IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 19, 2021, accepted June 4, 2021, date of publication June 9, 2021, date of current version June 23, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3087858

Multi-Objective Application-Driven Approximate Design Method

SALVATORE BARONE [1], (Graduate Student Member, IEEE),
MARCELLO TRAIOLA [2], (Member, IEEE),
MARIO BARBARESCHI [1], (Associate Member, IEEE),
AND ALBERTO BOSIO [2], (Member, IEEE)

[1]Department of Electrical Engineering and Information Technology, University of Naples Federico II, 80125 Naples, Italy
[2]CNRS, CPE Lyon, ECL, INSA Lyon, INL, University of Lyon, UCBL, 69130 Ecully, France

Corresponding author: Salvatore Barone (salvatore.barone@unina.it)

This work has been partially funded by the "IDEX Lyon OdeLe" project.

ABSTRACT Approximate Computing (AxC) paradigm aims at designing computing systems that can satisfy the rising performance demands and improve the energy efficiency. AxC exploits the gap between the level of accuracy required by the users, and the actual precision provided by the computing system, for achieving diverse optimizations. Various AxC techniques have been proposed so far in the literature at different abstraction levels from hardware to software. These techniques have been successfully utilized and combined to realize approximate implementations of applications in various domains (e.g. data analytic, scientific computing, multimedia and signal processing, and machine learning). Unfortunately, state-of-the-art approximation methodologies focus on a single abstraction level, such as combining elementary components (e.g., arithmetic operations) which are firstly approximated using component-level metrics and then combined to provide a good trade-off between efficiency and accuracy at the application level. This hinders the possibility for designers to explore different approximation opportunities, optimized for different applications and implementation targets. Therefore, we designed and implemented E-𝕀DE𝔸, an automatic framework that provides an *application-driven* approximation approach to find the best approximate versions of a given application targeting different implementations (i.e., hardware and software). E-𝕀DE𝔸 compounds 1) a source-to-source manipulation tool and 2) an evolutionary search engine to automatically realize approximate application variants and perform a Design-Space Exploration (DSE). The latter results in a set of non-dominate approximate solutions in terms of trade-off between accuracy and efficiency. Experimental results validate the effectiveness and the flexibility of the approach in generating optimized approximate implementations of different applications, by using different approximation techniques and different accuracy/error metrics and for different implementation targets.

INDEX TERMS Approximate computing, evolutionary algorithm, design space exploration, code mutation.

I. INTRODUCTION

Approximate Computing (AxC) term has been introduced to define a computing paradigm applied at different abstraction levels, spanning from hardware to software components [1]. Intuitively, instead of performing exact computation requiring a high amount of resources, AxC aims to carefully violate non-critical specifications, trading accuracy off for efficiency. For several real-world scenarios, the effectiveness of imprecise computation has been demonstrated in the literature, for both software and hardware components implementing inexact algorithms. Indeed, some applications show an inherent resiliency to errors [2], [3].

The inherent resiliency property tightly depends on the application domain. It can be observed for algorithms dealing with noisy real-world input data (e.g., image processing, sensor data processing, speech recognition, etc.), or with outputs that have to be interpreted by humans, such as digital signal processing of images or audio; also data analytic, web search and wireless communications exhibit an equivalent property [4]–[6]. Other involved applications are those that iteratively process large amounts of information, sample data, stop the convergence procedure early, or apply heuristics [7].

The associate editor coordinating the review of this manuscript and approving it for publication was Mauro Gaggero.

Diverse research articles in the literature explored the opportunities provided by AxC paradigm. Many of them proposed new methodologies to automatically define best trade-off configurations between result quality and performance. However, there are still open challenges holding AxC back from wider employment. In particular, the key point is the lack of a general and automatic design space exploration methodology to find the best trade-off between the degree of inaccuracy and the efficiency gain of an approximated application.

Most of the proposed techniques try to define new methods to generate alternative versions of specific component (either hardware or software) with less resources. For example, there are several proposals of approximate arithmetic operations [8]–[10]. Such variants differ from speculative implementations, because they do not focus on generating alternatives, rather on restoring the possible introduced error [11]–[13]. Other techniques generate variants by considering a high level description of the application or its implementation at low-level [4]. Moreover, existing approaches target only implementations at a specific level of the computing stack, i.e. either software or hardware.

In this work, we foster a unified methodology able to automatically explore the impact of different approximation techniques on a given application. The methodology does not need to specify which part(s) of the application should be approximated and how. It only requires the definition of the acceptable output degradation from the user. Moreover, it allows the user to define custom approximation techniques and describe the most suitable optimization objectives, according to the requirements.

Therefore, we developed Evolutionary-𝕀DE𝔸, or E-𝕀DE𝔸 (IIDEAA Is a Design Exploration tool for Approximate Algorithm), which is able to find approximate versions for an application expressed as C/C++ code, by mutating the original code. The output can be either a software or a hardware implementation of the approximated application, according to involved approximation technique(s).

E-𝕀DE𝔸 consists of two main components: (i) a source-to-source manipulation tool, the clang-Chimera, and (ii) an evolutionary search engine, the Bellerophon. Firstly, Clang-Chimera analyzes the Abstract Syntax Tree (AST) of the C/C++ code to apply approximation techniques. These techniques are described as *code mutators*, which allow introducing approximation by means of tunable parameters, and, as previously mentioned, new approximation techniques can be defined by implementing new code mutators. Secondly, Bellerophon is devoted to find the best approximation version of a given C/C++ code, according to user-defined optimization objectives. In other words, Bellerophon is confronted to a Multi-objective Optimization Problem (MOP). To face such a complex problem, Bellerophon gradually tunes the parameters of the mutators to obtain different approximate application codes. The evolutionary nature of the approach allows converging towards the Pareto-optimal solutions in terms of the defined objective functions, in a fairly reasonable time.

The remainder of the paper is structured as follows. In Section II we briefly report the state of the art about AxC design tools available in the literature. In Section III we present the global flow of the proposed approach, detailed in Subsection III-A (Clang-Chimera) and Subsection III-B (Bellerophon). Section IV illustrates experimental results. Finally, Section V draws the conclusion of this manuscript.

II. RELATED WORK

When dealing with the introduction of the approximation into an application, two challenging problems must be faced: (i) the selection of the portion of the application to approximate and (ii) the identification of the best combination of AxC techniques to use. The main goal is to obtain the best efficiency gain while satisfying the accuracy requirements of the final result. This paves the way to virtually infinite possibilities of approximation. However, most of the existing studies target a specific component at a given abstraction level rather than the full application. As an example, in [14] authors propose a circuit level model to systematically characterize different AxC circuits without considering the impact of the use of these circuits in a full application.

To the best of our knowledge, the literature lacks of an automatic approach for generating, exploring and selecting the best approximate versions of a given application. Indeed, existing methodologies allow the programmer (i.e., the user) to indicate what are the code lines or blocks to be approximated and how. For example, the programmer can annotate through compiler directives that a given loop has to be approximated by applying the loop perforation technique [15]. Moreover, existing approaches target a specific implementation, either software-level or hardware-level.

The established approach to validate the approximation effect (i.e., verify whether it satisfies the accuracy requirements) is the application profiling. After applying different combinations of AxC operators, the application is executed several times and the results are compared with those produced by a golden (i.e., approximation-free) implementation. The comparison is done exploiting metrics that suit the specific application domain (e.g., the structural similarity index in the image processing domain).

Early attempts dedicated to the approximation of the data precision are presented in [16]–[18]. The pioneering work in [18] illustrated the CMUFloat, a C++ library that redefines operators for integers, shorts and floating-point types in order to simulate reduced bit-width. Such a library allows simulating DSP units realized in an approximate manner. Anyway, one of the first attempt to define a systematic methodology for digital circuits approximation is [17], which introduced a power-aware design methodology by exploiting a word-length optimization. They proposed the *PowerCutter* tool to dynamically analyze the range of variables and reduce the precision of arithmetic operations for C/C++ code. Authors of [16] introduced Precimonious,

which provides an automatic tuning tool for IEEE 754 Floating Point standards. Indeed, Precimonious uses the Low Level Virtual Machine (LLVM) Intermediate Representation (IR) to perform a float-type exploration by means of annotations. This allows the user to specify the maximum loss in accuracy. Early approaches, however, were very unsophisticated and little automated, requiring several user interventions, including specifying which operations to approximate.

The state-of-the-art techniques working at software-level are ACCEPT [4] and its extension REACT [19]. ACCEPT is an adaptation for C/C++ of EnerJ [20], which has been designed to work within Java environment. ACCEPT introduces an extension of the C/C++ language to provide programming annotations. Such tools give to designers a program-guided technique for the inner-implemented approximation techniques. The implementation of ACCEPT involves the LLVM IR [21], providing a modified compiler to support the language extension. REACT uses a compiler-infrastructure to create a framework for the rapid exploration of well-known AxC techniques, proposing an energy model to evaluate the actual power saving for different approaches. REACT extends ACCEPT, so it exploits the LLVM IR to manipulate algorithms written in C/C++. Still on the use of reduced precision, recent GPU architectures introduced the opportunity to use floating-point types with different precision, with native support from the hardware. Indeed, the new *IEEE 754 half type* has been implemented within the most recent GPU architectures, aiming at enhancing the throughput and reduce the latency. However, development environments lack automatic precision tuning tools.

Concerning the hardware-level techniques, some approximate design approaches manually identify approximable sub-parts of circuits, mainly focusing on arithmetic components [22], [23] On the other hand, several other works attempt to define systematic and generic approaches. The paper by Nepal *et al.* introduced the ABACUS tool [24], which directly manipulates hardware circuits coded in Hardware Description Languages (HDL). It adopts a greedy algorithm to perform a design exploration to find solutions in terms of a trade-off between accuracy and power consumption. Another approach, proposed in [25], is able to take into account information about the underlying hardware architecture, such as the use of over-scaling techniques. Recently, authors of [26] proposed WOAxC, a workload-aware framework to automatically select pre-existing approximate units (adders and multipliers) minimizing energy consumption while meeting quality requirements of the application. Since quality requirements may change at run-time, authors of [27] propose an automated framework for the design of dynamically reconfigurable circuits. The framework modifies the circuit netlist by replacing the wires with *approximate switches* to reduce circuits' dynamic power consumption through a reduction of the switching-activity. Although, the added circuitry increases the static power consumption, a high reduction in total power consumption is achieved since the dynamic power is predominant in Application-Specific Integrated Circuit (ASIC) technologies considered by the authors. Unfortunately, the approach is tightly coupled with the target technology, hence it is not guaranteed that it could be used effectively on a different technology. A recent work claims approximate computing cannot be fully exploited by only considering hardware or software; therefore, hardware/software co-design should be considered in order to achieve better trade-off between accuracy and efficiency, in machine learning applications in particular [28].

