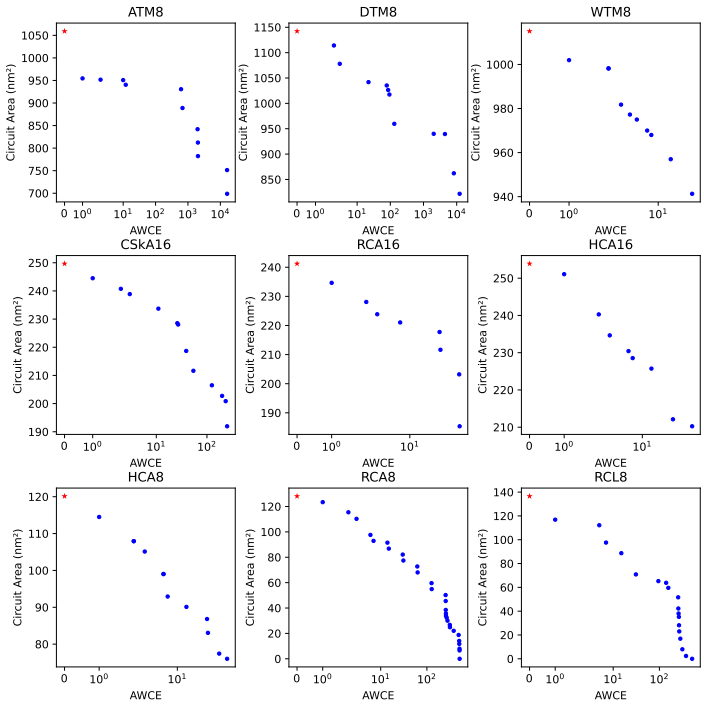


Fig. 4: Experimental results for arithmetic circuits. Kindly note that the x-axis is in semilogarithmic scale.

The solid-red box in Figure 5 depicts the Bellerophon flow. The tool analyzes the set of mutated files generated by Clang-Chimera and explores the different possible configurations of mutator tunable parameters, i.e., decision variables for the MOP driving the DSE. Therefore, Bellerophon explores the different approximate variants while 'moving' towards the *Pareto front* of the solutions, in terms of the defined fitness-functions. Fitness-functions might be defined accordingly to the particular exploited AxC technique. In the case of precision-scaling technique, for instance, a feasible fitness-function could be based on the NAB, since it translates in less hardware resources. Bellerophon is based on the NSGA-II [32], but, since implementing a full-featured NSGA-II may be cumbersome, it exploits the *ParadisEO* framework [50]. Furthermore, to be evaluated, each individual has to be compiled and executed. To speed up the execution time, the compilation strategy adopted by Bellerophon allows compiling just what it is necessary to retrieve information about approximate variants. Bellerophon uses the Just-in-Time engine provided by Clang-LLVM: each time the software needs to be altered to test a new variant, Bellerophon does not invoke the system loader, rather it alters the program image which is already loaded into the memory.

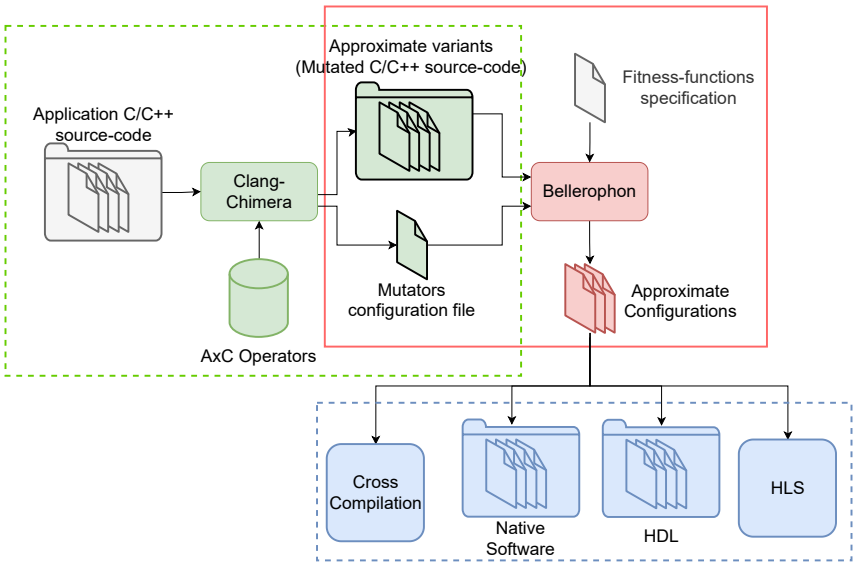


Fig. 5: E-IDEA flow, which includes Clang-Chimera and Bellerophon tool.

5.2 The DCT case-study

Most of the research work concerning image-processing applications focuses on the JPEG compression, either considering the algorithms as a whole or its individual computational steps. Concerning the design of hardware accelerators, researchers focused on the approximation of DCT accelerators, mainly targeting figures of merit such as circuit complexity, delay, area and power dissipation. Unfortunately, the effect of the different approximation techniques and relative configurations (i.e., approximation degrees) are only analyzed individually and without a supporting methodology.

In [4], for instance, a framework relying on inexact computing to perform the DCT computation for the JPEG has been proposed. The framework acts on three levels: (i) at the application level, it exploits human insensitivity to high-frequency variation to use a filter and discard high-frequency components; (ii) at the algorithmic level, multiplier-less fast algorithms are employed for the actual DCT computation on integer coefficients; (iii) at hardware level, rather than using a simple truncation for adder circuits, authors used Inexact-Adder Cells (IACs) to compute less significant bits instead of the Full-Adder Cells (FACs). Therefore, firstly, the JPEG quantization step is performed only low-frequency components of an image block; thus the high-frequency filter implementation comes down to simply setting some DCT coefficients to zero. Then, at the algorithmic level, since the DCT is the most effort-demanding step in JPEG, fast DCT algorithms have been used, reducing complexity from $O(N^2)$ to $O(N)$, and requiring only integer additions. Finally, at the hardware level, different families ofIAC are considered to further reduce the power-consumption.

The framework in [4] mainly aims at assessing the joint impact of those three levels of approximation. However, it presents one rather important shortcoming, i.e., approximation is introduced by manually tuning the individual approximation parameters. Conversely, in this case study, we assess the impact of approximation on the DCT computation by performing a fully automated DSE. Applying our methodology, we start from the DCT algorithm, and we perform an AST analysis to gather information on the operations suitable for approximation. Then, we generate parametric approximate versions which allow the approximation degree to be tuned through approximation parameters. Finally, we build a MOP to find the Pareto-optimal values for the aforementioned approximation parameters, using the NSGA-II to converge towards the Pareto-front.

First, we provide the reader with an overview on computing the DCT using fast algorithms in Section 5.2.1, as done for all the above-discussed case-studies. Then, we discuss the generation of approximate variants in Section 5.2.2, and several aspects concerning MOP-based DSE in Section 5.2.3, including MOP-modeling and fitness-functions to drive the DSE. Finally, in Section 5.2.4 we present experimental results.

5.2.1 Towards approximate DCT

As we already mentioned, the DCT computation is known to have $O(N^2)$ complexity and requires resource-intensive functional units, such as floating-point arithmetic modules. Cosine coefficients required by the DCT can be scaled and rounded such that floating-point operations can be superseded by integer ones: the resulting algorithms are significantly faster, and they find extensive use in practical applications. However, integer multiplication is still complex and resource intensive; thus, many low-complexity multiplier-less algorithms have been proposed [20, 21, 22, 18, 29, 64, 65]. They all avoid computing DCT transformed coefficients separately or iteratively, rather they extensively resort to matrix algebra and its properties.

The DCT-transform of an input image tile X, which is a 8×8 matrix, is given by $F = C \cdot X \cdot C'$, where C contains the cosine function values at the needed frequencies, and it is referred to as *DCT matrix*. Multiplier-less algorithms first split the C matrix into two sub-matrices, namely T and D, as reported in Equation (4). T contains only the values $\{0, \pm\frac{1}{2}, \pm 1, \pm 2\}$ and it is orthogonal – i.e. $T' = T^{-1} \Rightarrow TT' = T'T = I$, where I is the identity matrix – while D is a diagonal matrix consisting of values in the $[-1, 1]$ range, with $\{\frac{1}{2}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{8}}\}$ being typical values.

$$F = C \cdot X \cdot C' = D \cdot (T \cdot X \cdot T') \cdot D \tag{4}$$

The multiplication of D in (4) still requires floating-point operations; nevertheless, resorting to properties of diagonal matrices, the integer null-multiplicative part $T \cdot X \cdot T'$ can be isolated from floating-point operations required by D, as reported in Equation (5), where $\circ$ is the Hadamard product, i.e., an element-wise multiplication.

$$F = T \cdot X \cdot T' \circ (diag(D) \cdot diag(D)'), \tag{5}$$

Afterward, floating-point operations can be performed outside the DCT, and embedded into the JPEG quantization step, as shown in Equation (6), where $\hat{Q}$ is the *complete quantization matrix* and the $\oslash$ operator is the Hadamard division, i.e., an element-wise division.

$$\begin{aligned} F_Q &= \lceil F \oslash Q \rfloor = \lceil T \cdot X \cdot T' \circ (diag(D) \cdot diag(D)') \oslash Q \rfloor \\ &= \lceil T \cdot X \cdot T' \circ \hat{Q} \rfloor = \lceil (T \cdot (T \cdot X')') \circ \hat{Q} \rfloor \\ \hat{Q} &= (diag(D) \cdot diag(D)') \oslash Q, \end{aligned} \tag{6}$$

Besides allowing the use of integer arithmetic for the calculation of coefficients, Equation (6) also allows computing the two-dimensional DCT using the one-dimensional DCT transform twice, reducing the computational complexity from quadratic to linear.

