This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

SyRA: Early System Reliability Analysis for Cross-layer Soft Errors Resilience in Memory Arrays of Microprocessor Systems

A. Vallero, A. Savino, A. Chatzidimitriou, M. Kaliorakis, M. Kooli, M. Riera, M. Anglada, G. Di Natale, A. Bosio, R. Canal, A. Gonzalez, D. Gizopoulos, R. Mariani and S. Di Carlo

Abstract— Cross-layer reliability is becoming the preferred solution when reliability is a concern in the design of a microprocessor-based system. Nevertheless, deciding how to distribute the error management across the different layers of the system is a very complex task that requires the support of dedicated frameworks for cross-layer reliability analysis. This paper proposes SyRA, a system-level cross-layer early reliability analysis framework for radiation induced soft errors in memory arrays of microprocessor-based systems. The framework exploits a multi-level hybrid Bayesian model to describe the target system and takes advantage of Bayesian inference to estimate different reliability metrics. SyRA implements several mechanisms and features to deal with the complexity of realistic models and implements a complete tool-chain that scales efficiently with the complexity of the system. The simulation time is significantly lower than micro-architecture level or RTL fault-injection experiments with an accuracy high enough to take effective design decisions. To demonstrate the capability of SyRA, we analyzed the reliability of a set of microprocessor-based systems characterized by different microprocessor architectures (i.e., Intel x86, ARM Cortex-A15, ARM Cortex-A9) running both the Linux operating system or bare metal in the presence of single bit upsets caused by radiation induced soft errors. Each system under analysis executes different software workloads both from benchmark suites and from real applications.

Index Terms— Reliability, Cross-layer, Microprocessors, Soft Errors, Failures-in-Time.

1 Introduction

In the last decade, microprocessor performance has continuously increased, leveraging the benefits introduced by the relentless technology scaling. This trend is expected to continue providing an integration capacity of billions of transistors. However, several design constraints such as reliability, power, energy and performance, are barriers to future scaling. In this context, cross-layer reliability (or cross-layer resilience) is gaining increasing relevance [1][2][3]. In a cross-layer resilient system, physical and circuit level techniques can mitigate low-level faults. Hardware redundancy can be used to manage errors at the hardware architecture layer. Eventually, software implemented error detection and correction mechanisms can manage those errors that escaped the lower layers of the stack [3][4].

The decision of how to distribute the error management across the different layers has the goal to meet the system reliability requirements of a specific application, considering its sensitivity to hardware faults. Overall, by considering multiple layers, one can exploit a wider range of information when handling errors. This leads to globally optimized error management strategies dedicated not only to reliability, but also to other design constraints [5].

- *S. Di Carlo, A. Savino and A. Vallero are with the Department of Control and Computer Engineering of Politecnico di Torino in Italy.*
- *M. Kooli and G. Di Natale are with the Montpellier Laboratory of Informatics, Robotics and Microelectronics (LIRMM) in France.*
- *A. Bosio is with University of Lyon - Lyon Institute of Nanotechnology (UMR CNRS 5270), France*
- *A. Chatzidimitriou, D. Gizopoulos and M. Kaliorakis are with the Department of Informatics and Telecommunications of the University of Athens in Greece.*
- *M. Anglada, R. Canal, A. Gonzalez and M. Riera are with the Computer Architecture Departament Universitat Politecnica de Catalunya in Barcelona, Spain.*
- *R. Mariani is with is with Intel, Italy.*

Contact email: stefano.dicarlo@polito.it.

Several error resilience techniques spanning multiple system layers have been presented. A survey of single and cross-layer dependability approaches can be found in [3]. However, at the industrial level, cross-layer resilience is still mainly guided by the experience of the designers [2]. A cross-layer holistic design approach has several advantages compared to traditional single layer techniques, but it increases the complexity of the design process since a larger design space must be explored. This translates into an increasing demand for system-level reliability analysis frameworks able to evaluate different combinations of cross-layer error protection techniques early in the design cycle [6][7]. Unfortunately, such tools still lack maturity, especially compared to those available to optimize other design parameters such as power and performance.

Creating frameworks for cross-layer reliability analysis is difficult. They have to integrate data generated by different design teams (see Section 5 for a review of relevant publications in the field). Register Transfer Level (RTL) or gate level fault injection campaigns are among the most accurate approaches to precisely analyze the resilience of a circuit to different types of hardware faults [8]. RTL models enable to precisely simulate all hardware structures of a microprocessor, including memory arrays and control/functional units. Nevertheless, even using statistical fault injection [9], the complexity of RTL and gate level simulations (especially when considering large memory arrays) makes the characterization of several combinations of error mitigation mechanisms in a cross-layer approach difficult to afford. This is a critical issue in the early design phases when fast evaluations are required to take informed design decisions. Moreover, when considering cross-layer resiliency from a system perspective, applications include operating systems, drivers and filesystems.