All the aforementioned approaches either combine multiple design objectives in a single-objective optimization problem or optimize for a single parameter. Therefore, resulting solutions are centered around a few dominant design alternatives and do not cover the whole Pareto front [29]. Recent works addressed the circuit design problem by using multi-objective optimization to search for Pareto-optimal approximate circuit implementations. Such approaches focus on arithmetic components and image-processing operators, such as adders and multipliers. In [30], for instance, authors present a general-purpose method, based on multi-objective Cartesian genetic programming, for automating functional approximation of digital combinational circuits. Furthermore, many libraries consisting of thousands of elementary approximate circuits, which also supply hundreds of implementations of single arithmetic operations, have been proposed in the scientific literature [22], [31]. Authors of [32] addressed how to effectively combine approximate arithmetic circuits from libraries to design complex approximate accelerators. Their methodology aims at providing designs with a set of Pareto-optimal configurations (i.e., approximate arithmetic circuits to use) where the quality of results and hardware costs are both optimized. It is organized in three steps: (i) the target accelerator is profiled, in order to prune the library of approximate components discarding unsuitable circuits; (ii) machine-learning algorithms are employed to build predictors which enable the estimation of result quality and hardware cost; thus, the design space exploration does not require simulations nor hardware synthesis; (iii) the Pareto-front is iteratively constructed using a two-phases approach: first, predictors built in the second step are employed to quickly build a pseudo-Pareto front, hence refinement is needed towards the final Pareto front using quality and hardware estimation provided by actual simulations and hardware syntheses. In the above mentioned methodologies elementary components (i.e., arithmetic operations) are firstly approximated using component-level metrics and then combined to provide a good trade-off between efficiency and accuracy, according to application-level metrics. Anyway, such an approach do not provide the flexibility to use different approximation techniques.

Depending on the application needs and on the desired target (software or hardware) different approximations have different effects. Therefore, in this work, we propose a general methodology to explore the approximation opportunities at different levels (software and hardware) with an application-driven approach. The methodology itself is

completely independent of the final implementation and technology target. However, it takes them into account by means of custom models. Specifically, we resort to (i) a model of the target application (i.e., C/C++), (ii) a model of the approximation techniques (i.e., code mutators), and (iii) a model of the application accuracy requirement (i.e., error metrics). An early version of the proposed framework, 𝕀DEA, was presented in [33] and was applied only at hardware-level. Moreover, at that time, the search engine was not based on evolutionary algorithms. In fact, it was based on the Branch & Bound (B&B) algorithm. The latter is a well-known generic method to find an optimal solution of single-objective optimization problems. Therefore, this approach was not suitable to deal with MOPs, such as the ones that we face in this work. Moreover, B&B could not scale with the complexity of the input application. Finally, in [33] only results for the bit-width precision reduction approximate operator were presented.

Compared to state-of-the-art, we can list the main contributions of this paper as follows:

- we propose an application-driven approximation process (i.e., the approximation is performed based on application-level metrics);
- the methodology is compatible with any functional approximation technique;
- both software and hardware implementations of the approximate application can be produced;
- no profound knowledge of the application code is needed;
- the proposed framework is completely open-source [34] and extensible.

III. PROPOSED APPROACH

Existing AxC design tools consider specific transformations and specific domains, as discussed in Section II. Moreover, they are not fully automatic and simply provide a guided approach for approximation. Conversely, we aim to define an **automatic and general approach**. Figure 1 sketches the overall flow of the proposed approach, namely E-𝕀DEA. It requires the following inputs:

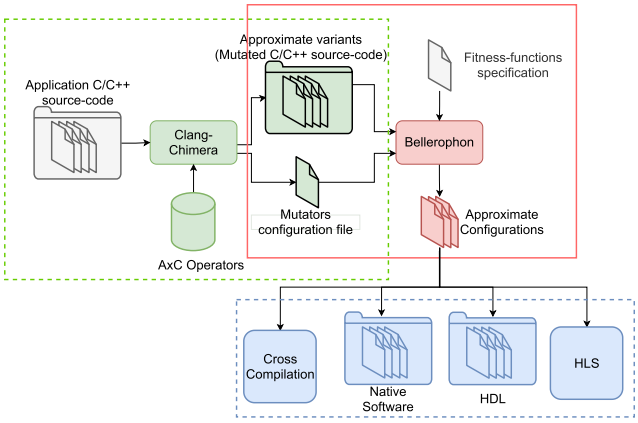


FIGURE 1. E-𝕀DEA flow, which includes Clang-Chimera and Bellerophon tool.

1) the *original application* described as C/C++ code;
2) the set of *approximate operator*, called ''mutators'';
3) the *fitness-functions* to select the appropriate approximation outcomes; in particular, the user must specify (i) the *error metric* to quantify the approximate version deviation compared to the original outcomes and the related *error threshold* in order to determine whether the approximate outcomes are acceptable or not, and, optionally, (ii) the *gain metrics* required to estimate gains achieved through the approximation.

Concerning mutators, E-𝕀DEA is already provided with a set of mutators implementing the most common approximation techniques.

As main output, E-𝕀DEA produces the mutator configuration of the best approximate application variants obtained (i.e., the non-dominated solutions). Such configurations are then used to produce software or hardware implementations, possibly involving also High Level Synthesis (HLS) tools.

Below, we report a few remarks on the requirements that drove E-𝕀DEA development.

- *E-𝕀DEA is general*: the implementation of any approximate computing technique is possible through code-mutation;
- *E-𝕀DEA is target independent*: it produces configurations applicable at any level of the computing stack;
- *E-𝕀DEA is application independent*: it allows the user to define custom error/reward functions to evaluate application approximation variants;
- *E-𝕀DEA is extensible*: it is an ensemble of tools written in C++ that can be easily extended and integrated;
- *E-𝕀DEA is free and open-source*: it is released under the GNU Affero General Public License and its code is publicly available [34].

As already mentioned, E-𝕀DEA is composed of two phases carried out by two components, Clang-Chimera and Bellerophon, respectively described in Subsections III-A and III-B.

A. Clang-Chimera

The green dotted box in Figure 1 reports Clang-Chimera flow. Clang-Chimera is a mutation engine for C/C++ code. It is based on the Clang compiler [**?**], 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, in order to make systematic modifications to the latter, and produce a set of mutated files, i.e. the configurable approximate variants of the input application code. In addition, it also provides the *mutators configuration* file, which will be subsequently used during the DSE.

Clang-Chimera borrows the terminology from the *mutation testing* technique, which is a software testing approach used to evaluate the quality of a test set, in terms of its ability to detect software faults [35]. Mutation technique consists in using supporting tools to mutate the original source code – thus emulating programmer mistakes – to generate erroneous

programs. The test set quality is evaluated according to the number of mutated versions detected.

In the context of Clang-Chimera, the mutation is used in order to generate modified (i.e., approximate) versions of the original code. Here, the concept of software fault is replaced by the concept of approximation technique. The latter is formalized as a *mutator*.

$$Mu = \{Match, Mutate\} \quad (1)$$

Eq. 1 defines the mutator Mu as a duplet where:

- **Match** identifies where in the code the mutation has to take place. This is defined by means of matching rules;
- **Mutate** indicates how to actually modify the source code. This is defined by means of mutation rules.

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 results in introducing constructs through which it is possible to tune the approximation degree.

Clang-Chimera utilizes LLVM/Clang facilities, such as *ASTMatcher* and *Rewriter* in order to apply a given mutator Mu_i. In details, *ASTMatcher* searches for all occurrences of the "Mu_i.*Match*" rule and *Rewriter* modifies the identified nodes by applying the "Mu_i.*Mutate*" rule.

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; (iv) a mutator supporting approximate arithmetic operator models of circuits being part of the EvoApproxLib [31] and EvoApproxLib-Lite [36] libraries. Moreover, adding new mutators to this set allows the IDEA framework to be easily extended. Next subsection presents some examples.

1) LOOP PERFORATION EXAMPLE

The goal of this example is to illustrate the application of the well know *loop perforation* technique to the C/C++code reported in listing 1.

The related AST is reported in Figure 2.

Basically, we want the loops to skip some iterations. This can be obtained by altering the stride of the loop variable(s). Therefore, let the Mutator related to the Loop Perforation (LP) be defined as follows:

$$Mu_{LP} = \{FOR\_STMT, var{+}{+} \rightarrow var{+}{=}stride_i\} \quad (2)$$

The application of the mutation described in Eq.2 modifies the AST (see Figure 2). For each occurrences of *FOR_STMT*, the corresponding *FOR_EXPR* node is altered. The resulting

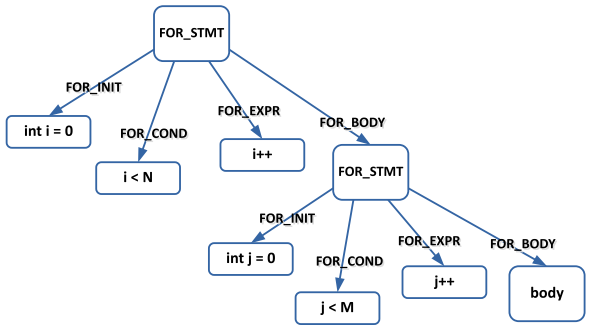


FIGURE 2. For loop AST example (see Listing1).

```
int main (void) {
  for (int i = 0; i < N; i++) {
      for (int j = 0; j < M; j++) {
          body;
      }
  }
}
```

Listing 1. Precise code

```
int main (void) {
  for (int i = 0; i < N; i+=stride1) {
      for (int j = 0; j < M; j+=stride2) {
          body;
      }
  }
}
```

Listing 2. Mutated code

mutated code, produced by Clang-Chimera, is reported in the following listing.

Finally, the loop perforation effects depend on the actual values of $stride_1$ and $stride_2$ variables. Indeed, by assigning different values to the variables, different trade-offs between skipped iterations (thus performance increase) and accuracy reduction can be obtained.

2) APPROXIMATE CIRCUIT EXAMPLE

With this example we apply mutators to alter an algorithm, such that in Listing 4, to be approximated by replacing every sum operation with a configurable approximated sum. The Clang-Chimera allows the automatic generation of an approximate variant having one, or more, sum operations replaced with a call to a function that implements the approximate sum. Let us suppose that the function performing the approximate sum is the one reported in Listing 3, with *add1* and *add2* being the addends and *ax* being the configurable parameters governing the approximation. The exact nature of the approximation being made by this function is hidden in the function itself: whether the function implements bit-width reduction, the latter parameter may govern the Number of Approximate Bit (NAB) of the sum; conversely, whether the function implements a library of approximate circuits, the *ax* parameter may allow to select which particular circuit is to be adopted. Anyway, from the Clang-Chimerapoint of view, the particular approximate technique being implemented does not matter: using the appropriate mutator, the Clang-Chimera mutates the code in Listing 4 and generates the code in Listing 5.

```
int ax_sum(int add1, int add2, int ax);
```

Listing 3. Example of approximate sum

```
...
y = x + 2
z = 2 * x + 3 * y + 2;
...
```

Listing 4. Example code to be mutated

```
int ax_0 = 0;
int ax_1 = 0;
int ax_2 = 0;
...
y = ax_sum(x, 2, ax_0);
z = ax_sum(ax_sum(2 * x, 3 * y, ax_1), 2, ax_2);
```

Listing 5. Mutated code

As for the loop-perforation technique discussed above, the effects of approximation depend on the actual value of configuration parameters. Hence, the main problem is to find an appropriate value for these parameters, in order to achieve the best trade-off between performance gains and accuracy losses.

As described in the next section, the main goal of Bellerophon is tuning these values to ultimately find non-dominated solutions in terms of trade-off between performance gain and accuracy loss.

B. Bellerophon

Red continuous box in Figure 1 depicts the Bellerophon flow. The tool analyzes the set of mutated files generated by Clang-Chimera and explores the different possible mutators configurations, i.e. different configurations for the tunable parameters in mutators, which results in different fitness values. The final result is a set of solutions corresponding to the (sub-)optimal trade-offs between the user-defined objective functions.