5.2.2 Generating of approximate variants

Once the addition-based equations for the DCT coefficients are defined, simple implementations for the DCT computation algorithm can be derived. Within those, we introduce further approximation by replacing exact sums by configurable approximate ones. Such approximate sums allow setting two parameters, i.e., the NAB and the type of adder cell to use (namely, a classic FAC or an IAC). We consider three different IAC families, i.e., the Approximate Mirror Adder (AMA) [36], the Approximate XOR-based Adder (AXA) [86] and the IneXact Adder (InXA) [3].

As mentioned, this is the same approach adopted in [4]. However, while in [4] the approximation was manually introduced, we automate the replacement process by considering the AST of the algorithm implementation, resorting to E-𝕀DE𝔸 [17]. Thus, we model each of the above-mentioned multiplier-less DCT algorithms [20, 21, 22, 18, 29, 64, 65] by using C/C++ implementations, and starting from such implementations, the generation of approximate variants is performed using the Clang-Chimera tool [17]. For each DCT algorithm, the tool produces mutated sources which allow configuring, for each of the sums, both the NABs and type of adder hardware cell to use (i.e., either FAC or IAC).

5.2.3 Design-space exploration

Decision variables of the MOP are parameters introduced during the generation step for approximate variants, i.e., the NAB value and the type of adder hardware cell to be used for each of the approximate operations. Thus, if N_{op} is the number of addition required by a given algorithm, each approximate configuration is identified through the use of a vector, i.e., a chromosome, which is composed of $2 \cdot N_{op}$ different elements, or *genes*. Chromosomes are provided with an additional gene representing the approximation degree for the high-frequency filter. Hence, each chromosome is composed of $2 \cdot N_{op} + 1$ genes.

As for error fitness-function, we resort to Structural DiSSIMilarity (DSSIM) [82] to evaluate differences among images. The higher the DSSIM the higher the error between X and Y sets. We compute the DSSIM between a standard JPEG compressed image X and an image Y which is obtained by using a certain approximate

configuration of a given approximate algorithm, with both X and Y originate from the same non-compressed source image. We considered the SIPI image data set [1], that consists of 44 different images, covering a wide set of common features, including among others a flat gray scale, foreground subject with a messy background, and high contrast images. Concerning silicon-area requirements, we again resort to model-based estimation to drive the DSE, estimating the hardware overhead based on the number of transistors required to implement one-bit adder cells, using the data from [4].

5.2.4 Experimental results

To be able to measure the final gains, we encoded all the above-mentioned DCT algorithms in VHDL. Such implementations guarantee high flexibility: they handle the configuration of both the types of adder cells to use for each addition and the number of bits to approximate (NABs). This allows the synthesis of any solution eventually found as a result of the DSE process. VHDL implementations follow Equation (6), and each of the partial sums is performed using a configurable approximate adder: it allows computing the least significant bits of the sum using IACs, while the most significant bits are computed by classical FACs. The number of approximate bits, i.e. IACs, is configurable through the NAB parameter.

As we mentioned above, seven different DCT algorithms and ten types of IACs are considered during this case study. As for the DCT algorithms, we considered BAS08 [20], BAS09 [21], BAS11 [22], BC12 [18], CB11 [29], PEA12 [64] and PEA14 [65]. As for the IACs families, we considered AMA [36], AXA [86] and InXA [3]. All of these are encoded in the C++ language, and the generation of approximate variants is performed through the Clang-Chimera tool. The latter variants are, then, evolved using the Bellerophon tool. After the DSE, to correctly evaluate the final gains, we synthesized the obtained approximate configurations to both Application-Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA) technologies.

Concerning ASIC, we synthesized all the obtained non-dominated approximate configurations while targeting the 65 nm Fin Field-Effect Transistor (FinFET) technology through the Cadence *Genus Synthesis Solution* tool. We resorted to the synthesis reports for the silicon-die area of the approximate configurations. In Figure 6, we report the results. Pertaining to the power consumption, to determine whether the synthesis power report provides a satisfying accuracy, we simulated the whole workload for two algorithms (BAS08 and BAS09) and collected the resulting power consumption. As a result, we realized that the difference between the power consumption resulted from the workload simulation and that coming from the synthesis tool only differed by 5%, on average. We considered the synthesis report accuracy sufficient, thus in Figure 7 we show the power results from the synthesis report. Kindly note that the scale on the left axis (static power) is different from the scale on the right axis (dynamic power). Power savings are achieved due to both the reduced area and the lower switching activity that IACs exhibit w.r.t FACs, as also reported in [4].

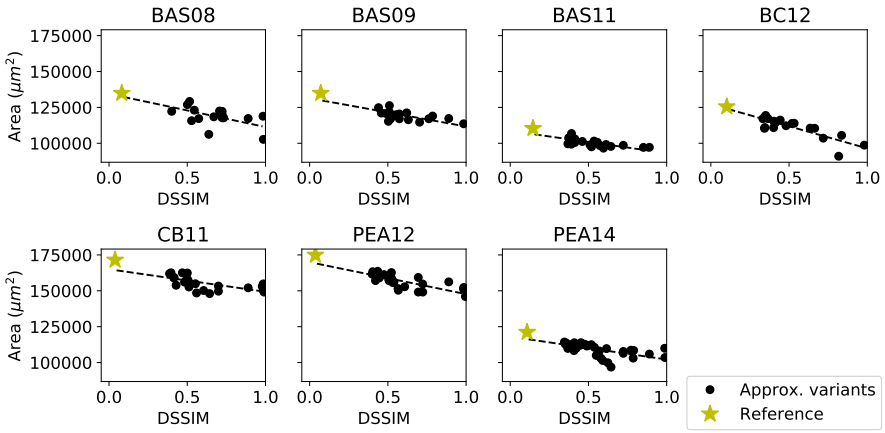


Fig. 6: silicon-die area requirements (in μm^2)DCT hardware-resource requirements while targeting the 65 nm FinFET technology

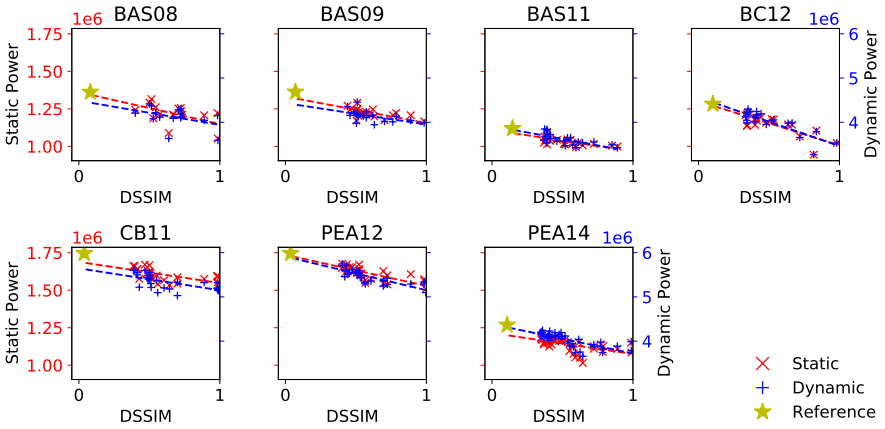


Fig. 7: Power-consumption requirements (in nW) DCT hardware-resource requirements while targeting the 65 nm FinFET technology

Regarding FPGA synthesis, we synthesized all the obtained non-dominated approximate configurations to a Xilinx Zynq-7020 MPSoC, using only its embedded FPGA and inhibiting Digital Signal Processings (DSPs) usage, to get a fair estimation of hardware requirements. Figure 8 reports the synthesis result in terms of number of LUTs for all the considered algorithms. As expected, approximate solutions require less resources than the precise implementation, as highlighted by the decreasing general trend. To correctly evaluate energy savings, we performed a post-synthesis timing simulation, using the Dynamic Power Analysis tool provided by the Xilinx Vivado. In this case, since the synthesis report has a very low confidence level for power consumption estimation, we resorted to a workload simulation for all the

solutions the DSE provided, for all the algorithms. In this way, we achieved a high confidence level of power estimation. Figure 9 shows static and dynamic power consumption for all the algorithms. The static power of the FPGA is largely caused by the fabric of the device and does not directly depend on used resources, while the dynamic one is directly linked to the user design, due to the input data pattern and the design internal activity. Being our hardware implementations of approximate DCT characterized by low overhead, i.e., device resources usage falls between 6 and 13%, it is necessary to split power consumption in static and dynamic since the former turned out to be about an order of magnitude greater than the latter one for the target FPGA device. Also, in this case, power savings are achieved thanks to both the reduced total area and the logical structure of IACs: FPGA LUTs implementing IACs have a lower switching activity than those implementing FACs, as reported in [4].

