ReDO: Cross-Layer Multi-Objective Design-Exploration Framework for Efficient Soft Error Resilient Systems

ReDO: Cross-Layer Multi-Objective Design-Exploration Framework for Efficient Soft Error Resilient Systems

Alessandro Savino, *Member, IEEE,* Alessandro Vallero, *Member, IEEE,* and Stefano Di Carlo *Senior Member, IEEE*

Abstract—Designing soft errors resilient systems is a complex engineering task, which nowadays follows a cross-layer approach. It requires a careful planning for different fault-tolerance mechanisms at different system's layers: starting from the technology up to the software domain. While these design decisions have a positive effect on the reliability of the system, they usually have a detrimental effect on its size, power consumption, performance and cost. Design space exploration for cross-layer reliability is therefore a multi-objective search problem in which reliability must be traded-off with other design dimensions. This paper proposes a cross-layer multi-objective design space exploration algorithm developed to help designers when building soft error resilient electronic systems. The algorithm exploits a system-level Bayesian reliability estimation model to analyze the effect of different cross-layer combinations of protection mechanisms on the reliability of the full system. A new heuristic based on the extremal optimization theory is used to efficiently explore the design space. Two exploration strategies are proposed. The first strategy aims at optimizing the reliability of the system alone. It is suited in those cases in which reaching a given reliability target is the sole goal. It focuses on finding a reduced set of system's components that, when protected, allow the designer to reach the desired reliability level. As a positive effect, by reducing the number of protected components, the overhead introduced by the fault tolerance techniques is reduced as well. The second strategy jointly considers the effect that the introduced fault-tolerance mechanisms have on the execution time, power, hardware area and software size. This strategy supports the exploration of the design space setting multiple objectives on different design dimensions. An extended set of simulations shows the capability of this framework when applied both to benchmark applications and realistic systems, providing optimized systems that outperform those obtained by applying state-of-the-art cross-layer reliability techniques.

Index Terms—Cross-layer Reliability, extremal optimization, multi-objective optimization, soft errors

1 Introduction

In highly integrated systems, process innovations and miniaturization have led to higher vulnerability to faults, and in particular to soft errors [1], [2]. Soft errors are transient faults that manifest as bit flips in a hardware structure. They can be caused by either internal or external sources such as high energy particle strikes [3]. The effect of soft errors may impact the correctness of the computation [4].

Unavoidably, designers devote significant resources (i.e., effort, budget, circuit area and computation time) to ensure sufficient resiliency to soft errors before a computing system is released to market. This activity is supported by a reach literature on soft error resilience techniques, many of which span multiple abstraction layers. A comprehensive survey of different reliability threats and single and cross-layer dependability approaches can be found in [5].

In a cross-layer soft error resilient system, soft error management (i.e., detection, diagnosis, reconfiguration, recovery and adaptation) is performed by a combination of hardware and software protection techniques implemented at different layers of the system stack [6], [7], [8], [9].

In this context, reliability related design decisions regarding the target hardware architecture, running software and their related fault-tolerance mechanisms are always translated into performance, area and power overheads [10]. This motivates the need to perform Design-Space Exploration (DSE) in order to take design decisions driven by a multi-objective optimization goal, in which reliability is traded-off with other design dimensions [10]. However, the number of constraints and parameters to consider when performing this task is rapidly growing to a level that cannot be handled anymore by system designers without the support of proper automatic DSE tools.

The literature presenting automatic multi-objective DSE techniques for cross-layer soft error resilience systems is rapidly growing, with several research groups addressing the problem from different perspectives (see section 2 for a survey of relevant works).

This paper proposes ReDO (Reliability Design Optimizer), a DSE framework to build soft error resilient computing systems. ReDO is designed to support the early phase of the design of a computing system. It evaluates the application of selected classes of cross-layer soft error protection techniques taking into account multiple design objectives. Exploiting this framework during the design phase, reliability can be traded-off with other design constraints, i.e., hardware area, software size, performance and power consumption.

ReDo internally models the target system resorting to an extended version of the reliability model proposed by Vallero at al. in [9]. This model provides a very compact

• *A. Savino, A. Vallero and S. Di Carlo are with the Department of Control and Computer Engineering, Politecnico di Torino, Torino, Italy, 10129. E-mail: name.surname@polito.it*

component based representation of the system stack (from the fabrication technology up to the software layer) based on a Bayesian model. The model is designed to analyze the resiliency of the system to soft errors according to different reliability metrics. More importantly, it enables to estimate the impact of a design decision (i.e., application of a selected protection mechanism to a selected component) on the reliability of the global system. It represents a good compromise between the accuracy of the evaluated reliability metrics and the complexity of the analysis, which is a challenging task when considering cross-layer reliability techniques.

On top of the reliability model of the system, ReDO introduces a new exploration heuristic, based on the extremal optimization (EO) theory [11]. The heuristic is able to efficiently explore the design space composed of different combinations of cross-layer protection mechanisms applied to the components of the system. The goal of the EO is to optimize a global variable by improving local variables that involve coevolutionary avalanches. This is interesting in a cross-layer reliability scenario in which we want to evaluate how the application of different combinations of local protection mechanisms in selected components of the system (improvements of local variables) affect the global characteristics of the system in terms of reliability combined with power, area and performance (global variable). The characteristics that make an EO based heuristic particularly suited for the specific DSE problem described in this paper will be analyzed in detail in section 4.1.

The combination of the proposed reliability model with the DSE heuristic supports the analysis of a complex system in a limited computation time. This makes ReDO an interesting option to support designers in the early phases of the design cycle, when strategic decisions must be taken to design highly optimized systems.

A large campaign of experiments is reported to demonstrate the capability of the proposed framework. Experiments aim at the optimization of a set of systems based on realistic microprocessor models and running a set of software benchmarks and real applications.

To perform DSE we choose from the literature a library of protection techniques at different layers and we use it to generate a large set of design options. Since we focus on the early phase of the design, the criterion used to build this library is based on the possibility of modeling the impact of the technique on the considered design parameters without a complete implementation of the technique itself. It is important to remark that the aim of the experiments is not to demonstrate which technique or which combination of techniques is superior. The considered techniques are not an exhaustive selection of protection mechanisms. They must be considered as a set of inputs used to show the capability of the implemented DSE engine.

To show the advantages of exploiting the proposed cross-layer and multi-objective DSE framework, the characteristics of the optimized systems are then compared with those of the same systems protected with state-of-the-art cross-layer approaches.

The remaining of this paper is organized as follows: section 2 shortly overviews related works while section 3 presents the formalism used to model the target system during DSE. Section 4 overviews the DSE strategy and section 5 reports and discusses the results of the performed experimental campaign. Finally section 6 summarizes the main contributions and concludes the paper.

2 Related work

In the reliability domain, DSE has been approached from different perspectives, both considering single-layer and cross-layer approaches. This section overviews a set of relevant publications in this domain.

2.1 Single-layer approaches

Coit at al. [10] and Xing et al. [12] review a set of techniques to solve the redundancy allocation problem that somehow can be considered as a DSE approach. Most of the analyzed solutions are based on genetic algorithms and all start from the assumption that data redundancy is the only design option to increase the fault-tolerance of the system. The intrinsic resilience of the system to selected hardware faults is not taken into account during DSE. The only goal is to optimize the amount of redundancy with respect to area constraints.

Shafique et al. propose a reliability oriented DSE framework focusing on the software layer [13]. The technique incrementally protects selected instructions until the target reliability is achieved or, at worst, all unprotected instructions are protected. The hardware is only considered as a source of faults and its architecture is not considered in the exploration process.

The same research group also exploited the idea of carefully scheduling the instruction execution to optimize reliability and other design constraints [14], [15], [16]. Even if a cross-layer model of the system is considered when evaluating the reliability, the design space exploration is still limited to the software layer.

2.2 Cross-layer approaches

One of the first attempts to perform cross-layer DSE is proposed by Wattanapongsakorn and Levitan in [17]. They use simulated annealing to evaluate random configurations of all available components, selecting at each iteration the best identified combination. The considered cost function is simply the sum of the cost of each component. The optimization ends when the best solution does not improve for a pre-defined number of iterations. Although the technique is interesting, the exploited reliability model is oversimplified. It considers a single failure probability for each component without accounting for the effect of the interaction between components. Moreover, the trade-off between reliability and other design dimensions is not considered.

Cotofana et al. [18] propose a cross-layer DSE framework that focuses on the lower layers of the system starting from the technology up to the hardware architecture. The paper proposes a cell based design in which reliability can be tuned playing on different combinations of cells. The software layer is however not taken into account.

More recently, Henkel et al. [7] show how the design flow of a generic electronic system can be improved to implement what they call multi-layer dependability. This approach analyzes the system hierarchically following the

way faults propagate from the bottom layer (devices and circuits) up to the application layer. At each layer DSE is used to select a set of protection mechanisms to cover faults that escaped the protection mechanisms implemented in the lower hierarchical level. The approach is biased toward low-level protection mechanisms that are applied first during DSE. Therefore, it does not allow to fully explore the space of possible design options.

Cheng et al. [8], [19] for the first time propose an impressive and massive simulation campaign showing how combinations of different protection techniques overall work together across different hardware designs (two processors SPARC Leon3 and Alpha IVM), different software benchmarks and hundreds of cross-layer combinations of protection techniques. In order to do so, the authors injected 9 million flip-flop soft errors into the RTL of the processor designs using three BEE3 FPGA emulation systems and also using mixed-mode simulations on the Stampede supercomputer. This has a very high value to give a generic picture of how cross-layer reliability techniques can work, providing results based on very accurate simulations. However, performing the same simulation campaign for every new product in the early stages of the design might not be affordable. In these stages, design decisions should be taken based on reasonably accurate models but still consuming affordable computing resources. The goal is to support the designers in these early decisions, demanding the final accurate assessment of the reliability of the system to the late stages of the design.

3 System level modeling

In this paper, the system under analysis is modeled using a component-based Bayesian reliability model. Section 3.1 introduces the basic characteristics of the model that was first introduced and validated in [9]. This information is required to understand this paper (the reader may refer to [9] for additional details). Section 3.2 instead focuses on the extensions of the model introduced in this work to enable its use in the proposed DSE framework.