1) TECHNICAL BACKBROUND

More formally, Bellerophon faces a MOP, which, as described in [37], can be defined as the problem of finding, for some *decision variables*, a set of values satisfying imposed constraints. At the same time, those values have to optimize (minimize/maximize) a set of *objective functions*. Those functions describe criteria in conflict with each other. In MOPs, we deal with finding compromises between the objectives rather than a single solution as in global optimization. Therefore, Bellerophon explores the different approximate variants while 'moving' towards the *Pareto front* of the solutions, in terms of the defined objective functions. The Pareto front is defined as the set of solutions for which it does not exist any other solution improving some objective without simultaneously deteriorating at least one other objective. Since finding exact solutions for MOPs is complex – and often computationally infeasible – the scientific community usually resorts to Evolutionary Algorithms (EAs), which provide satisfactory solutions in reasonable time.

EAs mimic the genetics by performing *optimization by evolution*, i.e. the best fitting individuals survive [38]. Indeed, EAs manage a group of solutions – called a *population* – and optimize them in parallel. Every solution in a population is called an *individual*. Every individual is represented by its *chromosome*, composed of *genes* (i.e., the parameters of the approximation techniques, in our context). A *fitness value* is assigned to each chromosome, according to the fitness functions (i.e., objectives). The basic principle is that individuals with best fitness values will survive. Individuals are mutated – to mimic genetic gene changes – until only the best population remains.

In particular, Genetic Algorithms (GAs), a subclass of EAs inspired by evolutionary theory, exploits *mutation, crossover* and *selection* concepts to cause the extinction – through generations – of weak and unfit species. Strong species have greater chances to survive and thrive by spreading their genes to future generations. A mutation entails a random change in genes. If it provides additional advantages to the specie, it will propagate to next generation. Conversely, disadvantageous changes are eliminated by selection. The crossover phenomenon combines two chromosomes – referred to as *parents* – to form a new one – referred to as *offspring*. The synergy of mutation, crossover, and selection makes good genes thrive through generations. In this way, starting from a randomly generated initial population, GAs make it evolve towards a set of non-dominated solutions which are as close as possible to the Pareto dominant ones.

2) MOP MODELING

Bellerophonmodels our problem as follows:

1) **Population:** each solution (i.e., approximate version) represents an individual of the population;
2) **Chromosomes:** each individual has a chromosome, modeled as an array of integers. Each value (i.e., gene) corresponds to an approximation parameter that can be *mutated* through generations;
3) **Fitness:** a single, or multiple user-defined and fitness-functions are employed to *select* the best individuals;
4) **Variation Operations:** the mutation and crossover operations randomly alter genes, or combines *parents'* genes to generate an *offspring*.

Fitness functions might be defined accordingly to the particular exploited AxC technique. In case of precision-scaling technique, for instance, a feasible fitness function could be based on the number of neglected bits, since it translates in less hardware resources.

To perform the exploration, Bellerophon creates new populations by tuning the approximation parameters (i.e., mutator parameters in the code) and evaluates the corresponding approximate variants according to the fitness functions, to finally select the best set. The *mutators configuration* file (see Figure 1) reports the maximum and minimum possible values for all approximation parameters. The main

```
1 stride1 , N, 1
2 stride2 , M, 1
```

Listing 6. Mutators configuration file example

objective of Bellerophon is to converge toward optimal solutions improving fitness-functions as much as possible.

In order to perform the exploration, we resorted to ParadisEO [39], a software framework for meta-heuristics for MOP. In particular, we resorted to the state-of-the-art GA, the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [40].

In order to be evaluated, each individual has to be compiled and executed. To speed-up the execution time, the compilation strategy adopted by Bellerophon allows to compile just what it is necessary to retrieve information about approximate variants. Bellerophon uses the Just-in-Time engine provided by clang LLVM [41]: each time the software needs to be altered to test a new variant, Bellerophon do not invoke the system loader, rather it alters the program image which is already loaded into the memory.

The next subsection will show an example of how Bellerophon creates a population and evaluates the fitness of its individuals to then select the best set.

3) EVOLUTION EXAMPLE

Let us refer again to the example in subsection III-A1. Clang-Chimera applied the mutations to the AST and, consequently, it generated the C/C++ code shown in listing 2 and provided the ''mutators configuration'' file. In this example, a Mutator (i.e., approximation technique) modeling the *loop perforation* has been used. The technique has been applied to both the loops in the code (lines 2 and 3 of the code). The *mutators configuration file* produced by Clang-Chimera reports the maximum and minimum values for the two stride variable introduced (i.e., $stride_1$ and $stride_2$). The following listing reports the *mutators configuration file* generated by Clang-Chimera for this example.

TABLE 1. Examples of a population made of *k* individuals.

Individual	Stride1	Stride2	Error	Reward
0	3	5	5	8
1	2	9	7	11
⋮	⋱		⋮	⋮
k-1	5	2	3	7

Thanks to mutation, crossover and selection operations, Bellerophon moves towards the Pareto-front. The first population is generated randomly. Within the limits imposed by the configuration file, Bellerophon is able to mutate the individuals and generate a non-dominant population as reported in Table 1. In this example, *k* individuals have been created. Each individual is characterized by its own chromosomes (i.e. value of the stride variables), reported in the table rows. Bellerophon evaluates the individuals according to the fitness functions and assigns corresponding values to each of them. For example, here the reward is simply defined as the sum of the stride variable values: the higher the stride value the more iterations will be skipped. However, skipping loop cycles also entails an accuracy loss. Depending on the fitness values measured for each individual, Bellerophon will select the set of best candidates to generate the next population.

IV. EXPERIMENTAL RESULTS

In this section, we report experimental results achieved with the proposed approach. The main goal of the experiments is to prove the effectiveness and the flexibility of the proposed approach in terms of target application, AxC techniques and implementation level. We thus applied E-𝕀DE𝔸 to different applications and target implementations. In particular, we show case studies of different complexities and targeting hardware accelerator and software implementations. Application source codes belong to the AxBench benchmark suite for AxC [42]. The goal of the experiments is to show the versatility and the efficiency of the proposed approach. To this end, we performed the experiments by using various error metrics and rewards, as well as diversified approximation techniques. For what pertains to scalability and sufficiency of the DSE, the E-𝕀DE𝔸framework addresses them both by allowing the user to set (i) the number of iterations, (ii) the size of the initial population, (iii) the technique to define the initial population, (iv) mutation and crossover probabilities, and (v) the amount of genes involved in crossover. Moreover, the framework allows to retry experimental campaigns each time results belongs to an uncrowded Pareto-front or they are too close to few minimum spots. Finally, the E-𝕀DE𝔸 framework is intended to be employed on common hardware. In order to run experiments reported below, we employed a common personal computer based on Intel I7 8550@1.8GHz, equipped with 16 GB of RAM and running Linux 4.9.

A. K-MEANS CLUSTERING SOFTWARE IMPLEMENTATION

As first case study, we target a software implementation of the K-means clustering algorithm, applied to image processing. This algorithm is used in computer vision for *image segmentation*. The goal of image segmentation is to process a digital image and divide it into multiple segments, i.e., sets of pixels. The segmentation helps in simplifying the image for analysis purpose [43]. Typical utilization are *object location* and *boundary recognition*. We executed this case study on a generic CPU.

1) SETUP

As approximation technique, we used the *loop perforation*, as illustrated in Subsection III-A1. The technique is applied on loops performing centroid computation; thus, skipping an iteration coincide to ignoring some points to be clustered. We instrumented Bellerophon to maximize the well-known Structural SIMilarity (SSIM) index [44], which corresponds to minimize the error entailed by the approximation. The SSIM is widely used in the image processing community. SSIM is a model based on the perception. It considers the degradation of the image as observed variation in structure,

luminance and contrast. The SSIM index is a decimal value between −1 and 1. The value 1 represents perfect structural similarity. A value of 0 indicates no structural similarity between the two input data sets. We evaluated the error entailed by the approximation over four 512 × 512 images. As for the reward function, we used the number of skipped loops in the approximate versions of the code.

2) RESULTS

In Figure 3, we report the percentage of skipped loops (y-axis) and the corresponding SSIM index (x-axis) for the non-dominated solutions found with E-𝕀DE𝔸. Moreover, in Figure 4, we report two examples of images produced with the approximate variants having the highest and the lowest SSIM. Figure 4a and 4b depict the input and output images to the k-means algorithm, for its precise version (i.e., without approximation). Figure 4c and 4d report the images obtained with two of the approximate variants, with the highest and lowest SSIM, respectively. As the images clearly show, the proposed approach allowed us to find approximate variants of the code leading to skip from 16% up to 29% loops, without significantly impacting the output quality. Indeed, in the best case, the achieved SSIM index was 0.99998; in the worst case, we obtained a SSIM of 0.99988. E-𝕀DE𝔸 found the reported set of approximate variants in about 90 minutes.

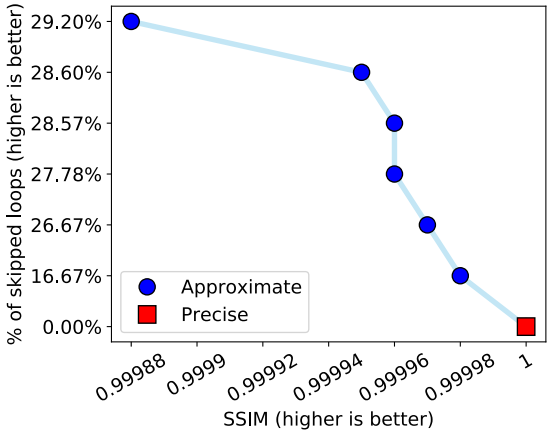


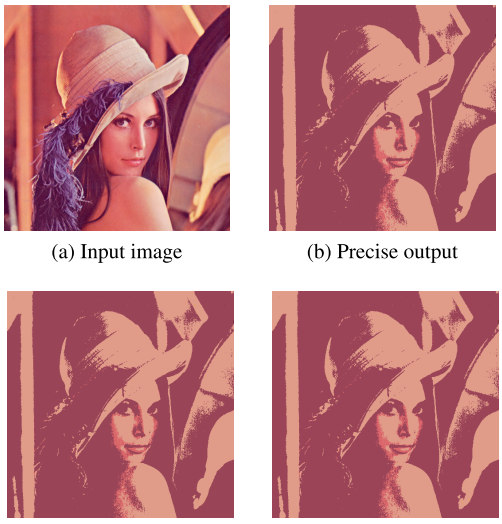
FIGURE 3. Non-dominated solutions found with E-𝕀DE𝔸: approximate variants for the K-means clustering algorithm, obtained by applying the loop perforation approximation technique.

B. TAYLOR SERIES SOFTWARE IMPLEMENTATION

The second targeted case study is a software implementation of the Taylor series expansion of the logarithmic function, expressed as follows:

$$f(x) = \ln(1+x) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n} = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots \tag{3}$$

Taylor series in general are largely used in computer graphics, in linearization problems (e.g., in Robotics), and in engineering and computational sciences in general. In this case, we applied two different versions of the same approximation technique and used a different error metric, better suited to the case study. We targeted a software implementation.