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

Most of these elements are hard to model in an RTL simulation environment. This is due either to limitations of the available simulators or due to the fact that evaluating their contribution to the reliability would require simulating several millions of clock cycles running again into performance issues.

This paper tries to overcome some of these limitations presenting SyRA, a system-level cross-layer reliability analysis framework for radiation induced soft errors in the memory arrays of a microprocessor. SyRA has been created to support designers in the early phases of the design, considering all layers of a system from the hardware up to the application software (including the operating system). SyRA exploits a multi-level hybrid Bayesian model to describe the target system and to estimate different reliability metrics. The construction of the system is based on simulations at the different abstraction levels. This allows us to speed up the analysis and therefore to cope with the complexity of the simulation of the full software stack. SyRA extends a preliminary attempt to use Bayesian networks for cross-layer reliability analysis [53]. Compared to [53], SyRA proposes a revised model that changes the way technological parameters are modeled in order to support the estimation of different reliability metrics including: Architecture Vulnerability Factor (AVF), Failures In Time (FIT) rate, and Executions Per Failure (EPF). In particular, the last metric enables the designer to trade-off reliability and performance in a single measure providing a valuable tool to optimize a computing system. Moreover, SyRA optimizes the model of the interface between components at different layers in order to cope with the complexity of the analysis in case of very complex applications, thus overcoming the scalability issues of [53] when considering large realistic software applications.

The complete tool-chain developed to build the model is described in detail to show the steps required to implement the proposed approach and to evaluate the computational cost to analyze a system. This is a key factor in the early design phases when several design options need to be evaluated. The proposed framework scales efficiently with the complexity of the system. On average it is 68% faster than full micro-architecture level fault injection and two orders of magnitude faster than RTL fault injection while maintaining a comparable accuracy.

Through the use of Bayesian inference, SyRA supports root cause and diagnostic analysis. This can drive the effort of the reliability engineers toward the weak portions of the system, thus reducing the need for over designing. Moreover, Bayesian inference supports speculation on the effects that different protection mechanisms have on the system. This feature could be a useful building block to support algorithms for automatic design space exploration, which are however out of the scope of this paper.

To demonstrate the capability of SyRA, we analyzed the reliability of a set of microprocessor-based systems characterized by different commercial microprocessor architectures (i.e., Intel x86, ARM Cortex-A15, ARM Cortex-A9) running both the Linux operating system or bare metal in the presence of single bit upsets caused by radiation induced soft errors. Each system under analysis executes different software workloads both from benchmark suites (MiBench [10]) and from real applications. Results demonstrate the accuracy of the reliability estimations provided by the framework when compared to estimations obtained by statistical RTL and micro-architectural level fault injection campaigns and show the capabilities offered by the framework.

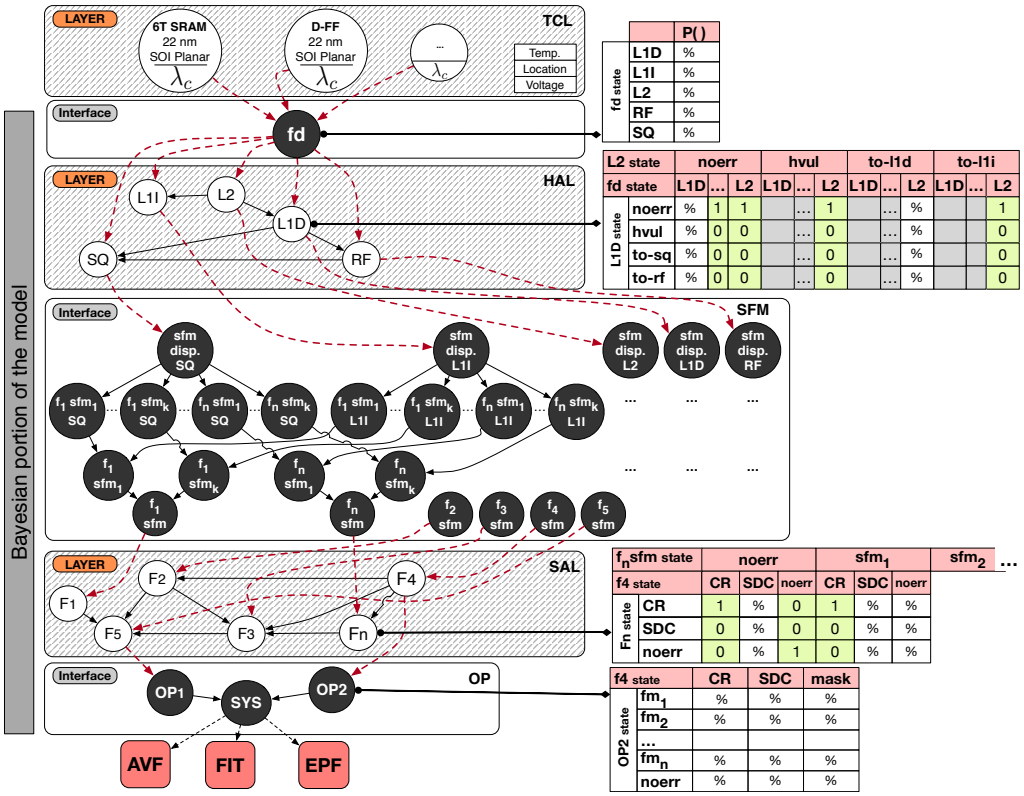

Fig. 1 System reliability model. The model is organized into different layers connected through a set of special macro nodes called interfaces. The core of the model is a Bayesian network that represents how soft errors propagate through the layers of the system. The figure shows a set of examples of the conditional probability tables of the nodes composing the different layers and interfaces. The output of the model is a set of reliability metrics for the system.