3.1 Reliability modeling

Fig. 1-A shows an example of system modeling. The system, denoted with S, is modeled using an extended Bayesian network defined as:

$$S = (N, E, \Theta, P) \tag{1}$$

where:

- $N = \{n_1, n_2, \cdots, n_m\}$ is the set of network nodes. Each node models a software/hardware component of the system. Each node is associated with a set of states indicating error or error-free conditions for the node (e.g., the L1 cache can be error free, it can be affected by a single bit-flip or by a double bit-flip).
- $E = \{(n_i, n_j) \in N \times N\}$ is the set of arcs. Each arch defines a temporal or physical relation between two components (e.g., a failure of a component may influence the state of other components).
- $\Theta = \{\theta_1, \theta_2, \cdots, \theta_m\}$ is a set of Conditional Probability Tables (CPT), one table for each node. The CPT θ_i of node n_i defines the probability of n_i to be in a given state, conditioned on the state of its parent nodes.
- $P = \{p_1, p_2, \cdots, p_m\}$ is a set of optional tables of generic parameters. These parameters can be associated with each node of the network to characterize the related component (e.g., area, power consumption, etc.). Any parameter that can influence the design decisions can be included in this table as far as it can be measured for each component.

Following a cross-layer reliability design, the model is hierarchically organized by grouping nodes into four stacked domains.

The *technology domain* (TD) models the physical layer of the system. Nodes in this domain model how soft errors distribute to the different components of the system. This is a function of the hardware fabrication technology used to build each component, which affects its soft error rate (e.g., 16nm Bulk CMOS SRAM, 16nm FinFET SRAM, etc.).

The *hardware domain* (HwD) models the hardware architecture of the system. Nodes in this domain represent the hardware blocks used to build the system (e.g., CPUs, GPUs, memories, accelerators, custom IP cores, etc.). The granularity of the hardware description depends on both the level of detail the designers need for the reliability analysis and the degree of freedom in selecting the components. Complex components, such as microprocessors, can be either modeled as a single node or split into clusters of nodes modeling the different subcomponents (e.g., register files, processing units, etc.).

The *software domain* (SwD) models the software architecture in terms of software modules. The interface between the HwD and the SwD is modeled by a set of special nodes called Software Fault Models (SFMs). The SFMs, first introduced in [20], model the effect of a hardware fault on the execution of an instruction of the Instruction Set Architecture (ISA).

The *system domain* (SD) finally defines the observation points where the behavior of the system in the presence of faults can be evaluated and properly classified.

As explained in [9], the partitioning of the model into different domains is mainly required for an efficient computation of the CPTs characterizing each node of the model. This activity is a time consuming task of the modeling phase. The analysis of each layer can therefore focus on the peculiarity of the layer itself, thus using dedicated and optimized tools. Details on the tools exploited in this work to perform experiments are provided in section 5.

The proposed model allows us to compute different system level reliability metrics such as the Architecture Vulnerability Factor (AVF), the Failures in Time (FIT) and the Mean Time Between Failures (MTBF). In this paper, we exploit the AVF as preferred reliability metric. The AVF first introduced in [21] is the probability that a fault in a hardware structure will result in a visible error in a program's final output. The AVF is a cross-layer metric that looks at how the software reacts to hardware faults. In a complex system the different domains contribute in a different way to the global AVF. The goal of the reliability model exploited in this paper is to model how the different components of the system

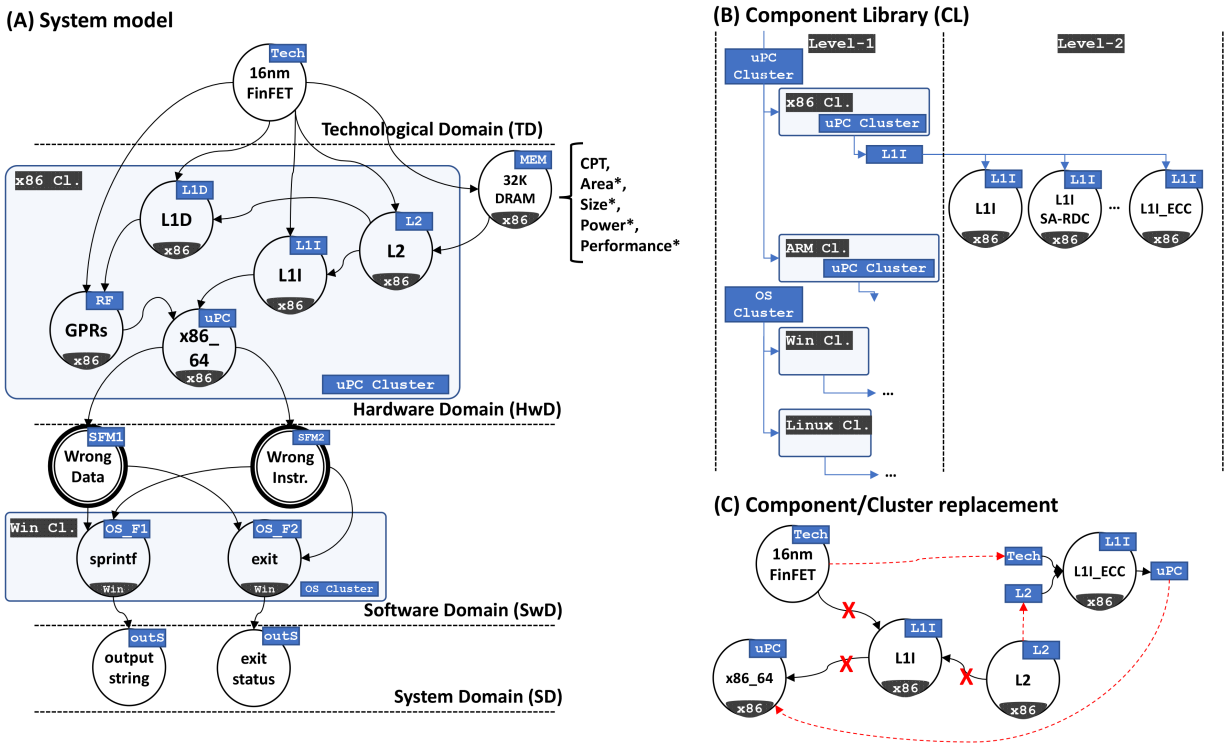

Fig. 1. System modeling. (A) *Bayesian model of the systems*: nodes are organized into domains and each node is characterized by a Conditional Probability Table (CPT) plus a set of optional parameters (e.g., area, size, power, performance) that can be used during the DSE process, (B) *design alternatives*: for each component (node) or cluster of components different implementations are defined, thus forming a library of design alternatives, (C) *component replacement*: showing how a component can be replaced with a different implementation in the model.

contribute to the overall AVF in order to quantify this contribution. In particular, the use of a Bayesian model is powerful to express how faulty conditions cross between the different components and layers. The AVF can be estimated by computing the belief of the output nodes to be in a failure state using Bayesian inference [22]. Moreover, the analysis of the beliefs on the state of each node can be used during the DSE to identify weak components to be protected as will be described later in the paper.

3.2 System modeling for optimization

The model proposed in section 3.1 is powerful when it is time to analyze the reliability of a single system. However, it lacks the possibility to express a list of design options for each component that can be evaluated in a DSE process.

This paper therefore proposes to enhance the model presented in section 3.1 with the definition of a *component library* (CL) including different implementations or configurations of the components of the target system (see section 1-B). Each element in this library must be fully characterized with its CPT and parameters.

The way the CL is populated depends on the target application domain and impacts the DSE process. In custom embedded applications, the designer has usually access to the design of the hardware architecture and the dedicated software applications that are executed. In this scenario, a large set of custom hardware and software components implementing different fault-tolerance techniques can be included in the library and analyzed as potential design options. Differently, in general purpose systems, in which the hardware architecture is designed to accommodate several software applications, the number of design options for the hardware domain is more limited. Nevertheless, general purpose hardware, such as complex microprocessors, usually offer several configuration knobs to play at run-time. These knobs enable the activation or deactivation of a set of built-in fault-tolerance mechanisms (e.g., ECC for memory structures), thus enabling run-time adaptation to different software requirements. These knobs can therefore be exploited to generate different elements to include in the CL, which is then used to perform DSE for different software applications.

To support the optimization process, components of the system (i.e., nodes) are organized into a k-levels hierarchy of *clusters*, following the hierarchical architecture of the real system.

Def. 1. A *cluster* node at the hierarchical level k of the CL is a component that can be split into a group of sub-components at level $k+1$ representing a fine-grained representation of that node.

As an example, Fig. 1-A defines a 2-level hierarchy that includes at the first level two clusters: the first labeled as `uPC Cluster` grouping all nodes modeling the components of the microprocessor, the second named `OS Cluster` grouping all nodes modeling the operating system functions and modules used by the application software. Different implementations of each cluster are available to the designer as reported in section 1-B and can be considered during

the DSE process. Using the concept of clusters, multiple levels of hierarchy can be defined. At the bottom of this hierarchy, different implementations of single components are defined in the library. For example, section 1-B defines three implementations of the L1 instruction cache in the x86 cluster: the basic implementation without any fault tolerance mechanism and two implementations of the same hardware component supporting different fault tolerance mechanisms.

For simplicity, hereinafter, we will use the generic term cluster to identify any generic node of the CL even if this node is a leaf of the hierarchy and is not further split into sub-nodes.

Using the proposed system model and the related library of components, the DSE framework presented in this paper is able to automatically analyze different versions of a target system. Each version is created starting from a reference implementation and replacing single components or clusters of components with alternative implementations available in the CL. The goal of this process is to efficiently explore the design space trying to identify architectures that optimize multiple design parameters, including reliability, performance, hardware/software resources and power. The process to replace a cluster or a component with an alternative implementation is graphically depicted in section 1-C. The enhanced model defines for each cluster a set of labelled input and output connectors. The labels are used to define how a cluster can be connected to the remaining portions of the model.

4 Design Space Exploration methodology

The proposed DSE heuristic is based on an extension of the extremal optimization theory [11].

4.1 Extremal optimization theory

The *extremal optimization* (EO) theory is a generic optimization theory inspired by nature's self-organizing processes. It is well suited to solve multi-objective optimization problems. In its basic formulation, the EO enables to explore a multi-variate search space. It tries to optimize a global objective function making local changes to selected local variables of the system that involve coevolutionary avalanches on the global objective function. It explores the search space trying to avoid sub-optimal solutions, thus driving the optimization towards real local-optima, as demonstrated in several publications [23], [24], [25], [26], [27].