(a) Input image (b) Precise output

(c) Approximate variant; SSIM = 0.99998, skipped loops = 16.67%

(d) Approximate variant; SSIM = 0.99988, skipped loops = 29.2%

FIGURE 4. Example of output of two approximate variants of the K-means clustering algorithm, obtained by applying loop perforation technique.

```
int main (void) {
  for (int i = 0; i < N; i++) {
      if (i % skip1 == 0){
          for (int j = 0; j < M; j++) {
              if (j % skip2 == 0){
                  body;
              }
          }
      }
  }
}
```

Listing 7. Example of mutated code with the loop-second variant

1) SETUP

As approximation technique, we adopted the *loop perforation* so that skipping the i-th iteration translates in skipping the computation of the i-th coefficient of the series. We instrumented Clang-Chimera to use two different loop perforation variants:

1) The one illustrated in Subsection III-A1, that we refer to as *loop-first*;
2) A technique, that we refer to as *loop-second*, which executes only loops having an index multiple of a certain value *'skip'* [45]. For instance, by using this variant, the mutated version of the example code shown in Subsection III-A1 is the following:

Concerning Bellerophon instrumentation, as error metric we employed the Maximum Absolute Error (MAE), calculated as follows:

$$\text{MAE} = \max_{(x)} \left| f(x) - f_{approx}(x) \right| \tag{4}$$

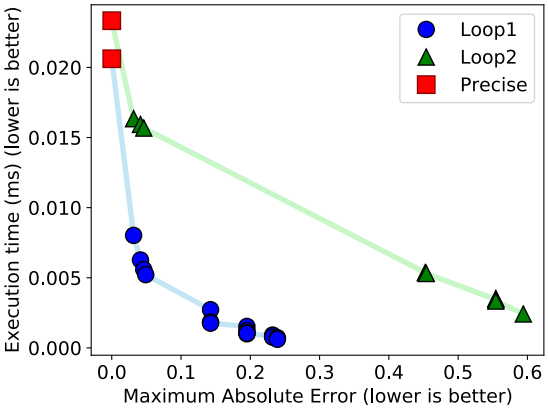


FIGURE 5. Non-dominated solutions found with E-𝕀DE𝔸: approximate variants for the Taylor Series expansion, obtained by applying both loop perforation approximation techniques.

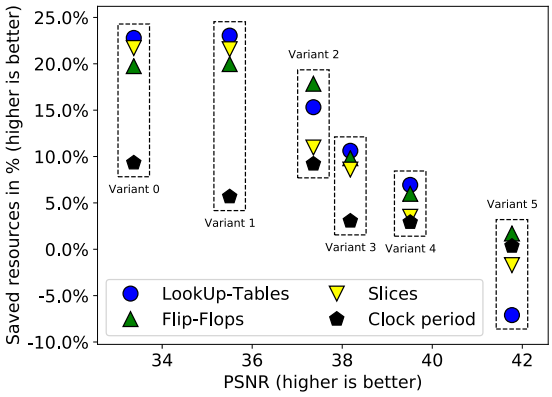


FIGURE 6. Non-dominated solutions found with E-𝕀DE𝔸: approximate variants for the JPEG benchmark, obtained by applying bit-width reduction approximation technique to the DCT algorithm.

We evaluated the error entailed by the approximate variants over 1,000 different x values and by calculating the Taylor expansion up to the order of 100 ($n \in [1, 100]$ in Equation 3). The error threshold was set to 0.6. As for the reward function, we used the inverse of the *execution time* in milliseconds, since we target a software implementation.

2) RESULTS

In Figure 5, we report the non-dominated solutions found with E-𝕀DE𝔸 for both variants of the loop perforation technique. We report the trade-off between the execution time (y-axis) and the MAE entailed by the approximate versions of the code (x-axis). The blue circles represent results obtained with the loop-first (Loop1) variant and the green triangles those obtained with the loop-second (Loop2) variant. Finally, red squares represent the precise variants of the program. The execution time needed by E-𝕀DE𝔸 to find the two sets of approximate variants was of few seconds for the loop-first variant and of about three minutes for the loop-second variant.

Hence, the loop-first variant clearly shows solutions dominating the loop-second variant, for this benchmark.

C. JPEG COMPRESSION ALGORITHM

In this case study, we targeted both a hardware accelerator implementation and a software implementation of the well-known JPEG compression algorithm. In particular, we addressed the approximation of the Discrete Cosine Transformation (DCT) employed by the JPEG algorithm. We applied two different approximation techniques and metrics suitable to the two different target technology implementations (i.e. custom hardware accelerator and software implementation).

1) BIT-WIDTH REDUCTION

a: SETUP

We configured Clang-Chimera to generate mutants by altering the bit-width of numeric variables involved in the DCT function. Furthermore, we configured the Bellerophon objective functions as follows. The accuracy loss estimation was computed by using the Peak Signal-to-Roise Ratio (PSNR). We calculated the PSNR between the image produced by the precise (i.e., non-approximate) JPEG and the output image of the approximate versions to measure the degradation. For each version, we applied this measure to four different pictures and used the maximum as reference. The maximum allowed value (i.e., error threshold) was set to 33 dB. Typically, acceptable PSNR values in lossy 8-bit image compression span between 30 and 50 dB (higher is better) [46]. The reward was measured as the total number of neglected bits, since we aimed to reduce the resources to implement a hardware custom accelerator. To perform the experiment, we adopted the FLAP library. This library allows replacing standard numeric variable types (e.g., double, int, ...) with custom-precision variables. Details about FLAP library can be found in [47]. As output, E-𝕀DE𝔸 produced a set of approximate variants of the original application. In details, it generated a set of configurations for the custom-precision numerical variables within the DCT. These are the non-dominated solutions in terms of trade-off between accuracy loss and gain of hardware resources. We processed the obtained configurations with an HLS engine to generate a hardware implementation of the DCT. In particular, for this experiment, we used Xilinx Vivado HLS [48].

b: RESULTS

Figure 6 depicts the obtained experimental results. The approximate variants were synthesized to hardware accelerators implemented on a Xilinx Zynq-7010 FPGA board. We report the percentage of hardware resources saved by the obtained variants w.r.t. the synthesis of the original precise version. In particular, for each approximate variant, we report savings (in percentage) in term of Look-Up Tables (LUT), Flip-Flops (FF), slices and clock period. For instance, the first variant saves 22% of Look-Up Tables (LUTs), 19% of FF 21% of slices and the clock period is 9% shorter compared to the original precise version. The PSNR for this variant

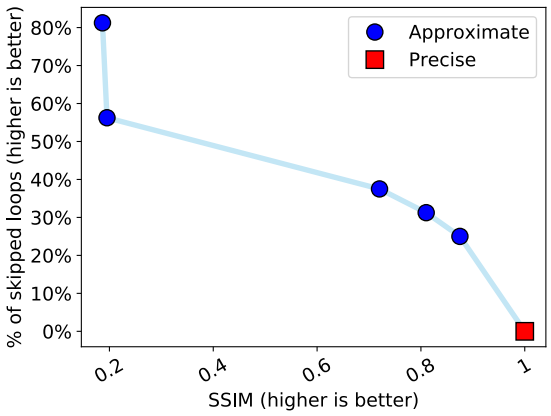


FIGURE 7. Non-dominated solutions found with E-𝕀DE𝔸: approximate variants for the JPEG benchmark, obtained by applying loop perforation technique to the DCT algorithm.

is 33 dB. When negative values appear for saved resources, the related variant requires more hardware resources than the original precise version. This can happen when the HLS engine is not able to perform as much hardware optimizations as in the full-precise synthesis. This kind of problem is also described in [25] and it is not due to E-𝕀DE𝔸, rather to the adopted AxC operator. Since the PSNR value of the precise version image against itself would be ∞, it is not reported in the figure.

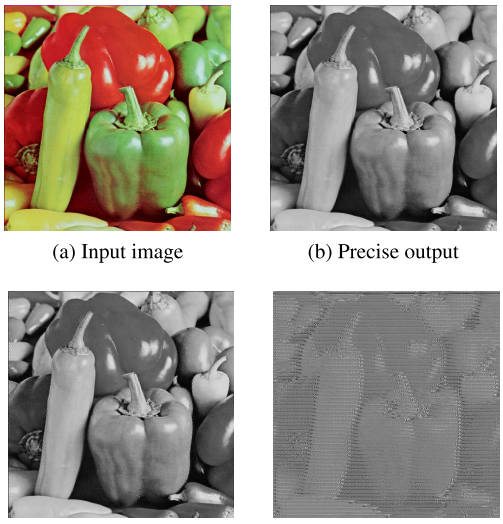
2) LOOP PERFORATION

a: SETUP

We performed the same experiment also by configuring Clang-Chimera to generate mutants using the loop perforation as approximation technique, (see Subsection III-A1). The technique is applied to loops that compute the DCT, and the resulting effect is that some of the terms of the transform are not computed. Hence, we instrumented Bellerophon to use (i) the SSIM index [44] to measure the accuracy loss of the final approximate images w.r.t. to the precise one and (ii) the percentage of skipped loops as measure of the gain, since we aimed to reduce the execution time of a software implementation. Once again, E-𝕀DE𝔸 produced a set of non-dominated approximate variants of the original application, i.e. a set of configurations for the loop perforation to obtain a good trade-off between accuracy loss and performance gain.

b: RESULTS

In Figure 7, we report the percentage of skipped loops (y-axis) and the corresponding SSIM index (x-axis) for the non-dominated solutions found with E-𝕀DE𝔸. The precise version is the reference and is placed in (SSIM = 1, skipped loops = 0). Moreover, in Figure 8, we report two examples of images produced with the approximate variants having the highest and the lowest SSIM. Figure 8a and 8b depict the input and output images of the precise (i.e. non approximate) JPEG compression algorithm. Figure 8c and 8d report the



(a) Input image (b) Precise output

(c) Approximate variant; SSIM = 0.87, skipped loops = 25%

(d) Approximate variant; SSIM = 0.18, skipped loops = 81.25%

FIGURE 8. Example of output of two approximate variants of the JPEG algorithm, obtained by applying loop perforation technique to the DCT algorithm.

images obtained using approximate variants at the borders of the Pareto-front, i.e. those exhibiting the minimum error and the minimum amount of retained loops. These variants exhibit, with the SSIM values equal to 0.87 and 0.18, respectively, having skipped 25% and 81% during the computation respectively. E-𝕀DE𝔸 found the reported set of approximate variants in about twenty minutes.

D. SOBEL EDGE-DETECTION FILTER

In this case study, we targeted the Sobel filter. This filter is usually employed in image processing and computer vision applications. Also for this benchmark we applied different approximation techniques and metrics, targeting different technology implementations (i.e. custom hardware accelerator and software implementation).

1) BIT-WIDTH REDUCTION

a: SETUP

Also for this case study, we instrumented Clang-Chimera to alter the bit-width of numeric variables involved in additions and multiplications within the Sober filter code. Bellerophon was configured as for the first case study: maximum PSNR as error metric among 4 images, 33 dB as error threshold and total number of neglected bits as reward function, since we aimed to reduce the resources to implement a hardware custom accelerator. The FLAP library was used to achieve the bit-width reduction. Finally, we processed the obtained solutions with the HLS engine.

b: RESULTS

In Figure 9 we depict the obtained non-dominated solutions achieved by E-𝕀DE𝔸 for the Sobel filter case study. As shown in the figure, for this case study E-𝕀DE𝔸 did not suggest

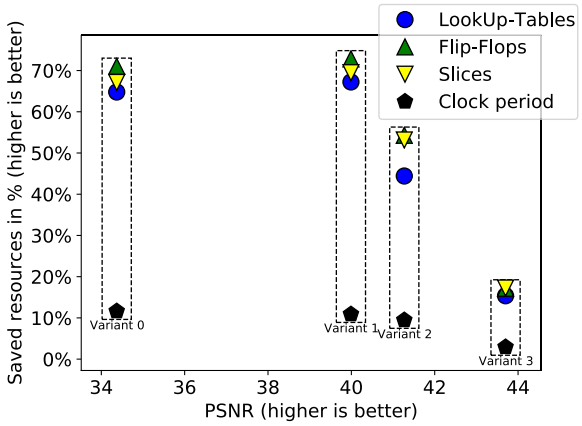


FIGURE 9. Non-dominated solutions found with E-𝕀DE𝔸: approximate variants for the Sobel edge-detection benchmark, obtained by applying bit-width reduction approximation technique.