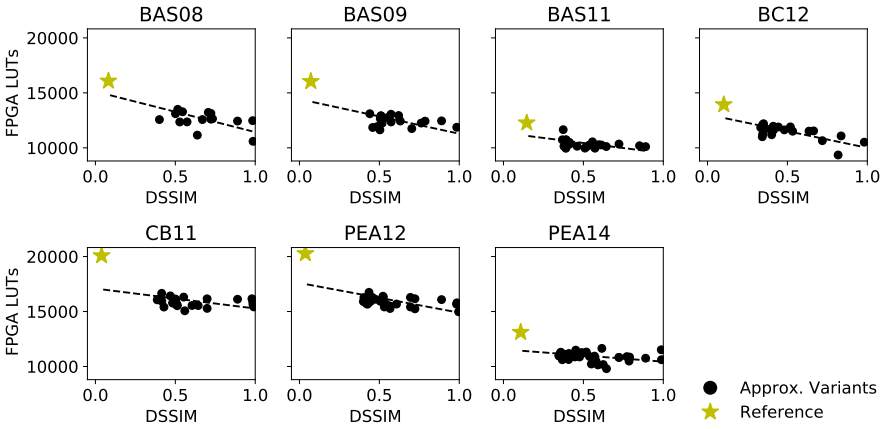


Fig. 8: LUTs requirements while targeting a Xilinx Zynq-7020 FPGA

Visual Test

Since JPEG belongs to the image processing domain, we also provide a visual test: Figure 10 shows, from left to right, the standard JPEG-compressed image of Baboon, as taken from the SIPI image database [1], the same image compressed using the exact version of the BC12 algorithm [18] – which exhibit a DSSIM of 0.10, and requires 125473.92 μm^2 and 5691946 μW when implemented on ASIC, or 5902 LUTs and 107933980 μW while targeting FPGA – and, finally, the ones compressed with its approximate variant having 0.33 as DSSIM value, which correspond to 8362.64 μm^2 and 352.711μW saved for ASIC and 1846 LUTs and 94506.744μW saved for FPGA. As the reader can easily figure out, the quality differences are barely perceivable.

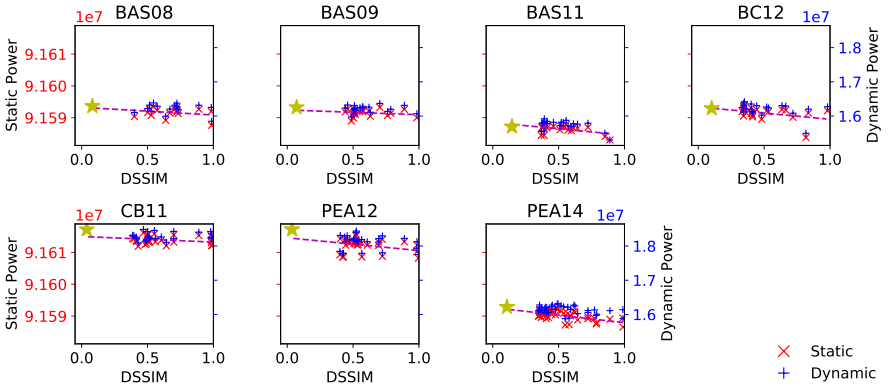


Fig. 9: Power-consumption (in nW) while targeting a Xilinx Zynq-7020 FPGA

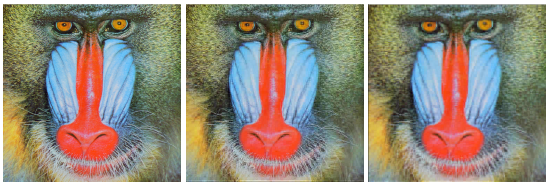


Fig. 10: Visual test

6 Automatic approximation of artificial intelligence applications

This section discusses the automatic approximation of two common Artificial Intelligence (AI) applications, namely DNNs and DT MCSs. These case studies are particularly relevant since state-of-the-art hardware accelerators targeting both these models are tremendously resource intensive, due to the massive amount of processing elements needed to effectively accelerate computations [27, 56]. In either case, the high demands in terms of both silicon area and power consumption utterly hinder the spread of commercial devices. The desire to reduce the hardware overhead by resorting to the AxC, however, cannot jeopardize more than half a century of efforts to achieve the accuracy that modern models exhibit. Luckily, the scientific literature demonstrated that MOP-based DSE can provide a full Pareto-front consisting of several trade-offs between considered fitness-functions to optimize [12, 15, 61].

In the following, we first provide the reader with a brief background concerning DNNs and DT MCSs before discussing how to design approximate systems based on that models.

6.1 Neural Networks

Recently, Artificial Neural Network (ANN) have won numerous contests in pattern-recognition, classification, object-recognition and so forth, imposing themselves on everyone's attention as one of the most successful learning techniques. The basic processing element is the *artificial neuron*: input signals are multiplied with *learned weights* at *synapses*, while *dendrites* carry the weighted input-signals to the *neuron*

body, where partial-products are summed and *biased*. An output is produced along the *axon* – i.e., the neuron "fires" – if the weighted sum is greater than a threshold, defined by the neuron's *activation function*. Synaptic weights, as well as biases, are learned by exploiting the backpropagation algorithm [47].

It is worth mentioning that the model of artificial neurons is quite different from that of biological ones: biological dendrites do not simply carry signals but, as well as biological synapses, they perform very complex – and still partially unknown – non-linear functions, and the exact instant in which the neuron "fires" encodes the information, not the frequency of firing [71].

Instead of being modeled as an amorphous blob of connected neurons, DNNs are organized in distinct *layers*, the number of which defines the network's depth. Three main types of layers are peculiar to Convolutional Neural Networks (CNNs), namely the Convolutional Layer (CL), the Pooling Layer (PL) and the Fully-Connected Layer (FCL). While PLs perform a single function, i.e., sub-sampling, CLs and FCLs perform computation that is not just function of the inputs, but also of learned synaptic-weights and biases. PLs are placed in between CLs, to progressively reduce the spatial size of the intermediate representation. In FCLs, neurons of the layer are connected to all the preceding layer, while in CLs, neurons in a layer are connected only to a small region of the previous layer. In any case, there is no connection between neurons within the same layer. In Recurrent Neural Networks (RNNs), cycles are allowed, while they are strictly forbidden in feed-forward ANNs, such as CNNs. Moreover, CNN architecture is constrained to be arranged in three dimensions, and explicit assumptions are made on the input, which is images, unveiling a more efficient forward function implementation, and a reduction in the amount of learned parameters.

6.1.1 Approximate DNNs

As we mentioned, the inner error-resiliency of ANN makes them the ideal field of application for AxC; consequently, a significant amount of research focused on both the training and the inference phase, attempting to further reduce resource requirements of hardware accelerators. In the following, we briefly report some of the most relevant contributions.

One of the approaches to identify approximation-resilient neurons in ANNs is discussed in [80]. It leverages the backpropagation algorithm [47] to obtain a measure of the sensitivity of the output of the considered DNNs, to the output of each neuron. The latter are sorted based on the magnitude of their average error contribution. Depending on whether the latter falls below a predetermined threshold, they are labeled as resilient or sensitive. Resilient neurons are replaced using approximate ones, which are designed using the precision-scaling technique, and allow modulating the bit-widths of both inputs and weights based on resilience. A subsequent retraining step suitably adjusts the learned parameters, alleviating the impact of approximation-induced errors, and allowing further approximation.

As mentioned, multipliers are recognized as the most demanding component within neurons. Therefore, as foreseeable, several contributions focus precisely on them. Authors of [6] investigated on properties that an approximate multiplier should

exhibit to maintain acceptable classification accuracy and, at the same time, reduce the use of silicon area. They observed that multipliers having low values for the variance σ_{ED} and Root Mean Squared Error (RMSE) do not deteriorate the classification accuracy. Furthermore, they noted that when a multiplier underestimates or overestimates the exact product with equal probability, the classification accuracy tends to increase, since such a multiplier prevents the errors from accumulating. However, this is a necessary-but-not-sufficient condition.

To alleviate approximation-induced error, the mentioned contributions resort to network-retraining. While, on one hand, this even allows for further approximation, on the other hand, it increases the overall design time. Moreover, retraining might not be possible, e.g., the dataset on which the network has been trained may not be available. A way to overcome this limitation while using approximate multipliers has been proposed in [59]: a weight-tuning algorithm adapts the learned weights to the employed multipliers, allowing accuracy recovering. The proposed algorithm exploits the fact that, for each of the multiplications, the value of the one operand – the one holding the synaptic weight – is constant, while the second operand varies with the input data. Thus, a *map-function* can be computed offline, and exploited during approximation to determine the suitable weight-update.