The EO is an interesting candidate to build a multi-objective DSE heuristic for cross-layer reliability. In fact, in particular in the early phases of the design, designers need support to evaluate how the application of different combinations of local protection techniques in selected components of the system (modifications of local variables) affect the global characteristics of the system (the effect of the local modifications on a global system level cost function). The performance of EO based heuristics overcome the efficiency of classical simulated annealing and genetic algorithms providing competitive accuracy [11], [28].

Another important characteristic that makes EO interesting in the cross-layer reliability domain is that, unlike genetic algorithms [29], which work with a population of candidate solutions, EO evolves a single solution and makes local modifications to the worst components of the solution. This is an important characteristic when considering the complexity of the Bayesian model presented in section 3.1. The reliability model of a real system may include thousands of nodes with their related CPTs. Working on big populations of such models can rapidly become computationally and memory expensive. Another interesting feature is that the EO process highly resembles the approach expert designers would use in manual DSE. Improvements to the characteristics of the system are in general implemented by selectively identifying critical components and replacing them with selected alternatives representing potential improvements toward the local-optimum. This is very different from other evolutionary techniques that look at combining "good" solutions in the attempt of improving the quality of the population. Moreover, EO is well suited for the optimization of multi-objective cost functions [24], [30], [31]. This is important in our case, since the DSE process must take into account several design parameters such as reliability, area, power, performance, etc.

Finally, the EO is particularly effective in solving optimization problems, where near-optimum solutions are widely dispersed and separated by barriers in the search space [27]. This is a typical case of the particular DSE problem we are facing. Macro changes in the design (i.e., replacements of full clusters of components) introduce high diversity in the generated systems that translates into a very sparse search space.

The main limitation of the basic formulation of the EO theory is that it is not intended to deal with a hierarchical organization of the search space as the one introduced in section 3.1. Therefore, the DSE algorithm proposed in this paper, described in section 4.3, takes advantage of the basic principles of this theory, but extends it to analyze a hierarchically organized search space in which both single variables and clusters of variables can be modified at the same time during the exploration process.

4.2 Definitions and notation

This section introduces some basic definitions and notations required to describe the proposed DSE strategy.

Def. 2. The *design space* $\Omega = \{S_0, S_1, \cdots, S_h\}$ is the set of all possible system's implementations of the target design. Every implementation is a system description defined according to (1).

S_0 represents the *reference implementation*, i.e., the initial design of the system that must be optimized (usually it does not include any fault tolerance mechanism). Starting from this implementation, the DSE strategy generates new implementations by selectively replacing worst clusters (nodes or groups of nodes) based on the alternatives available in the CL.

The hierarchical organization of the CL allows us to introduce a hierarchical concept of distance between system implementations.

Def. 3. The k-level neighborhood of a system implementation $S \in \Omega$, denoted as $N_k(S)$, is the set of implementations that can be created from S by replacing a single

cluster of the system placed at the k^{th} hierarchical level of the CL.

The k-level neighborhood hierarchically partitions the design space and therefore the DSE process. Informally, in a 2-level CL as the one reported in Fig. 1-B, we can identify two search levels. At a high level, we have different implementations that differ for macro changes of the system due to replacements of clusters of components (e.g., changing the full microprocessor architecture, or the full operating system architecture). At a fine grained level, selected nodes inside the clusters can be replaced based on the available alternatives to fine tuning the optimization of the system. Managing a hierarchical exploration is one of the main contributions of the DSE algorithm described in section 4.3. The starting hierarchical level can be freely selected and the algorithm is able to move up and down in the hierarchy during the design exploration trying to find the best solution (see section 5 for experimental result showing the impact of the initial hierarchical level on the optimization process).

Two different types of cost functions must be defined to implement the proposed DSE strategy.

Def. 4. The *global cost function* $C(S)$ with $S \in \Omega$ is any generic function defined on any variable of S (i.e., N, E, Θ, P in (1)) that allows for comparing two different implementations of the same system.

This function is used to monitor the progress of the DSE process. In general, the proposed algorithm supports any generic cost function defined over any variable in S. The AVF of the system introduced in section 3 is an example of cost function that can be exploited whenever reliability is the only factor to consider during DSE. More complex functions are instead required for multi-objective DSE. A description of the cost functions implemented for this paper is reported in the next section together with a detailed description of the optimization algorithm.

Def. 5. The *fitness* of a cluster (cls) of a system S, denoted as $\lambda(cls, S)$, (with $cls \subseteq N$) is any generic function that permits to measure the contribution of that component to the global cost function $C(S)$.

The fitness of a component is the criterion used during the DSE to select the components or the clusters of components to replace.

4.3 Optimization algorithm

Alg. 1 describes the proposed DSE algorithm. The main inputs of the algorithm are: the reference implementation of the system (S_0), the available component library (CL) and the level of the CL at which starting the design exploration (startlevel). The algorithm returns an improved implementation S_{best} and its related cost $C(S_{best})$.

The DSE process is an iterative process (lines 5-21). At a high-level of abstraction, at each iteration a new implementation of the system is generated by replacing one of the clusters of the current implementation of S with an alternative version from CL. The cost of this new implementation is evaluated to understand whether the introduced change leads toward a better implementation of the system or not. This iterative exploration process stops according to two different conditions (line 21):

Algorithm 1 Cross-Layer multi-objective DSE algorithm

Input: S_0, CL, MAX_ITER, stop(·), startlevel (default k)
Output: S_{best}, $C(S_{best})$

```
1:  lev = startlevel;
2:  iter_no = 0
3:  S = S_0
4:  S_best = S_0
5:  repeat
6:     λ_w=1;
7:     cls_w = ∅
8:     for each cluster cls at level lev do
9:        if λ(cls, S) ≤ λ_w then
10:          λ_w = λ(cls, S)
11:          cls_w = cls
12:       end if
13:    end for
14:    Generate S' ∈ N_k(S) by selecting an alternative
       implementation of cls_w ∈ CL
15:    if C(S') < C(S_best) then
16:       S_best = S';
17:    end if
18:    lev = nextLevel(lev, S', S_best)
19:    S = S'
20:    iter1_no = iter1_no + 1;
21: until iter_no < MAX_ITER and stop(S_best) not true
22: return S^best and C_k(S^best)
```

1) the number of iterations (iter_no) reaches a maximum limit (MAX_ITER),
2) a *contract* on the identified implementation of the system is satisfied.

The contract is provided as an input to Alg. 1 in the form of a generic *stop(·)* function. In our implementation, we have defined a stop function that terminates the simulation when the estimated AVF of the system is lower than a user defined threshold (i.e., the system has reached the target reliability constraint), but other conditions can be easily defined. The first stop condition allows to bound the duration of the DSE process, whereas the second allows to define the goal of the exploration.

Lines 6-13 describe the process used by the algorithm to identify the cluster to replace at each iteration. To understand how this process works, it is important to recall that the system is organized into a k-level hierarchy of clusters (see Fig. 1). At a given iteration the optimization process works at one of these k hierarchical levels. The hierarchical level sets the granularity of the replacement process. Let us consider the example of Fig. 1. If the algorithm works at level 1, replacements of clusters take place at this level of the CL. In the example of Fig. 1, this means that the algorithm can try to replace the `uPC Cluster` or the `OS Cluster` in the design. If the algorithm works at level 2, components composing the level-1 clusters can be replaced, e.g., L1 Instruction cache. Alg. 1 sets the initial level at line 1. If not specified by the user, the initial level is set to k. The idea behind this choice is that we want to investigate first if local modifications to the design allow to reach the selected goal. This is motivated by the fact that we believe designers would privilege solutions with reduced modifications to the

original design. Nevertheless, this option is not mandatory. By changing the starting level, the designer can privilege the modification of bigger clusters of components (e.g., use a completely different microprocessor model rather than trying to optimize the single parts of a microprocessor).

In lines 6-13 the fitness of every cluster belonging to the selected hierarchical level is evaluated for the current system implementation S in order to identify the cluster with the worst fitness. A new implementation of the system S' is then generated by replacing the cluster with the worst fitness with a random alternative implementation from CL (line 14). If the cost of this new implementation ($C(S')$) is lower than the one of the best implementation ($C(S_{best})$), then the new version of the system is selected as current best architecture $C(S_{best})$. Regardless the fact that S' improves the best architecture or not, line 19 updates the current solution to make sure the search process will move on evaluating a new alternative.

At the end of each iteration, the algorithm evaluates whether the exploration must continue at the current hierarchical level or it must move up or down. This decision is taken at line 18 by the *nextLevel* function whose implementation is reported in Alg. 2.

The function computes the next level by analyzing, for each hierarchical level, the number of iterations that have not produced improvements to the system. This is implemented using a set of counters (*count_worst[k]*). In case the new generated system implementation S' has introduced an improvement ($\delta \leq 0$ - line 1), the hierarchical level remains the same and the counter for the level is reset (line 13). Otherwise, the algorithm analyzes the counter of the layer (lines 3-11). If it is higher than a user defined threshold T (lines 4-5), it means we are not able to improve the system at the current level and therefore the algorithm moves up to a higher level. Otherwise (lines 7-10), the counter of the level is incremented and the algorithm tries to move down in the hierarchy to search if local changes can further improve the current solution. As explained before, the idea behind this strategy is to privilege the lower levels of the hierarchy that correspond to small changes in the system moving up only when really required.

Algorithm 2 nextLevel function

Input: k, S', S_{best}
Output: next k

```
1:  δ = C(S') − C(S_best)
2:  if δ > 0 then
3:     if count_worst[k] > T then
4:        count_worst[k]=0
5:        k = k - 1
6:     else
7:        count_worst[k]++
8:        if k < CL_LEVELS then
9:           k = k +1
10:       end if
11:    end if
12: else
13:    count_worst[k]=0
14: end if
15: return k
```

The current implementation of Alg. 1 comes with two different exploration strategies characterized by different cost and fitness functions. In details, these strategies are suitable for both reliability only optimization and multi-objective optimization and they are described in the next section.

4.3.1 DSE for best reliability

The design space is explored to identify architectures that maximize the reliability without taking into account other design parameters such as hardware area, software size, execution time, power consumption, etc. In this case the *cost function* is defined as the AVF of the system, i.e., the probability of a system failure given a hardware fault.