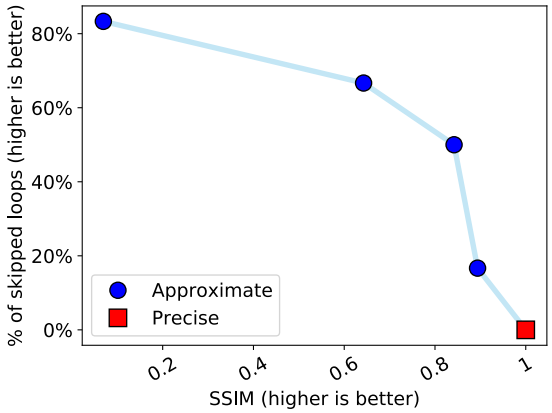


FIGURE 10. Non-dominated solutions found with E-𝕀DE𝔸: approximate variants for the Sobel edge-detection algorithm, obtained by applying loop perforation approximation technique.

any solutions with hardware overhead. Moreover, the variant 1 entailed a PSNR of 39.99 dB, while saving more than 65% of hardware resources and increasing the performance (i.e. lower clock-period) by 10%. E-𝕀DE𝔸 found the reported set of approximate variants in about fifty seven minutes.

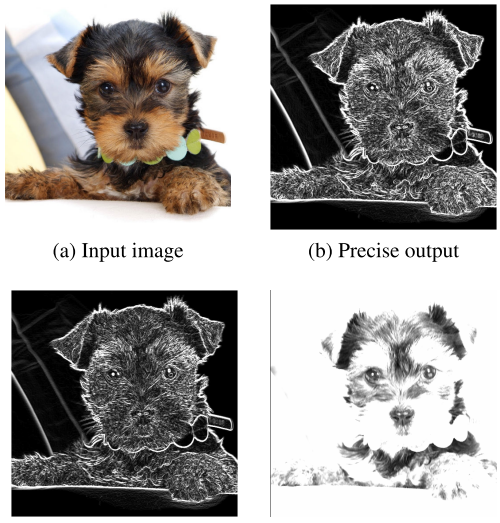
2) LOOP PERFORATION

a: SETUP

As done for the previous case study, we executed the same experiment by applying the loop perforation technique to the convolution algorithm, which results in skipping a few multiplications, according to the approximation degree. As fitting functions we used again the SSIM to measure the accuracy loss and the percentage of skipped loops as gain metric, since we aimed to reduce the execution time of a software implementation.

b: RESULTS

In Figure 10, we report the percentage of skipped loops (y-axis) and the corresponding SSIM index (x-axis) for the non-dominated solutions found with E-𝕀DE𝔸. Moreover, in Figure 11, we report two examples of images produced with the approximate variants having the highest and the lowest SSIM. Figure 11a and 11b depict the input and output images to the sobel filter edge-detection algorithm, for its precise version (i.e., without approximation). Figure 11c and 11d report the images obtained approximate variants at the borders of the Pareto-front, i.e. those exhibiting the minumum error and the minimum amount of retained loops. These variants exhibit SSIM values equal to 0.89 and 0.06, and 16% and 83% skipped loops during the computation, respectively. Also for this case study, E-𝕀DE𝔸 found the reported set of approximate variants in about twenty minutes.



(a) Input image

(b) Precise output

(c) Approximate variant; SSIM = 0.89, skipped loops = 16.66%

(d) Approximate variant; SSIM = 0.06, skipped loops = 83.33%

FIGURE 11. Example of output of two approximate variants of the sobel edge-detection algorithm, obtained by applying loop perforation technique.

3) APPROXIMATE CIRCUITS

a: SETUP

Aiming at reducing costs of a hardware accelerator for vertical edge detection, both in terms of silicon area and power consumption, in this experiment we configured the Clang-Chimera tool to replace exact arithmetic operators using approximate ones, by adopting implementations from the EvoApproxLib library of approximate circuits [31]. Thus, the Clang-Chimera tool generates an approximate version of the considered application in which it is possible to select, for each addition, an implementation between either the exact or an approximate implementation from the mentioned library.

Concerning optimization, during this experiment we considered three different fitness-functions, i.e. error minimization, silicon area minimization, and power consumption minimization.

We selected the PSNR as error metric, and, during error assessment, the PSNR is computed by considering

images resulting from the exact and approximate Sobel filter while resorting to a comprehensive data set [49] consisting of 44 different images. Circuit area and power consumption are estimated as the sum of the contributions of each single approximate circuit, as reported in [31].

Please, kindly note that, for this particular application, an exhaustive evaluation of the whole set of approximate configurations is computationally feasible, since the moderate amount of operations required by the filter. Indeed, vertical edge detection requires only five additions and two doubles, with the latter being implementable using wire-only left-shift, which nullifies hardware costs of multiplications. The total amount of approximate configurations is about 4.9×10^7. This gives the opportunity to compare the actual Pareto-front resulting from the exhaustive evaluation with the estimation provided by the E-𝕀DE𝔸 framework.

In order to also evaluate how the quality of the Pareto-front estimation is affected by the GA configuration parameters, we performed three different runs of the Bellerophon tool, varying the effort for the DSE phase while keeping mutation and crossover probability unmodified. In particular, (i) for the low-effort run, we considered a population of 500 individuals and 3 iterations, (ii) for the medium-effort run, we considered a population of 2000 individuals and 11 iterations, and (iii) for the high-effort run, we considered a population of 20000 individuals and 100 iterations. Table 2 summarizes the experimental setup.

TABLE 2. DSE parameters and relative results for the Sobel vertical edge detector case study. Note that the normalized distances is computed from fitness-function values normalized to [0, 1].

Effort	Pop.	Iter.	Time	Absolute Distance			Normalized Distance		
				Min.	Avg	Max	Min	Avg	Max
Exh.	-	-	≈170h	-	-	-	-	-	-
Low	500	3	≈5min	0.013	1.58	7.6	5.9e-6	2.4e-4	1.7e-3
Med.	2000	11	≈4h	0.002	1.57	5.8	3.7e-6	6.9e-6	5.6e-4
Hig.	20000	100	≈22h	0.001	1.5	4.4	3.6e-6	6.8e-6	5.4e-4

b: RESULTS

For comparison purposes, Figure 12 and Figure 13 report, respectively, experimental results in the "PSNR vs. area" and "PSNR vs. power" perspective. Furthermore, in order to measure how close configurations P provided by E-𝕀DE𝔸 are w.r.t. the optimal configurations Q resulting from exhaustive evaluation, we measure the distance from each point $p \in P$ to the nearest optimal configuration $q \in Q$, as reported in Equation (5). The minimum, average and maximum distance are reported in Table 2, along with normalized distances, as done in [32].

$$d_p = \min_{\forall q \in Q} |q - p| \quad \forall p \in P \tag{5}$$

As the reader can figure out, results from E-𝕀DE𝔸 are very close to those resulting exhaustive simulation, while, as reported in Table 2, the amount of time needed by exhaustive evaluation is prohibitively higher than that required by the high-effort run of the GA.

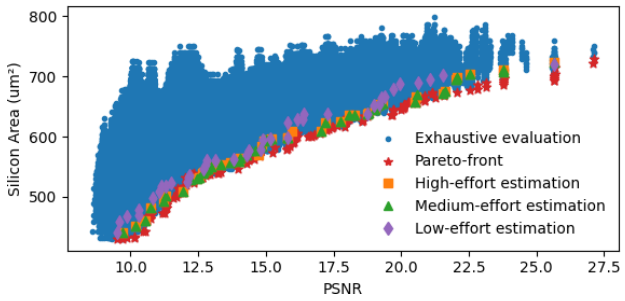


FIGURE 12. Comparison of results from exhaustive evaluation w.r.t estimation from E-𝕀DE𝔸 (PSNR vs. estimated silicon area).

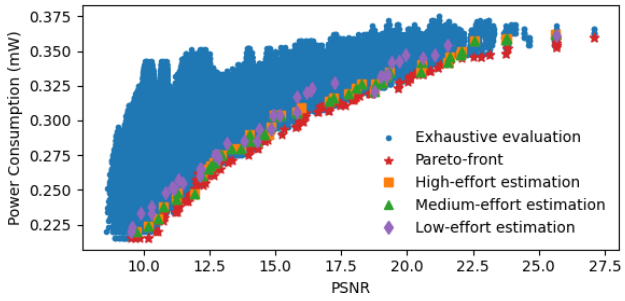


FIGURE 13. Comparison of results from exhaustive evaluation w.r.t estimation from E-𝕀DE𝔸 (PSNR vs. estimated power consumption).

TABLE 3. Comparison of results from [32], in terms of normalized distance between estimated and actual Pareto front.

Effort	Results from [32]			
	Avg.	Max.	Avg.	Max
Low	2.5e-3	7.5e-3	2.4e-4	1.7e-3
Med.	2.5e-4	1.3e-3	6.9e-6	5.6e-4
High	1e-5	6.5e-3	6.8e-6	5.4e-4

c: COMPARISON WITH PREVIOUS WORKS

A comparable case study is presented in [32], where a Sobel vertical-edge detector is approximated using circuits from the EvoApproxLib8b library [31], and results from hill-climbing search, varying the effort, are confronted with those from exhaustive exploration.

The E-𝕀DE𝔸 is capable of providing comparable experimental setup. In particular, Table 3 reports comparison of results from [32] with results from our method, for comparable effort levels. As the reader can figure out, the NSGA-II allows achieving better results by at least an order of magnitude. Furthermore, when compared to the one proposed [32], our method does not require (i) the user to know how the filter is implemented, (ii) a full characterization of circuits from the library, (iii) the library pre-processing step to eliminate irrelevant circuits, which significantly reduces activities demanded to the designer, and simplifies the design process.

E. CONVOLUTIONAL NEURAL NETWORK

In this case study, we target Convolutional Neural Networks (CNNs), that are a class of deep neural networks most commonly applied to analyzing visual imagery. In particular,

we focus on a LeNet5 [50] network, trained to classify images from the Modified National Institute of Standards and Technology (MNIST) benchmark [51].

Table 4 reports relevant structural parameters of the considered network, including the type of each layer, its input and output volume size, kernel size, activation functions and so forth.

TABLE 4. Structural characteristics of LeNet5 [50].

Layer	Conv1	Pool1	Conv2	Pool2	Conv3	Full1	Full2
Input volume	32x32x1	28x28x6	14x14x6	10x10x16	5x5x16	1x1x120	84x1x1
Kernel size	5x5	2x2	5x5	2x2	5x5	-	-
Stride	1	1	1	2	1	-	-
Padding	0	-	0	-	0	-	-
Filters	6	-	16	-	120	-	-
Activation	rect	-	rect	-	rect	rect	linear
Pooling	-	max	-	max	-	-	-
Output volume	28x28x6	14x14x6	10x10x16	5x5x16	1x1x120	84x1x1	10x1x1

We aim at designing a hardware accelerator suitable to speed-up weighted sums computed within neurons, in order to reduce both hardware requirements and power consumption. In this case study, we apply two approximate techniques, i.e. precision scaling and approximate circuits, to multipliers, since they are considered the most demanding arithmetic unit within CNNs [52], [53]. We discuss experimental setup and results in Section IV-E1 and Section IV-E2, respectively.

1) PRECISION-SCALING

a: SETUP

The network being considered during this first experiment has been trained using single precision floating-point, exhibiting 99.07% accuracy. We performed, then, performed 8-bit integer quantization, without experiencing any accuracy loss. Nevertheless, being the tool working on an already trained network model, the overall approach does depend neither on the particular network nor the particular implementation being considered.