In [60], the EvoApprox8b library of arithmetic components [57], designed through Cartesian Genetic Programming (CPG) as in [72], is further evolved, and employed to conduct a resiliency analysis targeting CNNs, specifically, different networks belonging to the ResNet [37] family whereby 8-bits quantization is exploited to preemptively lower resource requirements. Further savings are pursued using approximate hardware components, selected as in [58]. In this regard, the EvoApprox8b is expanded considering both standard $n \times n$ and $m \times n$ approximate multipliers, and CNNs are approximated either considering a single layer at a time or the network as a whole. As for the former, all multiplications of all layers are replaced using one particular implementation taken from the mentioned library, regardless of layers' resiliency.

6.1.2 Automatic approximation of DNNs applications

Reviewing the reported contributions, the following drawbacks can be easily recognized: (i) they are related to a specific approximation technique; (ii) they typically require a re-training step to alleviate the impact of approximation on the classification accuracy, which undoubtedly increases the design time; (iii) they either optimize a single parameter (silicon area, for instance) under quality constraints, and (iv) although they recognize that each neuron may contribute to error and performances differently, depending on the layer it belongs to, the degree of introduced approximation does not consider these differences.

Conversely, the approach we discussed (i) supports different approximation techniques; (ii) does neither leverage the backpropagation algorithm nor require retraining, to avoid lengthening the design time and make the method applicable even when retraining is not possible; (iii) allows for the different degree of error resilience exhibited by different parts of the same application to be considered; and (iv) is based on multi-objective optimization, which allows for solutions that simultaneously op-

timize multiple figures of merit, e.g., classification accuracy, ASIC silicon-area or FPGA LUTs, power-consumption, etc.

In the following, we discuss a case-study concerning the automatic approximation a DNN. To perform approximate variants generation and DSE, we use the E-𝕀DE𝔸 framework that we extensively discussed in Section 5.1; hence, through a MOP-based DSE, we find the correct approximation degrees leading to non-dominated solutions exhibiting near-Pareto trade-offs between accuracy-loss and hardware-efficiency. Indeed, E-𝕀DE𝔸 allows specifying multiple fitness-functions; for our scenario, we define the accuracy-loss and the hardware requirements to be both minimized. Furthermore, this approach allows to selectively introduce approximation within layers while considering the network as a whole, contextually analyzing error resiliency of layers.

The precision-scaling technique, for instance, can be applied by carefully manipulating the AST to supersede precise multiplications and/or additions in CL or FCL. As discussed, such approximate operations should allow selecting the appropriate degree of approximation to be introduced, through tunable parameters. Nevertheless, the amount of such parameters can easily explode, since the number of operations within layers. Structural properties of layers, however, can be exploited to reduce the amount of introduced parameters while effectively introducing approximation. In CLs, for instance, *weights-sharing*, which reduces the number of parameters to be learned during the training phase by sharing synaptic weights among neurons within the same layer, allows applying the same approximation degree to all neurons belonging to the same CL. However, operations in different CLs must have their approximation degree. Neurons belonging to FCLs usually do not share synaptic weights, yet they process the same input volume. Thus, neurons belonging to the same FCL can share the same approximation degree. PLs can also be subject to approximation. Their contribution in terms of calculation burden is, however, negligible w.r.t. CLs and FCLs. Moreover, sub-sampling utilizing stride in convolutions is progressively supplanting PLs. Therefore, we do not apply any approximation to such types of layers. We configured Clang-Chimera to truncate input operands and results of multiplications in CLs and FCLs. Thus, the Clang-Chimera tool produces an approximate version which allows configuring, for each of the approximate layers, the approximation degree for the multiplications and additions involved in the weighted sum, depending on the considered layer.

To estimate the error introduced by the approximation, we configured Bellerophon to execute the approximate CNN on the training test data set, to assess the classification-accuracy loss. Concerning hardware-requirements, we estimate savings by considering several parameters that definitely have some impact on the former, such as (i) the input-volume of a neuron, which impacts the number of operations performed within it; (ii) the number of neurons within a layer, i.e., the output volume size of a layer, which impacts the hardware requirements of the whole layer.

In the following, we consider the LeNet5 network [48]. LeNet5 is a CNN that performs well in large-scale image processing: it is basic CNN when compared to the state-of-the-art architectures, but it is a common reference point. The network being considered has been trained to classify images from the Modified National In-

stitute of Standards and Technology (MNIST) test data set [49], which consists of a training set of 60000 examples, and a test set of 10000 examples of handwritten digits. The network has been trained using 64 bits floating-point, and exhibits a 99.07% accuracy when classifying images from the mentioned data set. We performed 8-bit quantization, without accuracy loss.

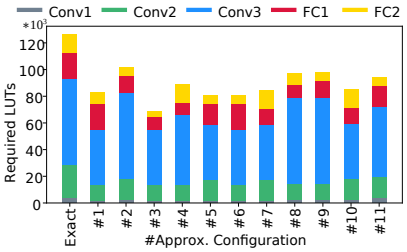
We configured the Clang-Chimera tool to supersede exact multiplications within the three CLs and two FCLs of LeNet5 using approximate multipliers designed while resorting to the precision-scaling technique. The latter allow selectively introducing approximation through configurable parameters. As discussed in Section 3, such configuration parameters constitute decision variables for MOP-based DSE; therefore, the Bellerophon tool encodes each approximate configuration, i.e., each individual, using a five-element-long vector, i.e., using a chromosome composed of five genes. Each of the latter governs the NAB for multiplications within a given layer. Concerning fitness-functions, we resort to simulations performed on the test dataset to assess the classification accuracy-loss due to approximation. As far as hardware-requirements are concerned, we estimate hardware requirements from the number of bits being kept after the precision-scaling is applied. Furthermore, we set our GA parameters as follows: initial population equals to 300 individuals, mutation, and crossover probabilities both set 0.9, and 31 generation epochs. Finally, we set a maximum error threshold equals to 1% accuracy loss.

After the DSE, to correctly evaluate the final gains, we performed FPGA synthesis of non-dominated configurations while targeting a Xilinx Virtex Ultrascale+. These syntheses involve only one single neuron, to provide a fair estimation of hardware requirements, i.e., as independent as possible from configuration parameters governing the structure of the accelerator. Furthermore, for the same reason, we disabled advanced FPGA features, e.g. DSPs, during syntheses.

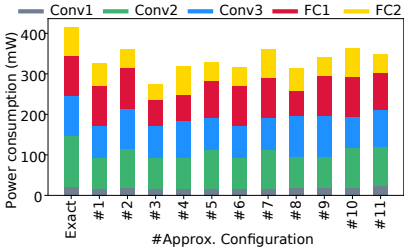
Figure 11a reports synthesis results: these stacked-bars graphs show, for each of the layers, the amount of FPGA LUTs required by a single neuron. A significant reduction, actually up to 45%, of required resources in terms of FPGA LUTs can be observed. As foreseeable, savings achieved due to precision-scaling do not only concern hardware requirements, but also energy consumption. Trivially, the less hardware a circuit requires, the less energy is spent to power it, and since the least significant bits of inputs, weights, and biases are always set to zero, the whole approximate circuit is also expected to exhibit a lower switching activity w.r.t. its exact counterpart. To evaluate potential power savings, we performed simulations on the exact neuron and on approximate ones lying on the Pareto-front, resulting from the DSE. Simulations involve 10000 input combinations, each consisting of an appropriate number of inputs, weights and bias vectors, depending on the input volume size of the considered neuron. Figure 11b reports simulation results. Again, up to 35% savings in terms of power consumption can be observed.

6.2 Decision-Tree based Multiple Classifier Systems

Decision Trees (DTs) stand out for their simplicity and high interpretability level, placing them as one of the most widely used classifier models [83]. A DT is a white-box classification model representing its decisions through a tree-like structure com-



(a) Required LUTs for 8-bits LeNet5 while targeting a Xilinx Virtex Ultrascale+ FPGA



(b) Estimated power consumption for 8-bits LeNet5 while targeting a Xilinx Virtex Ultrascale+ FPGA

Fig. 11: Hardware requirements for 8-bits LeNet5 while targeting a Xilinx Virtex Ultrascale+ FPGA

posed of an internal set of nodes containing *test conditions*, and leaf nodes which represent *class labels* [26]. Nodes are joined by arcs symbolizing possible outcomes of each test condition. Classes can be either categorical or numerical. In the former case, we refer to classification trees, while in the latter case, we refer to regression trees.

According to the number of attributes evaluated in each test condition – i.e., each internal node – two DT types can be induced: *univariate* and *multivariate* [26]. In the former, each test condition evaluates a single attribute to split the training set, while a combination of attributes is used in multivariate DTs. Some advantages of univariate DTs are their comprehensibility and the simplicity of their induction algorithms; however, they may include many internal nodes when the training data set instance distribution is complex. In such trees, test conditions are defined as $x_i \lessgtr c$, where x_i is the i-th attribute value, and c is a threshold value used to define a partition. Therefore, test conditions represent axis-parallel hyper planes dividing the instance-space, so they are also known as axis-parallel DTs. Anyway, when a categorical attribute is evaluated, the training set is split into as many subsets as values exist in the attribute domain.