When using this cost function, it is important to point that, in principle the optimization of the system is not bound. It is always possible to add additional protections mechanisms obtaining a system with higher resilience to errors. This can continue up to a level at which all components are protected. Nevertheless, when the DSE aims at reaching a target reliability level, this simple cost function enables to stop adding new protections as soon as the target AVF is reached, limiting the number of protected components. This simplifies the design even when when other design dimensions are not relevant to the designer.

Resorting to the Bayesian model defined in section 3.1, the hypothesis that a fault enters the system is emulated by propagating the information about fault causes (i.e., the failure rates of TD nodes) through the whole Bayesian model. The AVF is computed as the posterior probability that at least one of the nodes of SD is in a failure state, using the Bayesian inference theory [22].

In a similar way, the *fitness* of a component can be computed by conditioning the Bayesian model with the hypothesis that the component is in a failure state and computing the posterior probability that the system fails given this event (i.e., at least one of the nodes of SD is in a failure state). When working with a cluster this probability is computed separately for each component of the cluster and the fitness is computed as the average probability over all components of the cluster. This approach considers that all elements of the cluster equally contribute to the fitness. This is not always the case especially when the usage frequencies of the different elements is significantly different. If this information is available from simulations profiling the application, it can be used to weight the average obtaining a more precise fitness.

4.3.2 DSE for multiple objectives

The design space is explored to identify architectures that optimize different design dimensions. For this work we selected the following five design dimension, however others can be easily considered:

- AVF
- hardware area,
- software size,
- execution time,
- power consumption.

To compute the cost function, together with the AVF of the system, computed using the same approach described in

section 4.3.1, the percentage increment of the remaining four parameters w.r.t. S_0 is computed. Particular attention is paid when replacing a component, since each change may affect the behavior of other components. For example, if the replacement of a hardware component introduces some extra execution time, the execution time of all software components executed on top of that specific hardware component must be modified accordingly. The five contributions are then combined with a simple weighted sum. The designer is free to assign the weights of each contribution depending on the DSE goals. The weighted sum allows to account for all parameters during the exploration. In this way, the algorithm can identify design options that globally improve not only the reliability but also other design dimensions.

Similarly to the global cost function, the fitness function performs the same weighted sum but only considering percentage increases of the nodes of the considered cluster.

5 Experimental Results

This section reports results obtained by the application of a C++ implementation of the proposed DSE algorithm to a set of realistic electronic systems.

5.1 Experimental setup

Experiments were conducted by performing DSE for a set of microprocessor based systems running software applications on top of the Linux operating system. The DSE focused on the analysis of different hardware and software architectures of the system, i.e., hardened technologies were not considered as viable design options. We considered two real microprocessor architectures:

1) ARM Cortex®-A15 (ARM): a high-performance ARMv7-A processor used in a variety of premium mobile and infrastructure applications.
2) Intel-like i7® skylake architecture (SKYLAKE): a 64-bit microarchitecture that brings high performance and reduced power consumption.

Based on this hardware architecture, we analyzed 10 different systems, each running a different software application taken from the MiBench suite [32]. MiBench applications have been widely used in reliability studies [20], [33], [34], [35], [36], [37], [38], [39]. They combine a wide range of common workloads/algorithms with relatively small datasets, which effectively translate to small execution time. The following applications were selected: (1) Susan Smooth (susan_s), (2) Susan Edges (susan_e), (3) Susan Corners (susan_c), (4) Quick sort (QSort), (5) String search (ssearch), (6) Secure Hash Algorithm (SHA), (7) JPEG decode (DJPEG), (8) JPEG encode (CJPEG), (9) AES decode (AES), and (10) Fast Fourier Transformation (FFT).

Every system was modeled as described in section 3.1. In order to build the model and to compute the CPTs of the different nodes we resorted to the tool-suite proposed in [9], whose main concepts are summarized in Fig. 2. To characterize the CPTs of the hardware domain for the target microprocessors, we resorted to microarchitecture-level fault injection through the GeFIN tool [40], which is based on the Gem5 micro-architectural model [41]. Using this tool we were able to analyze the effect of soft errors in five of the main memory arrays of the microprocessor, whose characteristics are summarized in Table 1. For every simulation, we tracked the propagation of an error in a selected structure to another one, or its propagation to the software domain consisting in the commit of an instruction affected by the fault to the architectural state (i.e., a visible SFM was detected). Results of this fault injection campaign were used to build the different CPTs of the HwD nodes. A similar approach was used to build the CPTs of the software domain focusing on how the occurrence of a SFMs in a given function affected the behavior of the software. To perform this analysis, we performed SFMs injection experiments through the LIFILL tool presented in [9]. LIFILL injects SFMs into a LLVM (Low Level Virtual Machine) compiled version of the target application. The LLVM instruction set is independent of the target machine and allowed us to analyze the application independently of the target hardware. Results from this analysis were used to build both the architecture of the software domain (by automatically extracting the function call graph of the application), and the related CPTs. Interested readers may refer to [9] for a detailed description of the toolchain. The model and the tools were validated comparing the obtained AVF estimations for the reference implementation of all benchmarks with those computed with full fault injection campaigns on micro-architectural models of the considered microprocessor, which are often used for reliability analysis in the early design phases of a system. On average we obtained differences in the order of one percentage point. In both cases the software application was exercised with a relevant workload. For each node of the model, we performed about 2000 injection campaigns corresponding to 2.88% error margin and 99% confidence level [42]. It is worth to stress here that other toolchains modeling the hardware/software architecture at different abstraction levels can be used as well to fill the CPTs of the proposed model.

The effects of the protection techniques on the other design parameters are instead represented in the component library as variations with respect to the reference implementation. This is a simplified model that provides to the DSE engine qualitative information regarding the impact of a protection technique requiring low computational resources. Nevertheless, if efficient models with higher accuracy are available, they can be plugged in the DSE engine with low effort, thus improving the accuracy of the obtained results.

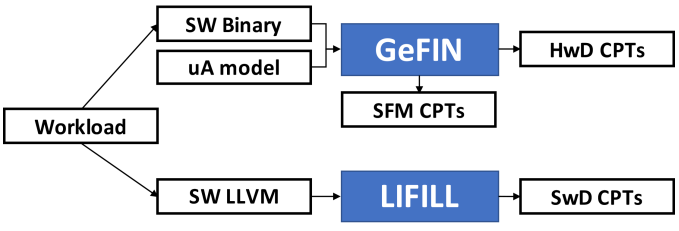

Fig. 2. High-level view of the toolchain used to build the reliability models of the target systems.

The CL was organized in two hierarchical levels. The first level consisted of the two microprocessor clusters, each comprising the five memory arrays considered in the study.

TABLE 1
Hardware parameters for the target microprocessors

	ARM	SKYLAKE
L1 Instruction Cache	32KByte	32KByte
L1 Data Cache	32KByte	32KByte
L2 cache	1MByte	1MByte
physical registers	128 32-bit registers	168 64-bit registers
Load/Store Queue	16 32-bit entries	72 64-bit entries

The second hierarchical level implemented various state-of-the-art fault tolerance mechanisms that can be applied to hardware or software components.

When selecting the set of protection techniques used to show the capability of the proposed DSE technique, it is important to remark that the main driver was the availability in the literature of an estimation of the following parameters: (1) soft error masking probability, (2) hardware area overhead, (3) software size overhead, (4) software execution time overhead, (5) power consumption overhead. Moreover, we searched for techniques ready to apply to single components in isolation. Table 2 summarizes the selected hardware fault-tolerance techniques. Overall, they improve tolerance to soft errors by: (i) modifying the circuit (LEAP, DICE and LEAP-DICE) [43], (ii) monitoring the data (SA-RDC) [44], or (iii) adding error correction codes (all ECC and Self-Immunity[1]) [45]. A total of 8 techniques were selected. In a similar way, Table 3 reports the selected software implemented fault tolerance techniques. We selected 16 different techniques. Most of them (VARx techniques) are different combinations of variable duplication and cross checking validation techniques [46]. We also included a fault tolerance technique based on control and data flow assertions [47].

We would like to remark again that all data provided in Table 2 and Table 3 are not intended to compare the different techniques in order to identify which is superior. The experimental setup of the different publications is different thus preventing this type of comparison. Differently, they represent a set of realistic inputs for our DSE engine exploited to drive the exploration process.

As reference implementation we considered systems based on the SKYLAKE microprocessor not implementing any fault tolerance mechanisms. Both DSE strategies described in section 4.3 were tested.

For each considered system the DSE algorithm was executed with a limit of 500 iterations without setting any stop condition based on the AVF of the system. This long simulation allowed us to clearly understand the dynamics of the optimization and to stress the limits of the optimization process. To deal with the stochasticity of the optimization process, every experiment was repeated 30 times selecting the best implementation over the 30 repetitions.

The characteristics of the systems optimized using the proposed DSE methodology were compared with those obtained by implementing three state-of the art solutions available in our library. More specifically, we considered: LEAP-DICE [43], the most effective hardware protection technique of the library, VAR1 [46], the most effective software protection mechanism of the library, and the combination of LEAP-DICE+SIFT (the LEAP-DICE technique for hardware components and the SIFT technique for software components) that was identified in [8] as one of the most promising cross-layer combinations of protection techniques.

1. Being the numbers provided in [45] application dependent, we used here average results provided in the paper.

5.2 Results and discussion

Table 4 compares the AVF of the reference implementation of the considered systems (AVF BASE) with the AVF obtained by adding protection mechanisms according to different approaches.

The AVF REL ONLY column shows the capability of the proposed DSE for best reliability to identify cross-layer combinations of protection mechanisms able to significantly increase the reliability of the target system, i.e., to decrease its AVF. Looking at the obtained results, one can see that the AVF is always decreased by several orders of magnitude. This result is not surprising. When optimizing for best reliability, it is always possible to add additional protection techniques obtaining a system with higher resilience to errors. Nevertheless, as will be discussed in Fig. 3, the gained protection is obtained working on a limited number of components, thus gaining on the global characteristics of the design.

It is also interesting to report that, in 7 out of the 10 benchmarks (DJPEG, FFT, QSORT, SHA, SUSAN_E, SUSAN_C, SUSAN_S), the DSE algorithm identified the ARM architecture as the better microprocessor option with respect to the SKYLAKE architecture that was used as reference implementation. This suggests that, for these specific applications, the ARM architecture is inherently more resilient to soft errors, enabling to reach high protection with lower effort. This is not an indication that the ARM architecture is superior to the SKYLAKE architecture. The results are application dependent and show how different applications may benefit from the characteristics of different hardware architectures, thus motivating the use of an automatic DSE algorithm such as the one presented in this paper.