We configured Clang-Chimera to truncate input operands and results of multiplications in the three convolutional and in the two fully connected layers of the considered network. Thus, the tool generates an approximate version of the considered CNN in which it is possible to configure, for each multiplication involved in the weighted sum, the NABs, in order to tune the introduced approximation degree.

To estimate the error introduced by the approximation, we configured Bellerophon to execute the approximate CNN to obtain its classification accuracy; then, Bellerophon compares it to the non-approximate 8-bits quantized CNN accuracy, on the MNIST test data set [51], to calculate the error as the difference between the two CNNs accuracy. One of the objectives of the Bellerophon is to find approximate solutions minimizing the so-defined error fitness function.

Concerning the *reward* fitness function, we estimate the gains by taking into account several network parameters that definitely have some impact on hardware requirements, such as: (i) the NABs within a single neuron, since the number of bits impacts hardware requirements of multipliers, (ii) the input-volume of a neuron, which impacts the amount of operations performed within it, and (iii) the amount of neurons within a layer, i.e. the output volume size of a layer, which impacts the hardware requirements of the whole layer. In details, let us consider the following:

- S: the data-width for inputs, weights, biases and outputs in the non-approximate CNN;
- N: the amount of approximate layers;
- $I_i = d_i \times h_i \times w_i$: the input volume size of each neuron belonging to the i-th layer;
- $O_i = D_i \times H_i \times W_i$: the i-th layer output volume size;
- NAB_i: the NABs for multiplications within the i-th layer.

Our proposed *reward fitness function* (6) is defined as the ratio between neglected and total bits, weighted according to input and output volume size of each layer.

$$\rho = \frac{\sum_i^N NAB_i \times I_i \times O_i}{S \times \sum_i^N I_i \times O_i} \tag{6}$$

b: RESULTS

The design-space exploration phase took about 5 hours so complete, and provided approximate configurations reported in Table 5. For each of the configurations, the amount of neglected bits for each layer and the corresponding fitness function values are reported. Being CNNs quite error resilient, approximate configurations exhibit negligible error, in spite of significant potential saving estimated through (6). Please note that, although negligible, some approximate configurations exhibit even some classification accuracy improvements.

TABLE 5. Bellerophon results for LeNet5 while applying precision-scaling.

Conf#	Error (%)	Reward (%)	Conv.1	Conv.2	Conv.3	F.C.1	F.C.2
1	-0.07	16.64	0	2	1	1	3
2	-0.06	24.60	2	2	0	2	3
3	-0.02	24.94	2	2	2	2	0
4	0.06	28.16	3	2	0	1	2
5	0.07	28.61	3	2	2	1	3
6	0.10	28.86	3	2	2	2	0
7	0.12	32.60	2	3	0	2	3
8	0.18	32.91	2	3	0	3	2
9	0.33	35.80	3	3	0	0	1
10	0.34	36.25	3	3	2	0	2
11	0.35	36.28	3	3	2	0	3
12	0.38	36.99	3	3	1	3	0
13	0.48	37.31	3	3	2	3	4

In order to measure actual hardware savings, we designed a parallel weighted-sum accelerator in VHDL. Such an accelerator guarantees high flexibility, since it handles the configuration of (i) the data width for synaptic weights, inputs, biases and outputs, (ii) input and output volume size, and (iii) the NABs for multiplications, allowing hardware synthesis of any solution eventually found during the design space exploration process performed with E-𝕀DE𝔸. Our implementation aims at taking advantage from the massive inner-parallelism hardware provides; therefore, unrelated operations are performed exploiting parallel multipliers and adders. Inputs and

weights signals are fed into a multiplier block that computes partial-products using $d \times h \times w$ parallel multipliers, where d, h and w are, respectively, the depth, height and width of the input-volume. These partial-products are, then, fed into a binary-tree-based sum-reduction block. It consists of $log_2(d \times h \times w) + 1$ levels, each one having 2^l parallel adders, with $l \in [0, log_2(d \times h \times w)]$; $l = 0$ is the root-node of the reduction tree, i.e. the one computing the final sum.

In order to obtain the hardware implementations, the non-dominated solution configurations (reported in Table 5) are employed to configure the above mentioned VHDL design. Finally, we performed FPGA synthesis targeting a Xilinx Virtex Ultrascale+ FPGA to measure the actual hardware gains entailed by the approximation. Figure 14 reports the synthesis results. The stacked bars graph represents, for each layer, the requirements of the above discussed accelerator in terms of both LUTs and Flip-Flops (FFs).

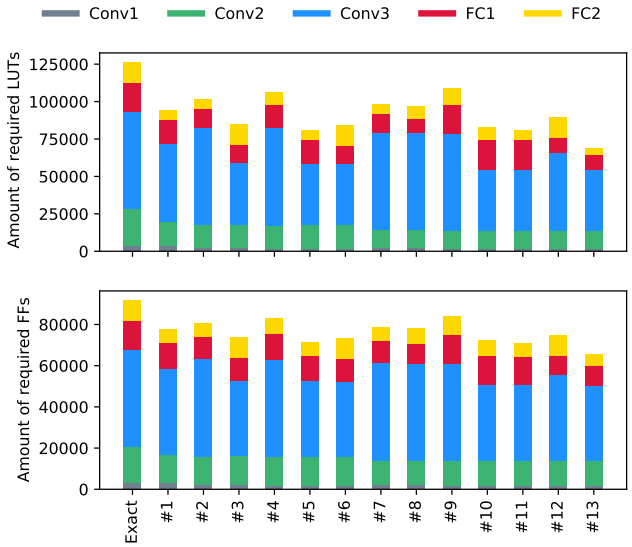


FIGURE 14. FPGA resource requirements (single neuron per layer) for the LeNet5 CNN.

As foreseeable, accelerating the weighted sum computation for the third convolutional layer (Conv.3) requires the largest amount of resources, due to the input volume size of such a layer, which is far larger w.r.t. other layers.

As it can be observed, overall the predicted trend is confirmed: the hardware resources significantly decrease as the introduced classification error increases. Configuration #13, for instance, allows achieving up to 40% and 30% savings in terms of LUTs and FFs, respectively, with only a 0.48% accuracy loss. As it is easy to imagine, savings due to precision-scaling does not only affect hardware requirements, but also energy consumption. Trivially, the less hardware a circuit require, the less energy is spent to power it. Moreover, the whole circuit is also expected to exhibit a lower switching activity w.r.t. its exact counterpart, since the least significant bits of inputs, weights and biases are constant-zero signals thanks to the approximation. In order to evaluate potential power savings, we performed simulations on the non-dominated approximate configurations shown in Table 5 to compare them to the exact (non-approximate) counterpart. Simulations involve 10,000 input combinations, each consisting of an appropriate amount of inputs, weights and bias vectors depending on the input volume size of the considered accelerator. Figure 15 reports the results. The third convolutional layer requires larger amount of power w.r.t. the other layers, since it performs far more parallel multiplications and additions.

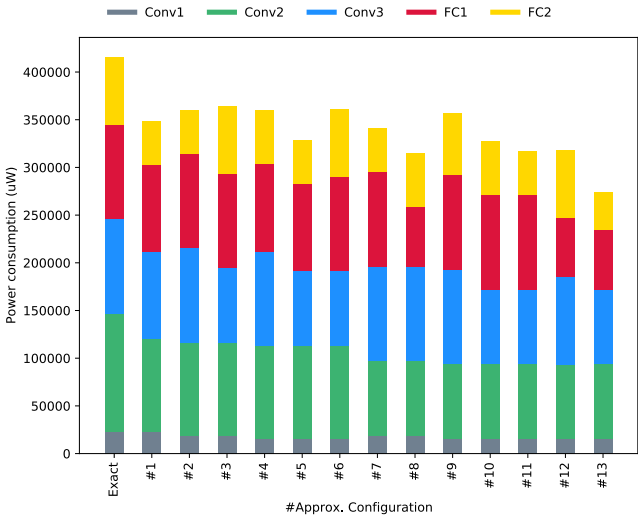


FIGURE 15. Power consumption (single neuron per layer) for the LeNet5 CNN.

The non-monotonic trend of the reported results are due to the heuristic nature of the algorithms used by the synthesis tool. Therefore, a solution needing less resources (i.e. higher reward in Table 5) could end up being less optimizable than one needing more (i.e. lower reward in Table 5), especially in complex application, such as CNNs. Nevertheless, the exploration performed with E-𝕀DE𝔸 is useful to identify the approximate configurations achieving good trade-offs between hardware gains and accuracy loss. This introduce the important benefit of synthesizing only a reduced set of the explored approximate configurations (i.e. 13 over ∼40,000) to finally find the best suited one for the given design requirements.

2) APPROXIMATE CIRCUITS

a: SETUP

The network we consider in this experiment results from 16-bits quantization on the single-precision floating-point network from which we begin the above discussed case study. After quantization, we witness no accuracy loss.

The Clang-Chimera is configured to supersede exact multiplications in the three convolutional and in the two fully connected layers using approximate circuits taken from the EvoApproxLib-Lite [36]. Thus, the tool generates an approximate version of the considered CNN in which it is possible

to select, for each multiplication involved in the weighted sum computation within neurons, an implementation between either the exact or an approximate implementation from the library.

Concerning MOP, as done for precision-scaling, one of the objectives of the Bellerophon is to find approximate solutions minimizing the classification-accuracy loss. Therefore, in order to perform error assessment, we configured Bellerophon to execute the approximate CNN to obtain its classification accuracy on the MNIST test data set [51]. Then, that accuracy is compared against the accuracy of the non-approximate 16-bits quantized CNN, and the error is computed as the difference between the two CNNs accuracy.

In this experiment we also pursue circuit area minimization: we estimate the circuit area by taking into account (i) the silicon area of exact and approximate multipliers, as reported in [36], (ii) the input-volume of a neuron, which impacts the amount of operations performed within it, and (iii) the amount of neurons within a layer, i.e. the output volume size of a layer. In details, being: (i) N: the amount of approximate layers, (ii) $I_i = d_i \times h_i \times w_i$: the input volume size of each neuron belonging to the i-th layer, (iii) $O_i = D_i \times H_i \times W_i$: the i-th layer output volume size, (iv) α_i: the silicon area of the multiplier being adopted within i-th layer as reported in [36], the Bellerophon aims at minimizing Equation (7), which is the sum of the silicon area of multipliers being adopted within each of the layers, weighted according to input and output volume size of each layer.

$$\rho = \sum_{i}^{N} \alpha_i \times I_i \times O_i \tag{7}$$

Furthermore, we also aim to minimize the power consumption of the circuit. We estimate power consumption resorting to the same reasoning as for silicon area minimization. In particular, being β_i: the power consumption of the multiplier adopted within i-th layer, the Bellerophon aims at minimizing Equation (8), which is the sum of the power consumption of multipliers being adopted within each layer, weighted according to the input and output volume size.

$$\psi = \sum_{i}^{N} \beta_i \times I_i \times O_i \tag{8}$$

b: RESULTS

The design-space exploration phase took about 25 hours to complete, due to the increased time needed to simulate C circuit models from [36]. Resulting approximate configurations are reported in Table 6. For each of the configurations, besides the corresponding fitness function values, also the multiplier circuits being adopted are reported.

In order to measure actual hardware requirements, we resort to the parallel weighted-sum accelerator we discussed above. In order to allow the hardware synthesis of any solution eventually found during the DSE process performed with E-𝕀DE𝔸, we added the possibility of using multipliers from [36] to that parallel weighted-sum accelerator.