In its essence, the procedure to construct DT MCS consists in using the training data set – that is constituted of historical, labeled data – to determine its best splits. These splits reduce the data set into smaller and smaller pieces, while aiming at splits that best emphasize the differences between data points belonging to the different partitions. The most widely adopted algorithms to construct DTs are CART [26] and C4.5 [66]. Significant improvements in classification accuracy have resulted from growing an ensemble of trees and letting them vote for the most popular class. Besides *bagging* predictors [23], several other techniques have been proposed. Some examples are *random split selection* [33], *random feature selection* [38], or searches for over a random selection of features for the best split at each node [5], and *random error injection* [24]. According to [25], all these procedures generate *random forest classifiers*.

6.2.1 Hardware accelerators targeting decision-tree based classifiers

As mentioned, their simplicity and understand-ability make DTs one of the most popular machine-learning algorithms [83]. Though, at the beginning, they were not considered for hardware accelerator, since they need only comparison to be performed during the inference phase; thus, they were not viewed as being computationally expensive. Nevertheless, the need to accelerate the inference phase emerged inherently in some application fields. Many real-time tasks require high prediction speed. However, the software implementation of the inference phase cannot meet the requirement even if the multi-threading technology is adopted. Hence, attention has been paid to the hardware-based accelerators [43].

As pinpointed in [62]. General Purpose - Graphic Processing Unit (GP-GPU) is an ill-fated choice for accelerating DT-based predictors since (i) high-precision arithmetic circuit – e.g., single/double precision Floating-Point Units (FPUs) – is inefficient both the amount of hardware and power consumption, since, during the inference phase, each node in the tree is evaluated using an *if-then-else* statements, which compare input values with constant values; (ii) DT-based predictors may consist of trees with a different size and depth, that causes an unbalanced computation; and (iii) all-to-all communication a DT-based predictor requires evaluating all the trees will certainly lead to performance bottleneck, since communication between near processing cores with the same local memory can be performed at a relatively high speed, while communication-penalty is large for the all-to-all communications. Authors of [77] compared performances of different implementations of hardware accelerators for DT-based predictors, including CPU, GP-GPUs and FPGAs while using off-line generated models. They empirically proved that FPGA implementations provide the highest performance, but may require a multi-chip / multi-board system to execute even modest sized predictors.

According to [51], architectures of FPGA-based accelerators for DT-based predictors can be categorized as comparator-centric and memory-centric. The former implements the model as a threshold network that consists of a layer of threshold logic units and a layer of combinational logic units, while the latter accelerates the model by introducing the pipeline in layers of the tree [70, 74, 63, 35, 2]. On the other hand, comparator-based accelerators reduce the reliance on memory elements by using custom comparators at each internal level [34, 41]. Comparator-centric accelerators could achieve high throughput and low latency as well; however, since designing such an accelerator requires all information of the model for the logic design, the architecture is tightly coupled with the underlying model, meaning any change in the model requires the accelerator to be reconfigured.

6.2.2 Approximate DTMCSs

Concerning DT MCSs, the wide-spread adoption of hardware-based accelerators is actually hindered by scalability issue, as reported in [77]. Nevertheless, there is very little research on both efficient architectures and methods to reduce their hardware-resource consumption, as the effort is mainly devoted to other classification systems, e.g., DNNs.

One of the few contributions from the scientific literature concerning the improvement of energy-efficiency of DT MCSs has been proposed in [81]. Authors noticed that, usually, only a part of the data of a given data set really needs the full computational power of a classifier. Therefore, they dynamically configure the classifier, making it more or less accurate, according to the difficulty in classifying the inputs. Hence, rather than building a single complex model, during the training phase they construct a set of models with progressively increasing complexity. Then, during the testing phase, the number of decision models applied to a given input varies depending on the difficulty of the considered input instance. To estimate the difficulty of a certain input, a confidence level for each classification is computed. If the latter confidence falls above a certain threshold, the classification process is terminated, otherwise, a more accurate classifier is used.

6.2.3 Automatic approximation of DT MCSs applications

In this section, we discuss a case-study concerning the approximation of a DT MCSs. In particular, we discuss the automatic approximation of some hardware accelerators trained to tackle with the Spambase dataset from the UCI Machine Learning Repository [40], which contains 4601 emails, 1813 of which are SPAM. This data set is freely available and makes use of 57 different features, expressed in the floating-point notation, to characterize elements that are part of the dataset. Each of the features specifies how often a word or a character appears in each element of the data set, i.e., in an email. During the training phase, conducted using the KNIME [46] tool, 40 different random-forest classifiers with a number of DTs ranging from 1 to 40 are trained.

Reference architectures

In the case studies discussed in this section, we consider the hardware implementation from [10]. In order to speed-up DTs visiting, the authors of [10] adopt a speculative approach, which consists in a DT flattening so that the visiting is performed over every possible path. Predicates are performed concurrently, regardless of the position and depth at which nodes are located, and a Boolean decision variable, that indicates whether a condition is fulfilled, is produced for each one of the evaluated predicates. To determine which leaf of the DT is reached, i.e., which class the input belongs to, a Boolean function, called *assertion*, is defined for each different class. Since a path that leads to a specific leaf is obtained by computing the logic-AND between the Boolean decision variables along that path, and since it is possible to compute the logic OR between the conditions related to different paths leading to leaves belonging to the same class, assertions can be defined as a sum of products Boolean functions. A majority-voter [11], combines the outcome of each tree to produce the outcome. The scalability of this approach has been formally demonstrated in [9]. In particular, the number of literals in each assertion is always less or equal to twice the size of the features set.

To generate approximate variants and to perform DSE, the authors of [12], resort to the E-IDEA framework, which we discussed in Section 5.1. Hence, through a MOP-based DSE, they find the correct approximation degrees leading to non-dominated solutions exhibiting near-Pareto trade-offs between accuracy-loss and hardware-efficiency, with accuracy-loss and the hardware requirements to be both minimized.

Generating approximate variants

Several opportunities inherently arise, for instance, from the hardware implementation which is discussed above. Both comparators and assertion-functions are excellent candidates for approximation, albeit the contributions of the former seem much more substantial w.r.t the latter, meaning their approximation can lead to significant savings. Furthermore, to introduce approximation, a wide plethora of techniques can be used regarding comparators, including precision-scaling, inexact hardware, and functional approximation. Since the latter is well suited to Boolean functions, it is also ideal for introducing approximation within assertion-functions.

As mentioned, DT MCSs may consist of hundreds, or even thousands of nodes, each comparing one of the many features considered by the predictive model with their corresponding threshold value. Consider introducing approximation through approximate comparators designed using the precision-scaling technique: such approximate comparators allow, through a configuration parameter, to govern the degree of introduced approximation by tuning the amount of neglected mantissa bits ad each comparison. Hence, the value to be assigned to each one of such parameters constitutes the decision variables of the MOP. Furthermore, the domain of such variables depends on the particular data-type being adopted for representing features. However, albeit straightforward, this naive MOP definition may result in an utterly infeasible DSE. Indeed, hardware implementations of DT MCSs may consist of hundreds, or even thousands of comparators, resulting in an enormous design-space. Nevertheless, we can exploit the fact that, albeit against different threshold values, the same feature can be taken into consideration for comparison several times while visiting trees. This allows to set the same degree of approximation to all the comparators processing the same feature, reducing the number of decision variables, and, consequently, the size of the solution space. Thus, being F the features set, each approximate configuration – i.e., individual in the GA context – can be represented using a vector consisting of $|F|$ elements – i.e., genes – each governing the degree of approximation for the corresponding feature.

As we mentioned, in [12], the actual generation of approximate variants is performed while resorting to the E-IDEA framework, applying the precision-scaling technique on comparators when using the mentioned double-precision floating-point representation [17].

Design space exploration

For what pertains to hardware-requirements, to accurately consider the resource savings in the DSE, we should measure area, power consumption and maximum clock speed of the explored approximate variants. This would require the hardware synthesis – and also simulations, in the case we are interested in measuring power consumption, for instance – of each variant explored in the DSE. This is utterly a time-consuming process. Hence, again, we resort to a model-based gain estimation to drive the DSE.

Since the purpose of the model is determining the $<$, $=$ or $>$ relation between two different approximate configurations, it is not necessary to focus on its accuracy. We rather focus on its fidelity, i.e., how often the estimated values are in the same relation as the real values for each pair of configurations. Concerning comparators, it is easy to recognize the fewer bit to compare, the fewer hardware requirements.

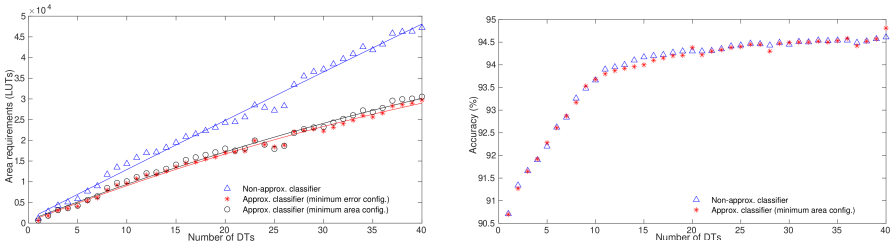
Results

The AxC exploration phase found, for each of the 40 classifiers, a certain number of approximate configurations on the Pareto-frontier, but for each of them only the configuration with minimum error and the one that requires less silicon area has been reported.