When comparing the results obtained with the DSE for best reliability with those obtained applying state-of-the-art cross-layer reliability techniques, in most of the cases the architectures identified by our algorithm provide a lower AVF, i.e., better reliability. In only two cases (SHA and SUSAN_E) LEAP-DICE+SIFT provides a lower AVF. This is probably due to the size of the search space that makes DSE complex for some benchmarks and probably requires more iterations. Nevertheless, as will be discussed later, architectures obtained with the proposed DSE strategies apply fault tolerance techniques to just a subset of the system components, thus minimizing the impact on other design dimensions.

Looking at Table 4, it is also interesting to note that when the DSE algorithm is executed considering the multi-objective optimization (Table 4, column AVF MULTI-OBJ), we still see significant improvements in the reliability of the system. In detail, AVF MULTI-OBJ is characterized by lower values with respect to LEAP-DICE and VAR1 (with the exception of SUSAN_S), similar results to LEAP-DICE+SIFT in some cases, but higher AVF if compared to DSE for best reliability. This trend is due to the fact that this strategy tries to properly weight the reliability improvement

TABLE 2
Hardware fault tolerance techniques

Technique	Description	Reliability Improvement (%)	Extra Time (%)	Extra Power Consumption (%)	Extra Hardware Area (%)	Ref.
4 ECC	ECC applied to 4 registers over 32	+40%	+6%	+1%	+15%	[45]
16 ECC	ECC applied to 16 registers over 32	+91%	+8%	+2%	+22%	[45]
FULL ECC	ECC applied to all registers	+100%	+9%	+3%	+28%	[45]
Self-Immunity	Parity code for portion of registers	+91%	+4%	+0.20%	+11%	[45]
SA-RDC	Self-Adaptive caches using monitoring features	+99.98%	+7.40%	+43%	+0.39%	[44]
LEAP	Flip-Flop layout technique	30%	10%	10%	10%	[43]
DICE	Flip-Flop layout technique	90%	0%	60%	100%	[43]
LEAP-DICE	Flip-Flop layout technique	100%	2%	63%	100%	[43]

TABLE 3
Software implemented fault tolerance techniques

Technique	Description	Reliability Improvement (%)	Extra Time (%)	Extra Power Consumption (%)	Extra Software Size (%)	Ref.
VAR1	Variable Duplication and Cross-check mechanisms	+95%	+70%	+66%	+66%	[46]
VAR1+	Variable Duplication and Cross-check mechanisms	+95%	+66%	+64%	+64%	[46]
VAR1++	Variable Duplication and Cross-check mechanisms	+94%	+70%	+60%	+60%	[46]
VAR2	Variable Duplication and Cross-check mechanisms	+95%	+77%	+74%	+74%	[46]
VAR2+	Variable Duplication and Cross-check mechanisms	+95%	+73%	+70%	+70%	[46]
VAR2++	Variable Duplication and Cross-check mechanisms	+94%	+68%	+65%	+65%	[46]
VAR3	Variable Duplication and Cross-check mechanisms	+94%	+42%	+45%	+45%	[46]
VAR3+	Variable Duplication and Cross-check mechanisms	+94%	+37%	+41%	+41%	[46]
VAR3++	Variable Duplication and Cross-check mechanisms	+93%	+32%	+36%	+35%	[46]
VAR4	Variable Duplication and Cross-check mechanisms	+91%	+32%	+36%	+36%	[46]
VAR4+	Variable Duplication and Cross-check mechanisms	+91%	+27%	+33%	+33%	[46]
VAR4++	Variable Duplication and Cross-check mechanisms	+90%	+21%	+27%	+27%	[46]
VAR5	Variable Duplication and Cross-check mechanisms	+87%	+27%	+28%	+28%	[46]
VAR5+	Variable Duplication and Cross-check mechanisms	+84%	+24%	+25%	+25%	[46]
VAR5++	Variable Duplication and Cross-check mechanisms	+78%	+17%	+20%	+20%	[46]
SIFT	Fault Tolerance via control and data flow assertions	+95%	+40%	+20%	+20%	[47]

TABLE 4
AVF Comparison across state-of-the-art reliability techniques and optimization strategies

	AVF BASE	AVF LEAP-DICE	AVF VAR1	AVF LEAP-DICE+SIFT	AVF REL ONLY	AVF MULTI-OBJ
AES	8.62E-02	8.62E-05	3.94E-04	3.90E-07	3.36E-09	1.58E-05
CJPEG	1.10E-01	1.10E-04	9.74E-02	9.73E-05	2.45E-06	2.75E-06
DJPEG	6.71E-02	6.25E-05	5.28E-02	5.30E-05	4.61E-07	1.16E-06
FFT	1.48E-01	1.31E-04	3.07E-02	3.07E-05	1.16E-06	1.14E-05
QSORT	6.99E-02	2.05E-05	5.78E-03	5.78E-06	4.78E-07	1.34E-06
SHA	7.52E-02	6.92E-05	6.43E-06	6.00E-09	8.09E-07	8.09E-06
SSEARCH	5.48E-02	5.48E-05	4.09E-04	4.10E-07	5.78E-09	2.13E-06
SUSAN_C	8.60E-02	7.66E-05	3.61E-05	4.00E-08	1.83E-08	4.37E-06
SUSAN_E	3.10E-02	2.92E-05	4.48E-06	4.00E-09	4.15E-07	1.39E-05
SUSAN_S	3.14E-02	2.69E-05	1.84E-05	2.00E-08	7.03E-09	5.44E-05

with the overhead introduced by the use of different fault tolerance mechanisms, as described in section 4.3.2. The multi-objective cost function used in this experiment (see section 4.3.2) weights the reliability as the 10% of the total value of the cost function, while the remaining 90% is equally shared among the other four design dimensions.

As discussed before, improving the reliability of a system always comes at a price. Fig. 3 gives a quantitative representation of the overhead for different design dimensions (i.e., execution time, power consumption, hardware area and software size overheads) when comparing different system implementations with the base implementation.

Looking at DSE for best reliability (REL ONLY), reliability improvements have a significant impact on the other design dimensions. This confirms that, performing DSE using a single objective is feasible, but probably it is not the best option. Fig. 3 also gives an indication on how the improvement is achieved. The HW area and SW size overheads clearly indicate that in all cases the best reliability is achieved by a combination of hardware and software fault tolerance mechanisms, thus confirming the benefit of applying cross-layer techniques to design reliable systems.

Fig. 3 shows that, despite the DSE for multiple objectives (MULTI-OBJ) is unable to reach the same reliability level of the REL ONLY approach, it is able to significantly minimize the impact of the introduced fault-tolerance mechanisms on the other design dimensions. The overheads of the execution time, power consumption, hardware area and software size are significantly reduced if compared to the other approaches. In detail, looking at the average on all benchmarks, the overhead on all considered dimensions is always lower than 25%. This result shows that, thanks to a careful analysis, systems can reach high reliability levels without significantly penalizing the other aspects of the

design. Moreover, specific optimization goals (e.g., low area or low power) can be achieved by modifying the weights of the cost function.

When comparing the proposed DSE techniques with LEAP-DICE+SIFT (yellow squares), Fig. 3 shows that LEAP-DICE+SIFT always introduces a higher overhead in terms of HW area because it is not applied selectively to the most critical hardware components. The figure highlights that in most of the cases the REL ONLY DSE (blue bullets) privileges SW protection mechanisms, which lead to a lower power consumption (see Table 2 and Table 3) even if they increase the size of the software. Looking a the execution time overhead, the two techniques are close. Even more interesting is the set of overheads provided by the MULTI-OBJ DSE approach (red bullets). Looking at Fig. 3, the overheads for this technique are far better then any other presented solution.

When analyzing the capability of DSE heuristics such as the one proposed in this paper, it is important to evaluate the impact of different parameters on the result of the algorithm.

To show the impact on the DSE of the characteristics of the reference implementation of the system (i.e., the starting point of the DSE process), we performed ten experiments on the FFT benchmark starting the DSE process from systems with increasing number of protected components. The starting systems were taken from the intermediate steps of the REL ONLY DSE process performed on the unprotected version of the system. The results presented in Table 5 for REL ONLY DSE show that, on average, the target AVF is not influenced by the selected initial system. What is instead influenced is the number of iterations required to identify the optimized system. The negative value is expected since, by design, the considered systems start from an increased number of protected component. Therefore, the DSE process becomes increasingly simple. Results for MULTI-OBJ DSE are instead more difficult to read. The most important result is that the cost function ($C(S)$ column) shows small variations confirming that the DSE algorithm always moves toward an optimized solution. The AVF that represents a significant contribution to the cost function also shows small variations. Regarding the remaining parameters, that are equally accounted in the cost function, we see higher fluctuations (some positive and some negatives). Nevertheless, this is not a negative result. It simply shows that there are several solutions that overall have a similar cost but may privilege in some cases one of the parameters against another. In this case, overall the DSE process requires more steps since the starting systems have increased reliability but not necessarily optimize the other design constraints.

Another important parameter affecting the behavior of the proposed DSE algorithm is the startlevel (line 1 of Alg. 1). To analyze its impact on the DSE process, we performed DSE for all benchmarks starting with the two possible values of the proposed experimental setup (i.e., startlevel$\in 0, 1$). Table 6 shows how the different parameters change by modifying the value of startlevel. As for the previous case, it is important to highlight that the AVF for REL ONLY DSE, and $C(S)$ and AVF for MULTY-OBJ DSE remain almost constant. This again confirms the capability of the algorithm to move toward a good solution regardless the starting condition. In MULTI-OBJ DSE, since multiple architectures can lead to similar costs, the remaining design parameter show higher variability with particular regard to the number of iterations required to reach the optimized system.

To improve the analysis of the capability of the presented DSE algorithm, we improved the library of available hardware level protection techniques with the possibility of applying Triple Module Redundancy (TMR) at the full microprocessor level. TMR is a very well established protection technique exploited in several safety critical applications that however incurs in significant design overhead.