TABLE 6. Bellerophon results for LeNet5 while using approximate circuits from [36].

Conf#	Error (%)	Silicon area (μm^2)	Power (W)	Conv.1	Conv.2	Conv.3	F.C.1	F.C.2
1	-0.04	1012.24×10^6	945.199	Exact	HHP	Exact	GK2	HDG
2	0.15	900.95×10^6	736.031	HFZ	HDG	Exact	HDG	Exact
3	-0.08	1143.84×10^6	1014.180	Exact	GK2	G7Z	HHP	HFZ
4	0.13	950.48×10^6	789.560	HFZ	G7Z	G80	Exact	G7Z
5	0.04	970.83×10^6	859.568	HDG	HEB	HHP	G7F	G7Z
6	0.07	970.17×10^6	784.903	G80	G7F	G80	HEB	G7F
7	0.01	997.27×10^6	832.993	G80	HDG	Exact	GK2	HFZ
8	-0.02	999.64×10^6	930.959	Exact	HHP	HHP	GK2	HDG
9	0.22	814.35×10^6	648.202	HDG	HFZ	Exact	HFZ	HFZ
10	0.34	731.97×10^6	563.760	HFZ	HFZ	GK2	HHP	G80

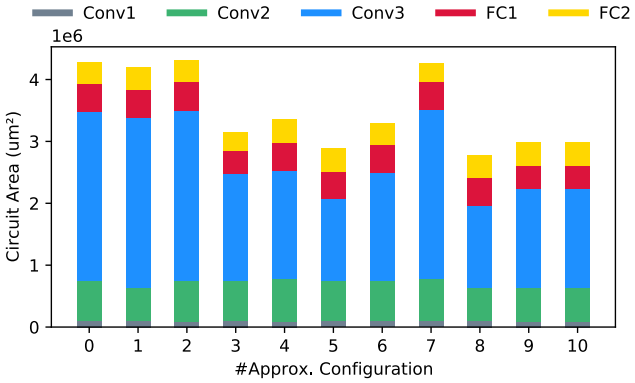


FIGURE 16. Silicon area requirements (single neuron per layer) for the LeNet5 CNN.

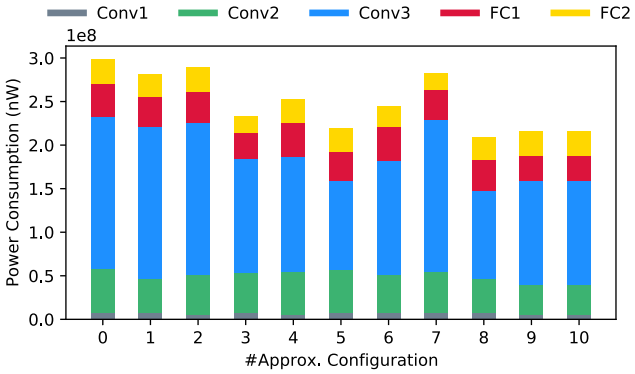


FIGURE 17. Power consumption (single neuron per layer) for the LeNet5 CNN.

Approximate configurations reported in Table 6 are employed to configure the above-mentioned accelerator; then, we performed ASIC synthesis targeting the $65\mu m$ FinFET library to measure the actual hardware requirements entailed by the approximation.

Figure 16 and Figure 17 report silicon die area requirements and estimated power consumption provided by the Cadence Genus Synthesis Solution tool. As discussed above, the third convolutional layer (Conv.3) is the most burdensome since its input volume size. Anyway, the predicted savings are still confirmed: using the E-𝕀DE𝔸 tool results in hardware resources significantly decrease as the introduced classification error increases. Configuration #8, for instance, allows achieving up to 35% and 30% savings in terms of silicon area and power consumption.

Authors of [36] proposed a comparable approach in which basic arithmetic components are first approximate while considering component-based error and saving metrics, thus employed to design a hardware accelerator CNNs. In particular, multipliers are employed to perform error resiliency estimation of single layers of CNNs, and to reduce figure of merits such as silicon area and power consumption of hardware accelerators.

As already shown in Section IV-D3, although the results are not directly comparable, those achieved through the use of our tool – 0% classification accuracy drop against 35% area and 30% power savings – are on the same quantitative relation as those obtained using state-of-the-art approaches – 1.8% accuracy traded for 29% power savings. On the other hand, looking at the methodological aspect, we deem that our approach provides a significant step forward. In fact, it allows to consider the application as a whole, and to explore, in a single run, the different degrees of approximation that each of the layers of the network is able to withstand, allowing to diversify the degree of approximation that can be exploited during the design of a hardware accelerator. Conversely, the state-of-the-art studies carry out the analysis of resilience with respect to the error by considering the layers one at a time, and, as far as the design of an accelerator is concerned, they do not fully exploit the approximation that is possible to introduce, employing the same degree of approximation for all the layers. Moreover, they do not allow to explore the effects of different approximation techniques on the same application.

3) COMPARING PRECISION-SCALING AND APPROXIMATE-CIRCUITS

In this section, exploiting flexibility provided by E-𝕀DE𝔸, we make a comparison between the precision-scaling and the approximate circuit techniques, while resorting to the CNN case study discussed in Section E.1 and E.2.

Resorting to the 16-bits quantized CNN model we adopted for the approximate circuit technique case study, here we configure Clang-Chimera to truncate input operands and results of multiplications in the three convolutional and in the two fully connected layers of the considered network. As a result, the tool generates an approximate version of the considered CNN in which it is possible to configure, for each multiplication involved in the weighted sum, the number of neglected bits, in order to tune the introduced approximation degree. Concerning the design-space exploration phase, we configured Bellerophon to minimize both the classification-accuracy loss and silicon die area. In particular, (i) Bellerophon compares the approximate and the non-approximate networks in terms of classification accuracy while considering the MNIST test data set, in order to assess error, and (ii) for what pertains to area minimization, we resort to Equation (9). Finally, we synthesized approximate configurations resulting from Bellerophon while targeting the 65 μm FinFET technology library – the same we targeted for the approximate circuit case study – in order to compare actual savings provided by the adopted approximate techniques, both in terms of silicon area and power consumption.

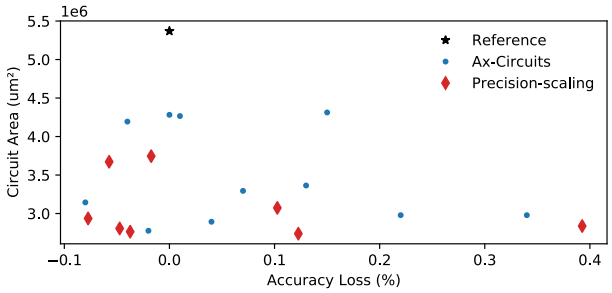


FIGURE 18. Comparison between precision-scaling and approximate circuits techniques, in the error v.s circuit area perspective.

Results of such a comparison are reported in the classification accuracy loss vs silicon area and power consumption perspectives, respectively in Figure 18 and Figure 19. In order to state which Pareto-front provides better results, we resort to the *Coverage of two sets* metric, proposed in [54]. Let A and B be two sets of non-dominated solutions for a MOP. The function $\mathcal{C}$ in Equation (9) maps the pair $(A,\ B)$ to the interval [0, 1]: where the expression α *covers* β ($\alpha \succeq \beta$) means that the solution α *dominates* the solution β or they are *the same* solution. The value $\mathcal{C}(A, B) = 1$ means that all points in B are dominated by or equal to points in A. Conversely, the value $\mathcal{C}(A, B) = 0$ represents the situation where no points in B are covered by any points in A. When using this metric, both $\mathcal{C}(A, B)$ and $\mathcal{C}(B, A)$ have to be considered, as they are not necessarily equal.

$$\mathcal{C}(A, B) := \frac{|\{\forall \beta \in B;\ \exists\, \alpha \in A : \alpha \succeq \beta\}|}{|B|} \tag{9}$$

We measured $\mathcal{C}$ between the Pareto-fronts resulting from precision-scaling and approximate circuits, and vice-versa. As reported in Table 7, we obtained 90% and 22% coverage, which means the precision-scaling dominates the approximate circuits techniques 90% of time, while the opposite occurs only 22% of time.

TABLE 7. ***Coverage of two sets*** **metric between Pareto fronts.**

Pareto-fronts	$\mathcal{C}$ (A, B)
Ax Circuits v.s. Precision scaling	0.22
Precision scaling v.s. Ax Circuits	0.91

Although the precision-scaling seems to provide better trade-offs w.r.t the approximate circuits technique, results might differ whether considering a different application domain, paving the way for future research.

F. DISCUSSION

The conducted experiments showed the flexibility of the proposed E-𝕀DE𝔸 framework. Indeed, we were able to quickly evaluate the resiliency of algorithms to a particular

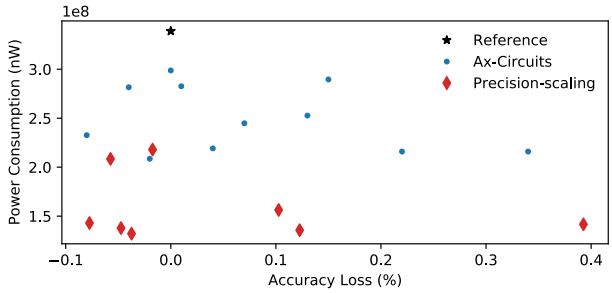


FIGURE 19. Comparison between precision-scaling and approximate circuits techniques, in the error v.s power consumption perspective.

approximation technique. This was possible thanks to the evolutionary-based exploration of different approximate configurations to ultimately produce efficient variants without significantly impacting the output accuracy, as shown, for instance, in the k-means case study.

Furthermore, with E-𝕀DE𝔸 it is also simple to automatically evaluate the impact of different variants of the same approximation technique: in the Taylor series expansion case study we showed that two variants of the same AxC technique (loop perforation) lead to different trade-offs between performance and accuracy loss (Figure 5). A simple modification in the E-𝕀DE𝔸 mutator allowed us to easily perform the analysis.

Moreover, since different approximation techniques are applicable to different targets, usually it is complicated to evaluate their impact on an application. For example, while the bit-width reduction is more suitable when targeting a custom hardware accelerator – since it is fully customizable – the loop perforation is mostly applied in software implementations. Nevertheless, with E-𝕀DE𝔸 it is possible to easily explore the trade-off opportunities stemming from the different AxC techniques within the same framework. In this regard, JPEG and Sobel filter case studies proved the possibilities of exploring the trade-off opportunities brought by two totally different AxC techniques – bit-width reduction and loop perforation – to the same application. To do so, we simply defined different code mutators and fitting functions.

Thus, E-𝕀DE𝔸 allows exploring the impact of different approximation techniques on different benchmarks, with different targets (i.e., software, hardware accelerators) and error metrics.

Finally, the approach introduced with E-𝕀DE𝔸 is scalable. Indeed, as shown in the last case study, it allows analyzing complex applications as the CNNs. We instrumented E-𝕀DE𝔸 to apply the *precision scaling* approximation technique to a 8-bit quantized CNN. This allowed us to find solutions achieving more than 30% savings, with a negligible accuracy loss (0.48%).

Besides, Table 8 summarize reported case studies, including the approximate technique being adopted and which are the parts of the application on which the technique is applied,

TABLE 8. Summary of reported case studies, including employed techniques, approximate parts, fitness functions and DSE.