Figure 12a shows the area requirements in terms of LUTs, as the number of DTs used by the classifier increases. For all the measured quantities, an increasing trend, as the number of trees grows, is shown for area requirements. The growth, however, is clearly sublinear. In addition, it can be seen that the difference between the requirements of the exact classifier and the approximate one increases as the number of trees grows. This is because even if the complexity of single DTs – i.e., the number of nodes of which they consist of and the height of DTs themselves – decreases significantly as the number of trees used by the classifier increases, the total number of nodes increases, providing more approximation opportunities. This behavior can be observed also for the number of FPGA slices and registers, and both when considering solutions providing minimum error and those requiring the minimum silicon area. Furthermore, it can be noted that the difference in terms of area requirements between the minimum error and the minimum area solutions always remains negligible.

Figure 12b compares the levels of classification accuracy, as the number of trees used by the classifier increases, provided by the precise version – without approximation – and by the approximate version that has minimum area requirements. It is evident, from the graph, that there is only a small difference in accuracy between the configurations. Moreover, it remains minimal as the number of trees used for classification varies. On the other hand, the increase in the number DTs used in the classification process makes a smaller contribution as the number of DTs grows. This asymptotic behavior can be seen in exact and approximate classifiers, and it is because, by increasing the number of models, datasets involved for training turn out

simpler and corresponding DTs get less branched, which leads to a saturation of the accuracy level provided by the classifier model.



(a) Required LUTs (b) Accuracy

Fig. 12: Resource requirements and accuracy of approximate DT MCSs for Spambase

7 Conclusion

In this work, we discussed a unified methodology able to automatically explore the impact of different approximation techniques on a given application, while resorting to the AxC design paradigm and MOP-based DSE. We discussed the steps the methodology breaks into, while devoting particular relevance to all those that can be automated. In particular, we discussed how to effectively select parts of an application to be approximated and how to choose a well-suited approximation technique. Then, we discussed how approximate variants can be generated automatically, how to identify decision variables and suitable fitness-functions to define the MOP driving the DSE. The methodology is not dependent on a particular application, rather it can be generally applied to all applications.

To evaluate the proposed methodology, we selected some significant and relevant applications in the scope of the AxC paradigm, including generic combinatorial logic circuits, image-processing applications, and artificial intelligence applications. For what pertains to generic logic, we propose local rewriting of AIG, reducing the number of nodes and resulting in lower hardware resources requirements, while resorting to MOP-based DSE to carefully introduce approximation. We evaluate our approach using different benchmarks, and experimental results show our method allows performing a meaningful exploration of the design space to find the best trade-offs in a reasonable time, thus resulting in approximate circuits exhibiting lower requirements and restrained error. Concerning image processing applications, the discussed case study concerns the design of hardware-accelerators for the Discrete Cosine Transform (DCT), which is the most demanding step of the JPEG compression algorithm. We analyzed and modeled several algorithms from the literature to compute a fast and lightweight version of the DCT, and, for each algorithm, we applied approximation by substituting full-precise adders with several approximate ones from the literature having configurable approximation degree. For each algorithm, we performed

a DSE to find the non-dominated approximate designs in terms of trade-off between inaccuracy and gains. We modeled the DSE as a MOP and we used a GA to solve it, and after the DSE, we synthesized the obtained designs by targeting both FPGA and ASIC. Experimental results clearly showed that, with the proposed approach, it is possible to perform a meaningful DSE to find the best trade-offs between output accuracy and resource gains in a reasonable time. Finally, the comparison performed with previous work clearly showed the advantages of the proposed approach.

Finally, we applied our methodology to two of the most promising classification models in the machine-learning domain, namely Deep Neural Networks (DNNs) and Decision-Tree based Multiple Classifier Systems (DT MCSs). Leveraging the AxC design paradigm, a very limited quantity of classification-accuracy is traded off for a reduction in the silicon area requirements and power consumption of hardware-implemented DT MCS and CNN. Concerning DNNs, we exploited our methodology to investigate the impact of approximation on the classification accuracy, and experimental results prove the validity and efficiency of our methodology, even in applications in which the error has to be minimized as possible, providing savings up to 75% for silicon area and 50% for power consumption. Pertaining to DT MCSs, to prove the validity of the proposed approach of several classifiers, the optimal number of bits to be used to represent each of the features of the model is searched using NSGA-II. Among all Pareto-optimal hardware configurations, the one providing minimum classification error configuration and the one requiring the minimum amount of silicon area were considered for further consideration. Experimental results show a significant reduction in area requirements, for both of them. Furthermore, since the classification is very resistant to error, those configurations are very similar both in terms of area requirements and classification error.

References

[1] (1977) SIPI Image Database. URL `https://sipi.usc.edu/database/`

[2] Alcolea A, Resano J (2021) FPGA Accelerator for Gradient Boosting Decision Trees. Electronics 10(3):314, publisher: Multidisciplinary Digital Publishing Institute

[3] Almurib HA, Kumar TN, Lombardi F (2016) Inexact designs for approximate low power addition by cell replacement. In: 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 660–665, iSSN: 1558-1101

[4] Almurib HA, Kumar TN, Lombardi F (2018) Approximate DCT Image Compression Using Inexact Computing. IEEE Transactions on Computers 67(2):149–159, DOI 10.1109/TC.2017.2731770, conference Name: IEEE Transactions on Computers

[5] Amit Y, Geman D (1997) Shape Quantization and Recognition with Randomized Trees. Neural Computation 9(7):1545–1588, DOI 10.1162/neco.1997.9.7.1545, URL `https://direct.mit.edu/neco/article/9/7/1545-1588/6116`

[6] Ansari MS, Mrazek V, Cockburn BF, Sekanina L, Vasicek Z, Han J (2020) Improving the Accuracy and Hardware Efficiency of Neural Networks Using Approximate Multipliers. IEEE Transactions on Very Large Scale Integration

(VLSI) Systems 28(2):317–328, DOI 10.1109/TVLSI.2019.2940943, conference Name: IEEE Transactions on Very Large Scale Integration (VLSI) Systems

[7] Apple Inc (2023) Clang C Language Family Frontend for LLVM. URL https://clang.llvm.org/

[8] Bandyopadhyay S, Saha S, Maulik U, Deb K (2008) A Simulated Annealing-Based Multiobjective Optimization Algorithm: AMOSA. IEEE Transactions on Evolutionary Computation 12(3):269–283, DOI 10.1109/TEVC.2007.900837, conference Name: IEEE Transactions on Evolutionary Computation

[9] Barbareschi M (2016) Implementing Hardware Decision Tree Prediction: A Scalable Approach. In: 2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA), IEEE, Crans-Montana, Switzerland, pp 87–92, DOI 10.1109/WAINA.2016.171, URL http://ieeexplore.ieee.org/document/7471178/

[10] Barbareschi M, Del Prete S, Gargiulo F, Mazzeo A, Sansone C (2015) Decision Tree-Based Multiple Classifier Systems: An FPGA Perspective. International Workshop on Multiple Classifier Systems DOI 10.1007/978-3-319-20248-8

[11] Barbareschi M, Barone S, Mazzeo A, Mazzocca N (2019) Efficient Reed-Muller Implementation for Fuzzy Extractor Schemes. In: 2019 14th International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS), pp 1–2, DOI 10.1109/DTIS.2019.8735029

[12] Barbareschi M, Barone S, Mazzocca N (2021) Advancing synthesis of decision tree-based multiple classifier systems: an approximate computing case study. Knowledge and Information Systems pp 1–20, DOI 10.1007/s10115-021-01565-5, URL https://link.springer.com/article/10.1007/s10115-021-01565-5, company: Springer Distributor: Springer Institution: Springer Label: Springer Publisher: Springer London

[13] Barbareschi M, Barone S, Mazzocca N, Moriconi A (2022) A Catalog-based AIG-Rewriting Approach to the Design of Approximate Components. IEEE Transactions on Emerging Topics in Computing (in press) DOI 10.1109/TETC.2022.3170502

[14] Barbareschi M, Barone S, Mazzocca N, Moriconi A (2022) Design Space Exploration Tools. In: Bosio A, Ménard D, Sentieys O (eds) Approximate Computing Techniques: From Component- to Application-Level, Springer International Publishing, Cham, pp 215–259, DOI 10.1007/978-3-030-94705-7_8, URL https://doi.org/10.1007/978-3-030-94705-7_8

[15] Barone S (2022) Catalog of approximate LUTs for pyALS. URL https://github.com/SalvatoreBarone/pyALS-lut-catalog, original-date: 2021-12-29T10:31:12Z

[16] Barone S (2022) pyALS. URL https://github.com/SalvatoreBarone/pyALS, original-date: 2021-06-30T11:20:07Z

[17] Barone S, Traiola M, Barbareschi M, Bosio A (2021) Multi-Objective Application-Driven Approximate Design Method. IEEE Access 9:86975–86993, DOI 10.1109/ACCESS.2021.3087858, conference Name: IEEE Access