Results obtained by introducing this additional technique are interesting. When performing REL ONLY DSE this technique is always selected in the optimized system given its strong protection capability. It is selected as first option when starting with startlevel=0, while it is selected at the first change of level when starting with startlevel=1. Interestingly, given its high overhead, this technique is never selected in the optimized system when performing MULTI OBJ DSE. In some cases, when starting with startlevel=0, it may be selected during the search process but is is always discarded due to its cost.

To conclude, it is important to remark that, since reliability has been evaluated through the AVF of the system, it only considers the resiliency of the system to a soft error. It does not take into account the additional number of soft errors affecting the system in case of increased computation time and the different soft error rates of the different technology nodes. Different reliability metrics such as the FIT rate that are related to the AVF but also account for the soft error rate of the system can be however used to build different cost functions taking into account these effects.

5.3 Computation time

Fig.4 reports the average number of iterations required by both DSE strategies to reach a significant design improvement. The figure shows that, when optimizing for best reliability, a higher number of iterations is required before reaching the optimum compared to the multi-objective optimization strategy. This can be ascribed to the fact that, when optimizing for multi-objectives, the multiple constraints introduced in the design reduce the degree of freedom in finding an optimal solution, while optimizing for a single parameter relaxes this constraint.

The figure also shows the computation time required to build the model of the reference implementation and the time required to perform the DSE process. Results are provided in hours of computation on a high-end workstation (Intel(R) Core(TM) i7-4771 CPU @ 3.50GHz, RAM 16 GB) running Linux Ubuntu 14.04TLS. The construction of the initial model is of course the most time intensive task. Once this is accomplished the DSE process can work in a faster way exploiting the capability of the Bayesian model used to represent the system. The current implementation of the application is single threaded. Significant improvement is expected from a multi-threaded implementation in which several design options are evaluated in parallel.

5.4 From benchmarks to real applications

The analysis conducted so far considers realistic hardware architecture but is limited to simple software applications.

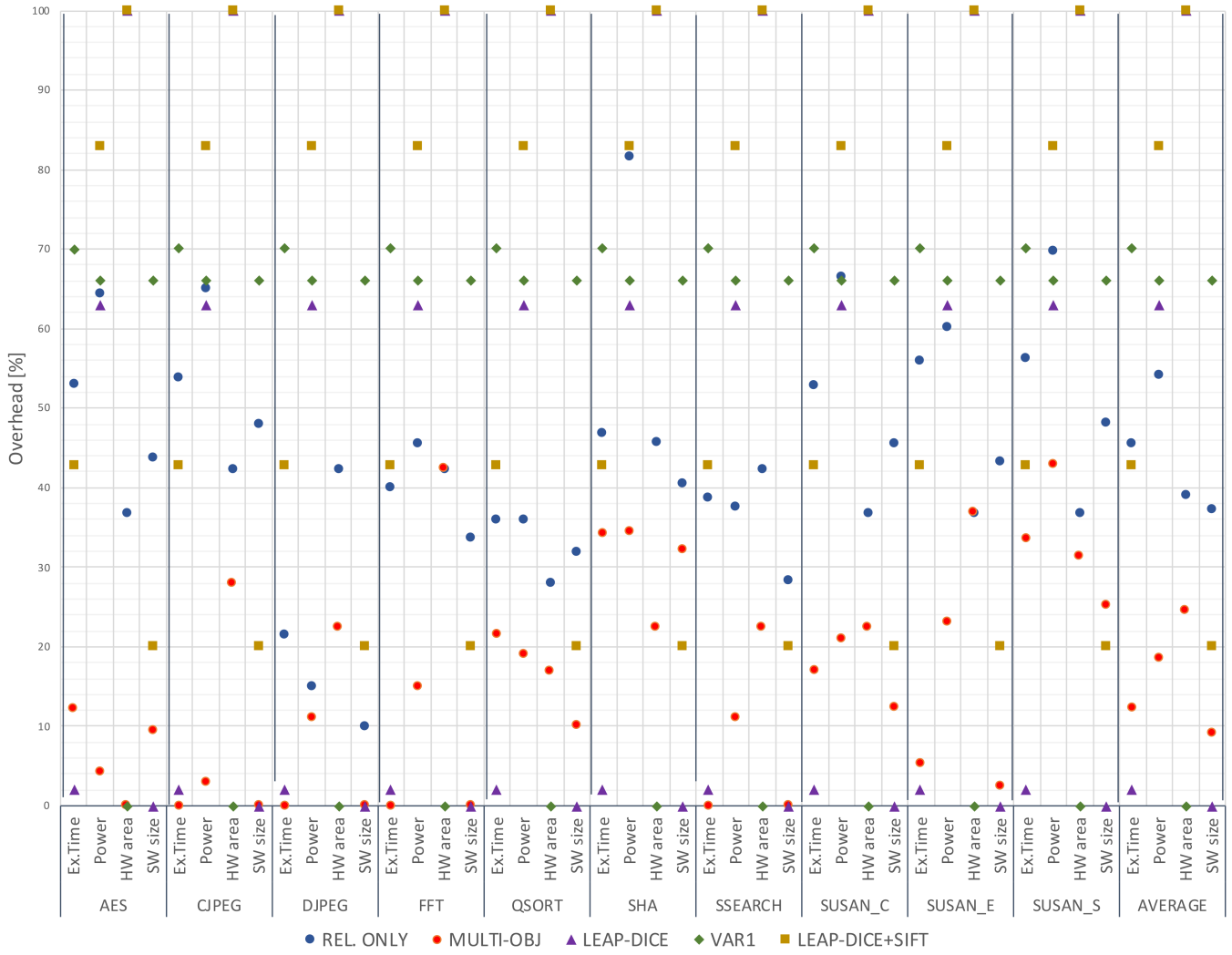

Fig. 3. Overhead comparison between the proposed DSE strategies and state-of-the-art reliability techniques protecting all portions of the system. The selected stare-of-the-art techniques are: (1) LEAP-DICE protecting the hardware layer only, (2) VAR1 protecting the software layer only and, (3) LEAP-DICE+SIFT proposing a cross layer protection of the system.

TABLE 5
Impact of the characteristics of the reference implementation on the DSE process for the FFT benchmark. Results show the average variation of the different parameters with respect to the ones obtained starting the DSE process with an unprotected system.

		Δ Step	ΔAVF(%)	$\Delta C(S)$	ΔEx. Time(%)	ΔPower(%)	ΔHW area(%)	ΔSW size(%)
REL ONLY	avg	-206.25	9.10E-06					
	stdev	115.43	2.61E-05					
MULTI-OBJ	avg	120.50	3.77E-06	5.24E-3	16.03	-5.12	-16.76	8.46
	stdev	158.80	2.76E-05	1.94E-2	8.53	-6.6.28	8.19	4.06

To show the DSE algorithm at work on a real case we performed an additional experiment on a system running a real HPC application. We selected the Sierpinski framework[2], an open-source HPC application for the solution of hyperbolic partial differential equations used in several fluid dynamics simulators. The software application is very complex and its Bayesian model accounts for more than 800 nodes, thus representing a good candidate to stress the capability of the algorithm. The application is not designed for low-end microprocessors such as the ARM Cortex A15, therefore the optimization was performed considering a single microprocessor architecture (SKYLAKE).

Fig. 5 and Fig. 6 show how the AVF of the system improves during the different iterations of the optimization process using the two considered optimization strategies. In this experiment, given the complexity of the system, the optimization was limited to 250 iterations. The red line shows the trajectory of the best implementation while blue dots represent the AVF of all evaluated systems. Looking at the figure, it is interesting to report that, in this case, the reference implementation started from a significantly high AVF and thanks to the optimization process we were

2. https://www5.in.tum.de/sierpinski/

TABLE 6
Impact of startlevel on the DSE process. Results show the variation for the different parameters executing Alg. 1 with startlevel equal to 0 or 1.

	REL ONLY		MULTI-OBJ						
	ΔStep	ΔAVF	ΔStep	ΔAVF	$\Delta C(S)$	ΔEx Time	ΔPower	ΔHW Area	ΔSW Size
AES	130	-2.96E-07	89	1.25E-04	0.005	-0.41	-0.50	0.56	2.10
CJPEG	-60	9.90E-06	-371	-4.30E-06	0.008	4.53	6.13	-16.80	10.00
DJPEG	-95	-5.00E-07	-48	-3.90E-06	-0.006	0.00	-3.70	5.56	-5.00
FFT	-56	-2.23E-05	75	-5.70E-06	0.002	-2.15	1.47	2.80	-1.00
QSORT	-16	4.50E-07	-206	-1.65E-06	-0.014	-10.87	1.70	7.16	-5.00
SHA	-61	7.25E-06	29	3.24E-04	-0.005	-12.55	-4.13	18.30	-6.36
SSEARCH	-60	0.00E+00	-151	-6.60E-08	0.015	11.33	3.83	-21.20	13.33
SUSAN_C	129	3.30E-06	-5	-8.15E-06	-0.040	-9.08	-3.46	-2.80	-4.57
SUSAN_E	30	2.00E-07	-3	-6.72E-05	-0.013	-10.90	6.81	1.50	-2.88
SUSAN_S	36	2.18E-07	-64	-6.60E-06	-0.046	-12.56	-12.20	11.20	-9.13

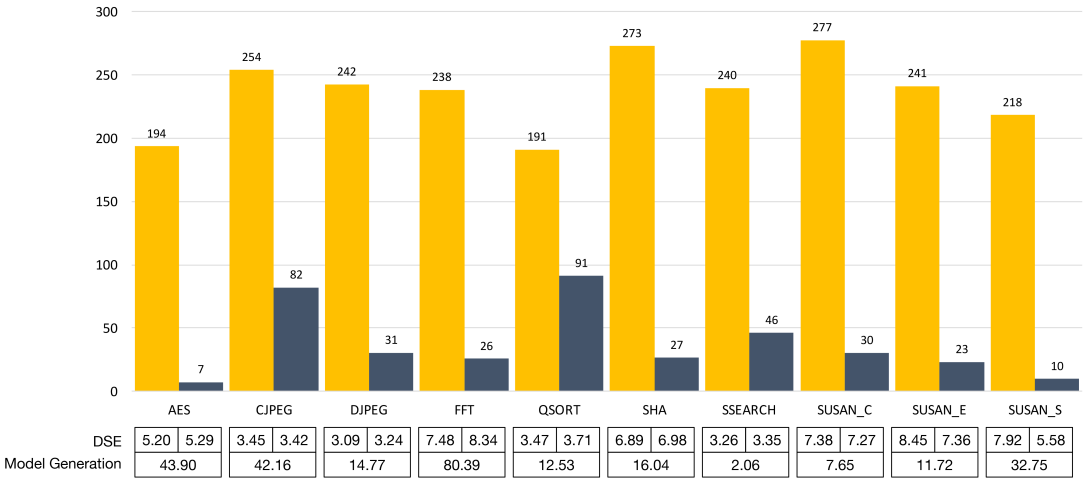

Fig. 4. Number of iterations before best solutions (500 iterations were simulated) and hours required to generate the model of the initial solution and to perform the DSE process. DSE for best reliability (yellow bars) vs. DSE for multiple objectives (dark blue bars).

able to significantly decrease the AVF and therefore increase the final reliability of the system. This is evident when looking at Fig.7 that shows the overhead with respect to the reference implementation for the 5 considered design parameters considering the two optimization strategies.