Application	Technique	Part(s)	Fitness	Exploration Time (min)
K-Means	Loop1	Centroid computation	Min.Error & Min.Comp.Time	≈90
Taylor	Loop1	Amount of series terms	Min.Error & Min.Comp.Time	<1
	Loop2			<3
JPEG Compression	Precision scaling	Addition & multiplications bit-width	Min.Error & Min.Bit-width	≈55
	Loop1	Amount of DCT terms	Min.Error & Min.Comp.Time	≈20
Sobel edge detection	Precision scaling	Addition & multiplications bit-width	Min.Error & Min.Area	≈57
	Loop1	Amount of multiplications in convolution	Min.Error & Min.Comp.Time	≈20
	Ax-cirtcuits	Addition operators in convolution	Min.Error, Min.Area & Min.Power	≈5
CNN	Precision Scaling	Multiplications within neurons	Min.Error, Min.Area	≈300
	Ax-cirtuits		Min.Error, Min.Area & Min.Power	≈1500

the fitness-functions driving the DSE, and the execution time that E-𝕀DE𝔸 spent to perform the latter. In general, the exploration time depends on the application, specifically on its requirements in terms of computation and code complexity. Also, the complexity of the metric calculation impacts the exploration time. Indeed, for instance, for all the applications used to process images (k-means, JPEG, and Sobel) E-𝕀DE𝔸 had to calculate the related metrics (PSNR, SSIM) to determine the accuracy loss. Moreover, to calculate the approximate-CNN accuracy loss, the inference process must be executed and the results compared to the non-approximate CNN. On the other hand, the Taylor benchmark only needed the calculation of the Maximum Absolute Error which was pretty quick.

V. CONCLUSION

In this paper, we presented E-𝕀DE𝔸, an automated framework able to generate approximate variants of a given input application described as C/C++ code. The approach is based on automatic code mutation and on an evolutionary engine, whose synergy enables the exploration of the best approximate variants in terms of trade-offs between accuracy reduction and efficiency gains. E-𝕀DE𝔸 has been validated over different kinds of applications and target implementations (i.e. software, hardware accelerators). Experimental results presented throughout the article showed that E-𝕀DE𝔸 can be very useful as a design exploration tool to help the user selecting the most suitable approximation technique, according to the desired requirements and constraints.

E-𝕀DE𝔸 is released under the *GNU Affero General Public License* and it is completely open-source [34]. We believe that the E-𝕀DE𝔸 approach will help software and hardware engineers to design the future energy-efficient integrated systems and applications.

REFERENCES

[1] W. Liu, F. Lombardi, and M. Shulte, "A retrospective and prospective view of approximate computing [Point of view]," *Proc. IEEE*, vol. 108, no. 3, pp. 394–399, Mar. 2020.

[2] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proc. 50th Annu. Design Autom. Conf. (DAC)*, 2013, p. 113.

[3] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proc. 52nd Annu. Design Autom. Conf.*, Jun. 2015, p. 120.

[4] A. Sampson, A. Baixo, B. Ransford, T. Moreau, J. Yip, L. Ceze, and M. Oskin, "Accept: A programmer-guided compiler framework for practical approximate computing," Univ. Washington, Washington, DC, USA, Tech. Rep. UW-CSE-15-01, 2015, vol. 1, no. 2.

[5] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. 18th IEEE Eur. Test Symp.*, May 2013, pp. 1–6.

[6] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.

[7] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," *Commun. ACM*, vol. 58, no. 1, pp. 105–115, Jan. 2015, doi: 10.1145/2589750.

[8] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "IMPACT: IMPrecise adders for low-power approximate computing," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design*, Aug. 2011, pp. 409–414.

[9] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 1, pp. 124–137, Jan. 2013.

[10] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1760–1771, Sep. 2013.

[11] A. Cilardo, "A new speculative addition architecture suitable for two's complement operations," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Apr. 2009, pp. 664–669.

[12] A. Cilardo, "Modular inversion based on digit-level speculative addition," *Electron. Lett.*, vol. 49, no. 25, pp. 1609–1610, Dec. 2013.

[13] A. Cilardo, "Variable-latency signed addition on FPGAs," in *Proc. 25th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2015, pp. 1–6.

[14] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2011, pp. 667–673.

[15] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. Accuracy trade-offs with loop perforation," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng. (SIGSOFT/FSE)*. New York, NY, USA: Association for Computing Machinery, 2011, pp. 124–134, doi: 10.1145/2025113.2025133.

[16] C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough, "Precimonious: Tuning assistant for floating-point precision," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2013, p. 27.

[17] W. G. Osborne, J. Coutinho, W. Luk, and O. Mencer, "Power-aware and branch-aware word-length optimization," in *Proc. 16th Int. Symp. Field-Program. Custom Comput. Mach.*, Apr. 2008, pp. 129–138.

[18] F. Fang, T. Chen, and R. A. Rutenbar, "Floating-point bit-width optimization for low-power signal processing applications," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 3. IEEE, 2002.

[19] M. Wyse, A. Baixo, T. Moreau, B. Zorn, J. Bornholt, A. Sampson, L. Ceze, and M. Oskin, "React: A framework for rapid exploration of approximate computing techniques," in *Proc. Workshop Approx. Comput. Across Stack (WAX W/PLDI)*, 2015.

[20] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 164–174, Jun. 2011.

[21] *Clang: A C Language Family Frontend for LLVM*. Accessed: Jun. 16, 2021. [Online]. Available: https://clang.llvm.org/

[22] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proc. IEEE*, vol. 108, no. 12, pp. 2108–2135, Dec. 2020.

[23] W. Liu, T. Cao, P. Yin, Y. Zhu, C. Wang, E. E. Swartzlander, and F. Lombardi, "Design and analysis of approximate redundant binary multipliers," *IEEE Trans. Comput.*, vol. 68, no. 6, pp. 804–819, Jun. 2019.

[24] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2014, p. 361.

[25] S. Lee, L. K. John, and A. Gerstlauer, "High-level synthesis of approximate hardware under joint precision and voltage scaling," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 187–192.

[26] D. Ma, R. Thapa, X. Wang, C. Hao, and X. Jiao, "Workload-aware approximate computing configuration," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 258–261.

[27] G. Zervakis, H. Amrouch, and J. Henkel, "Design automation of approximate circuits with runtime reconfigurable accuracy," *IEEE Access*, vol. 8, pp. 53522–53538, 2020.

[28] P. Huang, C. Wang, W. Liu, F. Qiao, and F. Lombardi, "A hardware/software co-design methodology for adaptive approximate computing in clustering and ANN learning," *IEEE Open J. Comput. Soc.*, vol. 2, pp. 38–52, 2021.

[29] Z. Vasicek and L. Sekanina, "Circuit approximation using single-and multi-objective Cartesian GP," in *Proc. Eur. Conf. Genetic Program.* Cham, Switzerland: Springer, 2015, pp. 217–229.

[30] L. Sekanina, Z. Vasicek, and V. Mrazek, "Automated search-based functional approximation for digital circuits," in *Approximate Circuits*. Cham, Switzerland: Springer, 2019, pp. 175–203.

[31] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 258–261.

[32] V. Mrazek, M. A. Hanif, Z. Vasicek, L. Sekanina, and M. Shafique, "AutoAx: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components," in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.

[33] M. Barbareschi, F. Iannucci, and A. Mazzeo, "Automatic design space exploration of approximate algorithms for big data applications," in *Proc. 30th Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, Mar. 2016, pp. 40–45.

[34] *Iidea*. Accessed: Jun. 16, 2021. [Online]. Available: http://wpage.unina.it/mario.barbareschi/iideaa/

[35] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, Sep. 2011.

[36] V. Mrazek, L. Sekanina, and Z. Vasicek, "Using libraries of approximate circuits in design of hardware accelerators of deep neural networks," in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Aug. 2020, pp. 243–247.

[37] A. Osyczka, "Multicriteria optimization for engineering design," in *Design Optimization*, J. S. Gero, Ed. New York, NY, USA: Academic, 1985, pp. 193–227. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B978012280910150012X

[38] X. Yu and M. Gen, *Introduction to Evolutionary Algorithms*. Springer, 2012.

[39] *Paradiseo*. Accessed: Jun. 16, 2021. [Online]. Available: http://paradiseo.gforge.inria.fr

[40] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *Proc. Int. Conf. Parallel Problem Solving Nature*. Berlin, Germany: Springer, 2000, pp. 849–858.

[41] *The LLVM on-Request Compiling API*. Accessed: May 5, 2021. [Online]. Available: https://llvm.org/docs/ORCv2.html

[42] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "AxBench: A multiplatform benchmark suite for approximate computing," *IEEE Des. Test.*, vol. 34, no. 2, pp. 60–68, Apr. 2017.

[43] G. Stockman and L. G. Shapiro, *Computer Vision*, 1st ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2001.

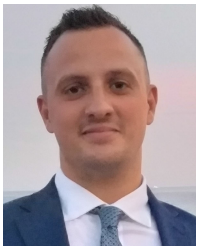
[44] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.

[45] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62-1–62-33, Mar. 2016, doi: 10.1145/2893356.

[46] S. T. Welstead, *Fractal and Wavelet Image Compression Techniques*, 1st ed. Bellingham, WA, USA: Society of Photo-Optical Instrumentation Engineers (SPIE), 1999.

[47] M. Barbareschi, F. Iannucci, and A. Mazzeo, "An extendible design exploration tool for supporting approximate computing techniques," in *Proc. Int. Conf. Design Technol. Integr. Syst. Nanosc. Era (DTIS)*, Apr. 2016, pp. 1–6.

[48] Vivado High-Level Synthesis. Accessed: Jun. 16, 2021. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration.html
[49] *The USC-SIPI Image Database*, S. Univ. Southern California and I. P. Institute, 2012. Accessed: Jun. 16, 2021. [Online]. Available: http://sipi.usc.edu/database
[50] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[51] *The MNIST Database of Handwritten Digits*. Accessed: May 21, 2020. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[52] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 345–353.
[53] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, and K. Roy, "Energy-efficient neural computing with approximate multipliers," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 2, pp. 1–23, Jul. 2018.
[54] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 257–271, 1999.



SALVATORE BARONE (Graduate Student Member, IEEE) received the master's degree *(cum laude)* in computer engineering from the University of Naples Federico II, Italy, in 2018, where he is currently pursuing the Ph.D. degree. His research interests include safety critical systems, railway systems, hardware security and trust, approximate computing, and embedded systems based on the FPGA technology.

MARCELLO TRAIOLA (Member, IEEE) received the M.Sc. degree *(cum laude)* in computer engineering from the University of Naples Federico II, Italy, in 2016, and the Ph.D. degree in computer engineering from the University of Montpellier, France, in 2019. He is currently a Postdoctoral Researcher with the Lyon Institute of Nanotechnology, Ecole Centrale de Lyon, France. His main research interests include emerging computing paradigms with special interest in design, test, and reliability.

MARIO BARBARESCHI (Associate Member, IEEE) received the master's degree *(cum laude)* in computer engineering and the Ph.D. degree in computer and automation engineering from the University of Naples Federico II, in 2012 and 2015, respectively. His research interests include hardware security and trust, cyber physical security, approximate computing, and embedded systems design based on the FPGA technology. He has authored more than 30 peer-reviewed articles published in leading journals and international conferences.

ALBERTO BOSIO (Member, IEEE) received the Ph.D. degree in computer engineering from the Politecnico di Torino, Italy, in 2006. He is currently a Full Professor with the Ecole Centrale de Lyon, Institute of Nanotechnology, France. He published articles spanning diverse disciplines, including testing, verification, reliability, approximate computing, and emerging technologies. He is the Chair of the European Test Technical Technology Council (eTTTC).