[18] Bayer FM, Cintra RJ (2012) DCT-like Transform for Image Compression Requires 14 Additions Only. Electronics Letters 48(15):919, DOI 10.1049/el.2012.1148, URL http://arxiv.org/abs/1702.00817, arXiv:1702.00817 [cs, stat]

[19] Bosio A, Ménard D, Sentieys O (eds) (2022) Approximate Computing Techniques: From Component- to Application-Level. Springer International Publishing, Cham, DOI 10.1007/978-3-030-94705-7, URL https://link.springer.com/10.1007/978-3-030-94705-7

[20] Bouguezel S, Ahmad MO, Swamy MNS (2008) Low-complexity 8$\times$ 8 transform for image compression. Electronics Letters 44(21):1249–1250, publisher: IET

[21] Bouguezel S, Ahmad MO, Swamy MNS (2009) A fast 8×8 transform for image compression. In: 2009 International Conference on Microelectronics - ICM, pp 74–77, DOI 10.1109/ICM.2009.5418584, iSSN: 2159-1679

[22] Bouguezel S, Ahmad MO, Swamy M (2011) A low-complexity parametric transform for image compression. In: 2011 IEEE International Symposium of Circuits and Systems (ISCAS), pp 2145–2148, DOI 10.1109/ISCAS.2011.5938023, iSSN: 2158-1525

[23] Breiman L (1996) Bagging predictors. Machine Learning 24(2):123–140, DOI 10.1007/BF00058655, URL http://link.springer.com/10.1007/BF00058655

[24] Breiman L (1998) Randomizing Outputs to Increase Prediction Accuracy. Machine Learning p 14

[25] Breiman L (2001) Random forests. Machine learning 45(1):5–32, publisher: Springer

[26] Breiman L, Friedman JH, Olshen R, Stone CJ (1984) Classification and Regression Trees. Routledge Publisher: Wadsworth

[27] Capra M, Bussolino B, Marchisio A, Shafique M, Masera G, Martina M (2020) An Updated Survey of Efficient Hardware Architectures for Accelerating Deep Convolutional Neural Networks. Future Internet 12(7):113, DOI 10.3390/fi12070113, URL https://www.mdpi.com/1999-5903/12/7/113, number: 7 Publisher: Multidisciplinary Digital Publishing Institute

[28] Chippa VK, Chakradhar ST, Roy K, Raghunathan A (2013) Analysis and characterization of inherent application resilience for approximate computing. In: Proceedings of the 50th Annual Design Automation Conference on - DAC '13, ACM Press, Austin, Texas, p 1, DOI 10.1145/2463209.2488873, URL http://dl.acm.org/citation.cfm?doid=2463209.2488873

[29] Cintra RJ, Bayer FM (2011) A DCT Approximation for Image Compression. IEEE Signal Processing Letters 18(10):579–582, DOI 10.1109/LSP.2011.2163394, conference Name: IEEE Signal Processing Letters

[30] Coello Coello C, Lechuga M (2002) MOPSO: a proposal for multiple objective particle swarm optimization. In: Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600), vol 2, pp 1051–1056 vol.2, DOI 10.1109/CEC.2002.1004388

[31] Das I, Dennis JE (1997) A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems. Structural Optimization 14(1):63–69, DOI 10.1007/BF01197559, URL http://link.springer.com/10.1007/BF01197559

[32] Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2):182–197, DOI 10.1109/4235.996017, conference Name: IEEE Transactions on Evolutionary Computation

[33] Dietterich TG (1998) An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. Machine learning 40(2):19

[34] Dávila-Rodríguez IA, Nuño-Maganda MA, Hernández-Mier Y, Polanco-Martagón S (2019) Decision-Tree Based Pixel Classification for Real-time Citrus Segmentation on FPGA. In: 2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig), pp 1–8, DOI 10.1109/ReConFig48160.2019.8994792, iSSN: 2640-0472

[35] Elnawawy M, Sagahyroon A, Shanableh T (2020) FPGA-Based Network Traffic Classification Using Machine Learning. IEEE Access 8:175637–175650, DOI 10.1109/ACCESS.2020.3026831, conference Name: IEEE Access

[36] Gupta V, Mohapatra D, Raghunathan A, Roy K (2013) Low-Power Digital Signal Processing Using Approximate Adders. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 32(1):124–137, DOI 10.1109/TCAD.2012.2217962, conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

[37] He K, Zhang X, Ren S, Sun J (2016) Deep Residual Learning for Image Recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Las Vegas, NV, USA, pp 770–778, DOI 10.1109/CVPR.2016.90, URL http://ieeexplore.ieee.org/document/7780459/

[38] Ho TK (1998) The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence 20(8):832–844, DOI 10.1109/34.709601, conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence

[39] Homma N, Aoki T (2022) Arithmetic Module Generator. URL https://www.ecsis.riec.tohoku.ac.jp/topics/amg/

[40] Hopkins M, Reeber E, Forman G, Suermondt J (1999) Spambase Data Set. URL https://archive.ics.uci.edu/ml/datasets/spambase

[41] Ikeda T, Sakurada K, Nakamura A, Motomura M, Takamaeda-Yamazaki S (2020) Hardware/Algorithm Co-optimization for Fully-Parallelized Compact Decision Tree Ensembles on FPGAs. In: Rincón F, Barba J, So HKH, Diniz P, Caba J (eds) Applied Reconfigurable Computing. Architectures, Tools, and Applications, Springer International Publishing, Cham, Lecture Notes in Computer Science, pp 345–357, DOI 10.1007/978-3-030-44534-8_26

[42] Jiang H, Santiago FJH, Mo H, Liu L, Han J (2020) Approximate Arithmetic Circuits: A Survey, Characterization, and Recent Applications. Proceedings

of the IEEE pp 1–28, DOI 10.1109/JPROC.2020.3006451, URL https://ieeexplore.ieee.org/document/9165786/

[43] Jiang W, Prasanna VK (2009) Large-scale wire-speed packet classification on FPGAs. In: Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, pp 219–228

[44] Keramidas G, Kokkala C, Stamoulis I (2015) Clumsy Value Cache: An Approximate Memoization Technique for Mobile GPU Fragment Shaders. Workshop on approximate computing (WAPCO'15) p 6

[45] Kirkpatrick S (1984) Optimization by simulated annealing: Quantitative studies. Journal of Statistical Physics 34(5-6):975–986, DOI 10.1007/BF01009452, URL http://link.springer.com/10.1007/BF01009452

[46] KNIME AG (2022) KNIME | Open for Innovation. URL https://www.knime.com/

[47] LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD (1989) Backpropagation Applied to Handwritten Zip Code Recognition. Neural Computation 1(4):541–551, DOI 10.1162/neco.1989.1.4.541, URL https://doi.org/10.1162/neco.1989.1.4.541

[48] Lecun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11):2278–2324, DOI 10.1109/5.726791, conference Name: Proceedings of the IEEE

[49] LeCun Y, Cortes C, Burges C (1998) MNIST Handwritten digit database. URL http://yann.lecun.com/exdb/mnist/

[50] Liefooghe A, Jourdan L, Legrand T, Humeau J, Talbi EG (2010) ParadisEO-MOEO: A Software Framework for Evolutionary Multi-Objective Optimization. In: Kacprzyk J, Carlos A Coello Coello, Dhaenens C, Jourdan L (eds) Advances in Multi-Objective Nature Inspired Computing, vol 272, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 87–117, DOI 10.1007/978-3-642-11218-8_5, URL http://link.springer.com/10.1007/978-3-642-11218-8_5, series Title: Studies in Computational Intelligence

[51] Lin X, Blanton RS, Thomas DE (2017) Random Forest Architectures on FPGA for Multiple Applications. In: Proceedings of the on Great Lakes Symposium on VLSI 2017, Association for Computing Machinery, New York, NY, USA, GLSVLSI '17, pp 415–418, DOI 10.1145/3060403.3060416, URL https://doi.org/10.1145/3060403.3060416

[52] Melanie M (1998) An Introduction to Genetic Algorithms. MIT Press p 162

[53] Mishchenko A, Cho S, Chatterjee S, Brayton R (2007) Combinational and sequential mapping with priority cuts. In: 2007 IEEE/ACM International Conference on Computer-Aided Design, pp 354–361, DOI 10.1109/ICCAD.2007.4397290, iSSN: 1558-2434

[54] Mishchenko A, Chatterjee S, Jiang R, Brayton R (2015) FRAIGs: A Unifying Representation for Logic Synthesis and Verification. ERL Technical Report p 7

[55] Mittal S (2016) A Survey of Techniques for Approximate Computing. ACM Computing Surveys 48(4):1–33, DOI 10.1145/2893356, URL https://dl.acm.org/doi/10.1145/2893356

[56] Mittal S (2020) A survey of FPGA-based accelerators for convolutional neural networks. Neural Computing and Applications 32(4):1109–1139, DOI 10.1007/s00521-018-3761-1, URL https://doi.org/10.1007/s00521-018-3761-1