Interestingly, in this complex application, the multiple-objectives DSE was able to obtain high reliability by protecting a few key software functions. This further confirms and motivates the benefit of an automatic DSE framework such as the one presented in this paper.

Thanks to the proposed DSE strategies, we were able to obtain almost the same reliability level of the LEAP-DICE+SIFT implementation, but with significant overhead reductions in all other design dimensions.

For a complex application like the Sierpinski framework, the construction of the model required about 172 hours of simulation. Nevertheless, after building the initial model the DSE process required about 10 hours to complete. In this case, the availability of a single microprocessor architecture reduced the size of search space compensating for the complexity of the model.

As demonstrated in this section, the application of the proposed DSE engine is not limited to cases in which all layers of the design can be freely modified. It can be also exploited to optimize systems in which a full layer (e.g., the full hardware architecture) or portions of a layer (e.g., portions of already verified legacy code, a predefined microprocessor architecture) are fixed and cannot be modified. Working with a fixed portion of the system has the effect of restricting the design space. This on the one end simplifies the life of the DSE heuristic since less design alternatives must be considered. However, it limits the number of available options potentially impacting the characteristics of the generated design. There is no way to quantify how this will affect the optimized systems since this is strictly application dependent. Nevertheless, the DSE engine guarantees that the space of the available design options will be properly explored.

6 Conclusions

This paper presented a cross-layer multi-objective DSE algorithm for complex soft error resilient electronic systems. The DSE engine is based on an extension of the extremal optimization theory to deal with a hierarchical organization of the search space in conjunction with a model able to analyze the resiliency of the system to soft errors.

The proposed framework allows the designers to evaluate the effect that different HW/SW architectures and different protection techniques have on the system design parameters such as reliability, execution time, power, hardware area and software size. The DSE can be driven by a generic cost function in order to give the final user freedom to setup its specific optimization goals. Two DSE strategies,

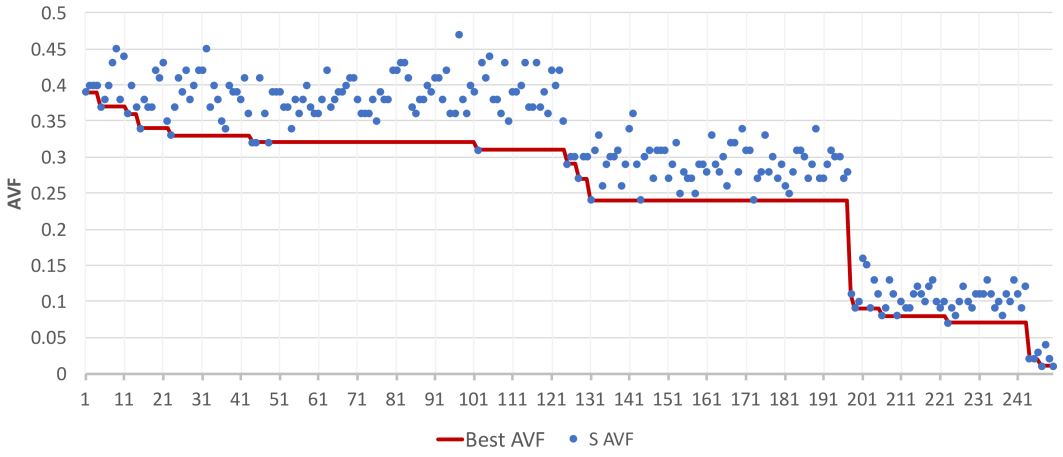

Fig. 5. AVF trajectory when performing DSE for the Sierpinski framework for best reliability.

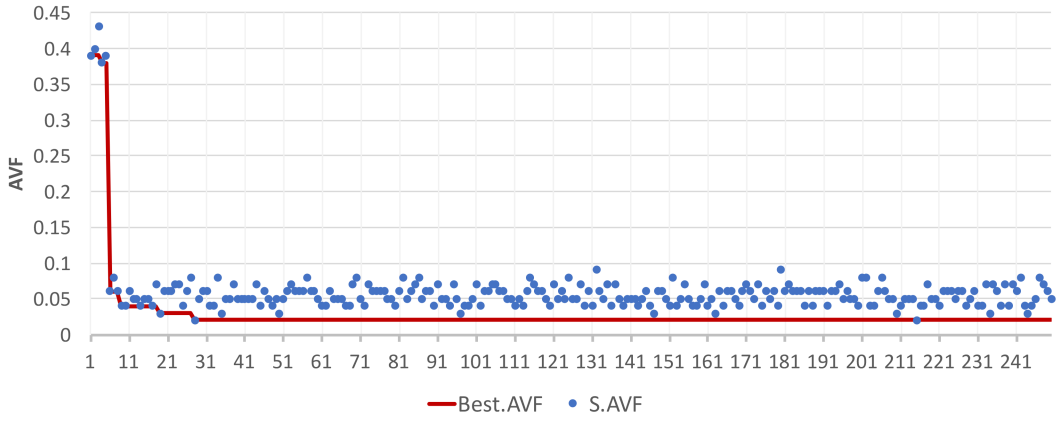

Fig. 6. AVF trajectory when performing DSE for the Sierpinski framework for multiple objectives.

one for best reliability and one for multiple objectives were presented in this study.

The algorithm was tested on a set of systems based on realistic hardware models. For the software we considered both benchmark applications and a very complex open-source HPC application. In all cases the algorithm demonstrated its capability of optimizing the system.

While the presented technique demonstrated a good potential, it is open to future improvements. Currently, the proposed DSE engine works with an orthogonal library of protection techniques. This actually means that we do not represent compatibility/incompatibility relations between the techniques. Expressing these relations when modeling the different design options will be an important step forward to express additional constraints during the DSE and to generate systems that better fit the requirements of the designer. Some of these constraints could be for instance modeled in the cost function, giving higher scores to solutions that concurrently use selected combinations of nodes.

Acknowledgment

This paper is part of the results of a three year Ph.D. research project on cross-layer reliability analysis [48]. The research has been supported by the 7th Framework Program of the European Union through the CLERECO Project, under Grant Agreement 611404.

References

[1] A. Messer, P. Bernadat, G. Fu, D. Chen, Z. Dimitrijevic, D. Lie, D. D. Mannaru, A. Riska, and D. Milojicic, "Susceptibility of commodity systems and software to memory soft errors," *IEEE Transactions on Computers*, vol. 53, no. 12, pp. 1557–1568, Dec 2004.

[2] M. Riera, R. Canal, J. Abella, and A. Gonzalez, "A detailed methodology to compute soft error rates in advanced technologies," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 217–222.

[3] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design Test of Computers*, vol. 22, no. 3, pp. 258–266, May 2005.

[4] A. Savino, S. D. Carlo, G. Politano, A. Benso, A. Bosio, and G. D. Natale, "Statistical reliability estimation of microprocessor-based systems," *IEEE Transactions on Computers*, vol. 61, no. 11, pp. 1521–1534, Nov 2012.

[5] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, May 2013, pp. 1–10.

[6] A. DeHon, N. Carter, and H. Quinn, "Final report for ccc cross-layer reliability visioning study," March 2011. [Online]. Available: http://www.relxlayer.org/FinalReport?action=AttachFile&do=get&target=final_report.pdf

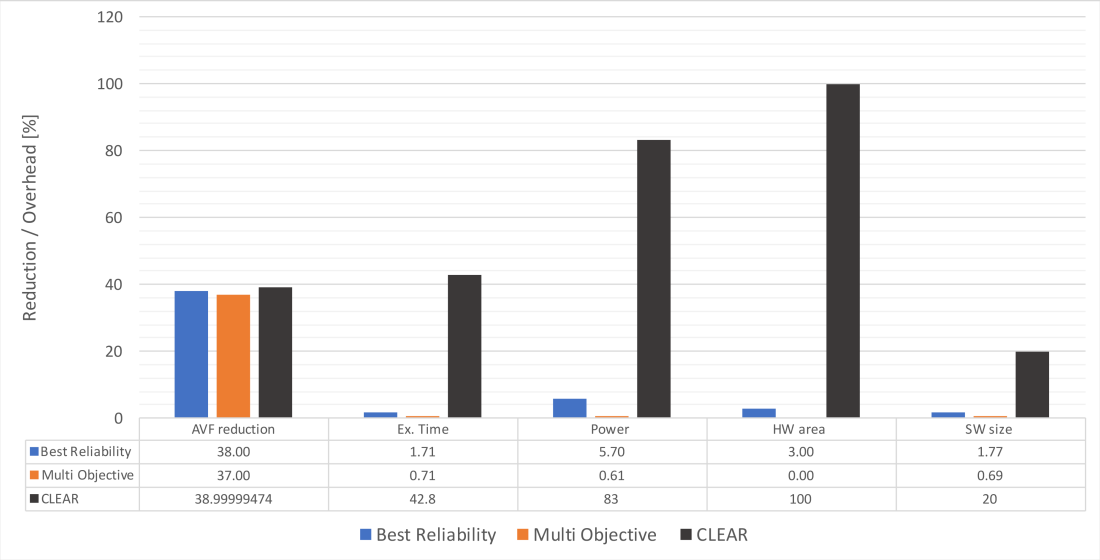

Fig. 7. Percentage improvement for all design dimensions when performing DSE for the Sierpinski framework using the two DSE strategies.

[7] J. Henkel, L. Bauer, H. Zhang, S. Rehman, and M. Shafique, "Multi-layer dependability: From microarchitecture to application level," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014, pp. 1–6.