[57] Mrazek V, Hrbacek R, Vasicek Z, Sekanina L (2017) EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2017, pp 258–261, DOI 10.23919/DATE.2017.7926993, iSSN: 1558-1101

[58] Mrazek V, Hanif MA, Vasicek Z, Sekanina L, Shafique M (2019) autoAx: An Automatic Design Space Exploration and Circuit Building Methodology utilizing Libraries of Approximate Components. In: 2019 56th ACM/IEEE Design Automation Conference (DAC), pp 1–6, iSSN: 0738-100X

[59] Mrazek V, Vasicek Z, Sekanina L, Hanif MA, Shafique M (2019) ALWANN: Automatic Layer-Wise Approximation of Deep Neural Network Accelerators without Retraining. 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD) pp 1–8, DOI 10.1109/ICCAD45719.2019.8942068, URL http://arxiv.org/abs/1907.07229, arXiv: 1907.07229

[60] Mrazek V, Sekanina L, Vasicek Z (2020) Libraries of Approximate Circuits: Automated Design and Application in CNN Accelerators. IEEE Journal on Emerging and Selected Topics in Circuits and Systems 10(4):406–418, DOI 10.1109/JETCAS.2020.3032495, conference Name: IEEE Journal on Emerging and Selected Topics in Circuits and Systems

[61] Mrazek V, Sekanina L, Vasicek Z (2020) Using Libraries of Approximate Circuits in Design of Hardware Accelerators of Deep Neural Networks. In: 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), pp 243–247, DOI 10.1109/AICAS48895.2020.9073837

[62] Nakahara H, Jinguji A, Sato S, Sasao T (2017) A Random Forest Using a Multi-valued Decision Diagram on an FPGA. In: 2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL), pp 266–271, DOI 10.1109/ISMVL.2017.40, iSSN: 2378-2226

[63] Owaida M, Kulkarni A, Alonso G (2019) Distributed Inference over Decision Tree Ensembles on Clusters of FPGAs. ACM Transactions on Reconfigurable Technology and Systems 12(4):1–27, DOI 10.1145/3340263, URL https://dl.acm.org/doi/10.1145/3340263

[64] Potluri US, Madanayake A, Cintra RJ, Bayer FM, Rajapaksha N (2012) Multiplier-free DCT approximations for RF multi-beam digital aperture-array space imaging and directional sensing. Measurement Science and Technology 23(11):114003, DOI 10.1088/0957-0233/23/11/114003, URL https://iopscience.iop.org/article/10.1088/0957-0233/23/11/114003

[65] Potluri US, Madanayake A, Cintra RJ, Bayer FM, Kulasekera S, Edirisuriya A (2014) Improved 8-Point Approximate DCT for Image and Video Compression Requiring Only 14 Additions. IEEE Transactions on Circuits and Systems I: Regular Papers 61(6):1727–1740, DOI 10.1109/TCSI.2013.2295022, conference Name: IEEE Transactions on Circuits and Systems I: Regular Papers

[66] Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA

[67] Raha A, Venkataramani S, Raghunathan V, Raghunathan A (2015) Quality configurable reduce-and-rank for energy efficient approximate computing. In: 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 665–670, DOI 10.7873/DATE.2015.0569, iSSN: 1558-1101

[68] Ranjan A, Venkataramani S, Jain S, Kim Y, Ramasubramanian SG, Raha A, Roy K, Raghunathan A (2019) Automatic Synthesis Techniques for Approximate Circuits. In: Reda S, Shafique M (eds) Approximate Circuits: Methodologies and CAD, Springer International Publishing, Cham, pp 123–140, DOI 10.1007/978-3-319-99322-5_6, URL https://doi.org/10.1007/978-3-319-99322-5_6

[69] Roy P, Ray R, Wang C, Wong WF (2014) ASAC: automatic sensitivity analysis for approximate computing. In: Proceedings of the 2014 SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems, ACM, Edinburgh United Kingdom, pp 95–104, DOI 10.1145/2597809.2597812, URL https://dl.acm.org/doi/10.1145/2597809.2597812

[70] Saqib F, Dutta A, Plusquellic J, Ortiz P, Pattichis MS (2015) Pipelined Decision Tree Classification Accelerator Implementation in FPGA (DT-CAIF). IEEE Transactions on Computers 64(1):280–285, DOI 10.1109/TC.2013.204, conference Name: IEEE Transactions on Computers

[71] Schmidhuber J (2015) Deep learning in neural networks: An overview. Neural Networks 61:85–117, DOI 10.1016/j.neunet.2014.09.003, URL https://linkinghub.elsevier.com/retrieve/pii/S0893608014002135

[72] Sekanina L, Vasicek Z, Mrazek V (2019) Automated Search-Based Functional Approximation for Digital Circuits. In: Reda S, Shafique M (eds) Approximate Circuits, Springer International Publishing, Cham, pp 175–203, DOI 10.1007/978-3-319-99322-5_9, URL http://link.springer.com/10.1007/978-3-319-99322-5_9

[73] Sidiroglou-Douskos S, Misailovic S, Hoffmann H, Rinard M (2011) Managing performance vs. accuracy trade-offs with loop perforation. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering - SIGSOFT/FSE '11, ACM Press, Szeged, Hungary, p 124, DOI 10.1145/2025113.2025133, URL http://dl.acm.org/citation.cfm?doid=2025113.2025133

[74] Tong D, Qu YR, Prasanna VK (2017) Accelerating Decision Tree Based Traffic Classification on FPGA and Multicore Platforms. IEEE Transactions on Parallel and Distributed Systems 28(11):3046–3059, DOI 10.1109/TPDS.2017.2714661, conference Name: IEEE Transactions on Parallel and Distributed Systems

[75] Tong JYF, Nagle D, Rutenbar RA (2000) Reducing power by optimizing the necessary precision/range of floating-point arithmetic. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 8(3):273–286, DOI 10.1109/92.845894, conference Name: IEEE Transactions on Very Large Scale Integration (VLSI) Systems

[76] Traiola M, Savino A, Barbareschi M, Carlo SD, Bosio A (2018) Predicting the Impact of Functional Approximation: from Component- to Application-Level. In: 2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS), pp 61–64, DOI 10.1109/IOLTS.2018.8474072, iSSN: 1942-9401

[77] Van Essen B, Macaraeg C, Gokhale M, Prenger R (2012) Accelerating a Random Forest Classifier: Multi-Core, GP-GPU, or FPGA? In: 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, pp 232–239, DOI 10.1109/FCCM.2012.47

[78] Vasicek Z (2019) Formal Methods for Exact Analysis of Approximate Circuits. IEEE Access 7:177309–177331, DOI 10.1109/ACCESS.2019.2958605, conference Name: IEEE Access

[79] Vasicek Z, Sekanina L (2015) Circuit Approximation Using Single- and Multi-objective Cartesian GP. European Conference on Genetic Programming 9025:229, DOI 10.1007/978-3-319-16501-1_18

[80] Venkataramani S, Ranjan A, Roy K, Raghunathan A (2014) AxNN: Energy-efficient neuromorphic systems using approximate computing. In: 2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pp 27–32, DOI 10.1145/2627369.2627613

[81] Venkataramani S, Raghunathan A, Liu J, Shoaib M (2015) Scalable-effort classifiers for energy-efficient machine learning. In: Proceedings of the 52nd Annual Design Automation Conference, ACM, San Francisco California, pp 1–6, DOI 10.1145/2744769.2744904, URL https://dl.acm.org/doi/10.1145/2744769.2744904

[82] Wang Z, Bovik A, Sheikh H, Simoncelli E (2004) Image Quality Assessment: From Error Visibility to Structural Similarity. IEEE Transactions on Image Processing 13(4):600–612, DOI 10.1109/TIP.2003.819861, URL http://ieeexplore.ieee.org/document/1284395/

[83] Wu X, Kumar V, Ross Quinlan J, Ghosh J, Yang Q, Motoda H, McLachlan GJ, Ng A, Liu B, Yu PS, Zhou ZH, Steinbach M, Hand DJ, Steinberg D (2008) Top 10 algorithms in data mining. Knowledge and Information Systems 14(1):1–37, DOI 10.1007/s10115-007-0114-2, URL https://doi.org/10.1007/s10115-007-0114-2

[84] Xu Q, Mytkowicz T, Kim NS (2016) Approximate Computing: A Survey. IEEE Design Test 33(1):8–22, DOI 10.1109/MDAT.2015.2505723, conference Name: IEEE Design Test

[85] Yang S (1991) Logic synthesis and optimization benchmarks user guide: version 3.0. Citeseer

[86] Yang Z, Jain A, Liang J, Han J, Lombardi F (2013) Approximate XOR/XNOR-based adders for inexact computing. In: 2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013), pp 690–693, DOI 10.1109/NANO.2013.6720793, iSSN: 1944-9399

[87] Yeh T, Faloutsos P, Ercegovac M, Patel S, Reinman G (2007) The Art of Deception: Adaptive Precision Reduction for Area Efficient Physics Acceleration.

In: 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007), pp 394–406, DOI 10.1109/MICRO.2007.9, iSSN: 2379-3155