[8] E. Cheng, S. Mirkhani, L. G. Szafaryn, C. Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, J. A. Abraham, P. Bose, and S. Mitra, "Clear: Cross-layer exploration for architecting resilience: Combining hardware and software techniques to tolerate soft errors in processor cores," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2016, pp. 1–6.

[9] A. Vallero, A. Savino, G. Politano, S. Di Carlo, A. Chatzidimitriou, S. Tselonis, M. Kaliorakis, D. Gizopoulos, M. Riera, R. Canal, A. Gonzalez, M. Kooli, A. Bosio, and G. Di Natale, "Cross-layer system reliability assessment framework for hardware faults," in *2016 IEEE International Test Conference (ITC)*, Nov 2016, pp. 1–10.

[10] D. Coit, T. Jin, and H. Tekiner, "Review and comparison of system reliability optimization algorithms considering reliability estimation uncertainty," in *Reliability, Maintainability and Safety, 2009. ICRMS 2009. 8th International Conference on*, July 2009, pp. 49–53.

[11] S. Boettcher and A. Percus, "Nature's way of optimizing," *Artificial Intelligence*, vol. 119, pp. 275 – 286, 2000.

[12] B. Xing, W.-J. Gao, and T. Marwla, "The applications of computational intelligence in system reliability optimization," in *Computational Intelligence for Engineering Solutions (CIES), 2013 IEEE Symposium on*, April 2013, pp. 7–14.

[13] M. Shafique, S. Rehman, P. V. Aceituno, and J. Henkel, "Exploiting program-level masking and error propagation for constrained reliability optimization," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, May 2013, pp. 1–9.

[14] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel, "Reliable software for unreliable hardware: embedded code generation aiming at reliability." in *CODES+ISSS*, R. P. Dick and J. Madsen, Eds. ACM, 2011, pp. 237–246.

[15] S. Rehman, M. Shafique, and J. Henkel, "Instruction scheduling for reliability-aware compilation." in *DAC*, P. Groeneveld, D. Sciuto, and S. Hassoun, Eds. ACM, 2012, pp. 1292–1300.

[16] S. Rehman, K.-H. Chen, F. Kriebel, A. Toma, M. Shafique, J.-J. Chen, and J. Henkel, "Cross-layer software dependability on unreliable hardware." *IEEE Trans. Computers*, vol. 65, no. 1, pp. 80–94, 2016.

[17] N. Wattanapongsakorn and S. Levitan, "Reliability optimization models for embedded systems with multiple applications," *Reliability, IEEE Transactions on*, vol. 53, no. 3, pp. 406–416, Sept 2004.

[18] S. Cotofana, A. Schmid, Y. Leblebici, A. Ionescu, O. Soffke, P. Zipf, M. Glesner, and A. Rubio, "Conan - a design exploration framework for reliable nano-electronics architectures," in *2005 IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP'05)*, July 2005, pp. 260–267.

[19] E. Cheng, S. Mirkhani, L. G. Szafaryn, C. Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, J. A. Abraham, P. Bose, and S. Mitra, "Tolerating soft errors in processor cores using clear (cross-layer exploration for architecting resilience)," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2017.

[20] A. Vallero, S. Tselonis, N. Foutris, M. Kaliorakis, M. Kooli, A. Savino, G. Politano, A. Bosio, G. Di Natale, D. Gizopoulos, and S. Di Carlo, "Cross-layer reliability evaluation, moving from the hardware architecture to the system level: A CLERECO EU project overview," *Microprocessors and Microsystems*, vol. 39, no. 8, pp. 1204 – 1214, 2015.

[21] S. S. Mukherjee, C. T. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "Measuring architectural vulnerability factors," *IEEE Micro*, vol. 23, no. 6, pp. 70–75, Nov 2003.

[22] G. E. Box and G. C. Tiao, *Bayesian inference in statistical analysis*. John Wiley & Sons, 2011, vol. 40.

[23] S. Boettcher, "Extremal optimization: heuristics via coevolutionary avalanches," *Computing in Science Engineering*, vol. 2, no. 6, pp. 75–82, Nov 2000.

[24] I. De Falco, A. Della Cioppa, D. Maisto, U. Scafuri, and E. Tarantino, "A multiobjective extremal optimization algorithm for efficient mapping in grids," in *Applications of Soft Computing*. Springer, 2009, pp. 367–377.

[25] A. Nakada, K. Tamura, and H. Kitakami, "Optimal protein structure alignment using modified extremal optimization," in *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, Oct 2012, pp. 697–702.

[26] S. Khan, "Application of extremal optimization algorithm to multi-objective topology design of enterprise networks," in *Computer, Control, Informatics and Its Applications (IC3INA), 2013 International Conference on*, Nov 2013, pp. 135–140.

[27] J. Chen, G.-Q. Zeng, K.-D. Lu, W.-W. Peng, Z.-J. Zhang, and Y.-X. Dai, "Extremal optimization algorithm with adaptive constants dealing techniques for constrained optimization problems," in *Industrial Electronics and Applications (ICIEA), 2014 IEEE 9th Conference on*, June 2014, pp. 1745–1750.

[28] J. Duch and A. Arenas, "Community detection in complex networks using extremal optimization," *Phys. Rev. E*, vol. 72, p. 027104, Aug 2005.

[29] D. G. Bounds, "New optimization methods from physics and biology," *Nature*, vol. 329, pp. 215–219, 1987.

[30] M.-R. Chen and Y.-Z. Lu, "A novel elitist multiobjective optimization algorithm: Multiobjective extremal optimization," *European Journal of Operational Research*, vol. 188, no. 3, pp. 637–651, 2008.

[31] M.-R. Chen, J. Weng, and X. Li, "Multiobjective extremal optimization for portfolio optimization problem," in *Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on*, vol. 1, Nov 2009, pp. 552–556.

[32] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, Dec 2001, pp. 3–14.

[33] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, B. Chin, M. Nicewicz, C. Russell, W. Y. Wang, L. B. Freeman, P. Hosier *et al.*, "Ibm experiments in soft fails in computer electronics (1978–1994)," *IBM journal of research and development*, vol. 40, no. 1, pp. 3–18, 1996.

[34] A. Chatzidimitriou and D. Gizopoulos, "Anatomy of microarchitecture-level reliability assessment: Throughput and accuracy," in *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 69–78.

[35] M. Ebrahimi, N. Sayed, M. Rashvand, and M. B. Tahoori, "Fault injection acceleration by architectural importance sampling," in *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 2015, pp. 212–219.

[36] S. Raasch, A. Biswas, J. Stephan, P. Racunas, and J. Emer, "A fast and accurate analytical technique to compute the avf of sequential bits in a processor," in *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 2015, pp. 738–749.

[37] M. Ebrahimi, L. Chen, H. Asadi, and M. B. Tahoori, "Class: Combined logic and architectural soft error sensitivity analysis," in *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*. IEEE, 2013, pp. 601–607.

[38] N. J. George, C. R. Elks, B. W. Johnson, and J. Lach, "Transient fault models and avf estimation revisited," in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*. IEEE, 2010, pp. 477–486.

[39] G. S. Rodrigues and F. L. Kastensmidt, "Soft error analysis at sequential and parallel applications in arm cortex-a9 dual-core," in *Test Symposium (LATS), 2016 17th Latin-American*. IEEE, 2016, pp. 179–179.

[40] M. Kaliorakis, S. Tselonis, A. Chatzidimitriou, N. Foutris, and D. Gizopoulos, "Differential fault injection on microarchitectural simulators," in *2015 IEEE International Symposium on Workload Characterization*, Oct 2015, pp. 172–182.

[41] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[42] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation Test in Europe Conference Exhibition*, April 2009, pp. 502–506.

[43] K. Lilja, M. Bounasser, S.-J. Wen, R. Wong, J. Holst, N. Gaspard, S. Jagannathan, D. Loveless, and B. Bhuva, "Single-event performance and layout optimization of flip-flops in a 28-nm bulk technology," *IEEE Transactions on Nuclear Science*, vol. 60, no. 4, pp. 2782–2788, 2013.

[44] S. Wang, J. Hu, and S. G. Ziavras, "Self-adaptive data caches for soft-error reliability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1503–1507, 2008.

[45] H. Amrouch and J. Henkel, "Self-immunity technique to improve register file integrity against soft errors," in *VLSI Design (VLSI Design), 2011 24th International Conference on*. IEEE, 2011, pp. 189–194.

[46] E. Chielle, F. L. Kastensmidt, and S. Cuenca-Asensi, "Tuning software-based fault-tolerance techniques for power optimization," in *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2014 24th International Workshop on*. IEEE, 2014, pp. 1–7.

[47] L. Xiong and Q. Tan, "A configurable approach to tolerate soft errors via partial software protection," in *Parallel and Distributed Processing with Applications Workshops (ISPAW), 2011 Ninth IEEE International Symposium on*. IEEE, 2011, pp. 260–265.

[48] A. Vallero, "Cross layer reliability estimation for digital systems," Ph.D. dissertation, Politecnico di Torino, 2017. [Online]. Available: http://porto.polito.it/2673865/

Alessandro Savino (M'14) received a Ph.D. in information technologies and a M.Sc. degree in computer engineering from Politecnico di Torino, Italy, where he is currently an Assistant Professor with the Department of Control and Computer Engineering. His main research topics are microprocessor test and software-based self-test as well as bioinformatics and image processing.

Alessandro Vallero (S'15) received a Ph.D. in computer engineering from Politecnico di Torino in Italy and a M.Sc. degree in electronic engineering from the University of Illinois at Chicago, US, and Politecnico di Torino, Italy. Currently he is a postdoc at the Department of Control and Computer Engineering of Politecnico di Torino in Italy. His research interests focus on system level reliability and reliable reconfigurable systems.

Stefano Di Carlo (SM'00-M'03-SM'11) received a M.Sc. degree in computer engineering and a Ph.D. degree in information technologies from Politecnico di Torino, Italy, where he is a tenured Associate professor. His research interests include DFT, BIST, and dependability. He has coordinated the EU-FP7 CLERECO on Cross-Layer Early Reliability Estimation for the Computing cOntinuum. Di Carlo has published more than 150 papers in peer reviewed IEEE and ACM journals and conferences. He regularly serves on the Organizing and Program Committees of major IEEE and ACM conferences. He is a golden core member of the IEEE Computer Society and a senior member of the IEEE.