2 System reliability model

This paper focuses on building a framework for early system reliability assessment in presence of radiation induced soft errors in the memory arrays of a microprocessor. To achieve this goal, we first propose a system-level model whose main characteristics are:

- *component-based*: in component-based reliability modeling, system reliability is estimated through reliability parameters of individual system components and the way they interact in the system. Component-based reliability modeling can be easily integrated with typical component-based system design work-flows [11];
- *cross-layer:* the system is described with a clear separation of its composing layers [49]. The concept of *layer* here is used to identify a well-defined portion of the architecture of the modelled system (e.g., the hardware or the software architecture). This manages the complexity of a cross-layer analysis by splitting the system into sub-models. Each sub-model can be analyzed using dedicated techniques aiming at optimizing the simulation and analysis time, later recombining the results of the local analysis at the system level;
- *high parametrization and extension:* the model can fit a high number of heterogeneous parameters and is easy to extend.

To accommodate these characteristics, this paper proposes the *multi-level* and *hybrid* Bayesian model whose high-level structure is reported in Fig. 1. The concept of level does not have to be confused with the previously introduced concept of layer. The level is a characteristic of the modelling approach. It represents the system by means of a graph organized in two nested levels: the system-level, in which nodes represent layers and interfaces between layers, and the component-level that splits the system-level nodes into their components. At the system-level, the system denoted with S is modeled by means of a bipartite acyclic directed graph defined as:

$$S = (L \cup I, E) \tag{1}$$

where:

- $L = \{TCL, HAL, SAL\}$ is the set of layers composing the system (Fig. 1). Currently, three layers have been defined: the technology and circuit layer (TCL), the hardware architecture layer (HAL) and the soft-ware architecture layer (SAL). At the component-level, each layer is a graph itself. Its nodes represent the components of the layer and its arcs the interactions among them;
- I is the set of interfaces: special macro nodes that model how errors propagate across layers. As for the layers, the interfaces can also be split at the component-level into graphs of nodes;
- E is the set of edges connecting layers with interfaces and vice versa. They are depicted with dashed red lines in Fig. 1 to distinguish them from the arcs connecting nodes at the component-level that have different meanings depending on the context;

The bipartite property of the model in (1) satisfies the cross-layer property required by our framework and the isolation between different layers through a set of interfaces. The information that is transferred between the layers through the interfaces that will be discussed in the following sections can be summarized as follows:

- the interface between the TCL and the HAL, starting from the raw soft error rate of single cells, enables to compute the conditional probability of soft errors into the hardware structures;
- the interface between the HAL and the SAL transfers the information of those soft errors that managed to corrupt the execution of one instruction from the hardware to the software layer.

The proposed model is hybrid since different portions of the model integrate different types of information. In particular, part of the model is a Bayesian model representing how faults propagate through the system in a probabilistic way, whereas the other parts of the model are deterministic and are more related to the physical characteristics of the system. The main characteristics that lead us toward the application of Bayesian models for early reliability analysis is their capability of representing a complex system split into its components and their interconnections (i.e., to represent the architecture of the system). This differs from simulation approaches that consider the system as a whole. This is an important feature in the early design stages, when the system is built and a full picture of the system is still not available. Bayesian models are a compact representation of multivariate statistical distributions. They are a powerful formalism expressing how the state of different components (e.g., faulty or healthy) is affected by their interaction, and how events propagate through components. Bayesian models can be efficiently fit on simulation data on a component base. This enables us to devise dedicated fitting approaches for different components of the system at different layers, thus optimizing the creation of the model as will be discussed in the next subsections.

Finally, Bayesian models are well known to be a powerful tool for decision support, which is a key element in the early design stages when decisions must be taken. Interested readers may refer to [32] for additional information on Bayesian models.

2.1 Technology and circuit layer

The *technology and circuit layer (TCL)* is the lowest layer of the system stack. It models the physical and electrical phenomena leading to different classes of faults in an electronic circuit. These faults, when propagated up to the system layer, are potential root causes of system failures.

Circuit and electrical level fault modeling is a mature research field. This paper focuses on Radiation Induced Failures (RIF) leading to soft errors in memory structures. RIF can be caused by alpha particles from packaging materials and neutrons from the atmosphere. While alpha particles induced RIF can be mitigated selecting appropriate packaging materials, neutron induced RIF are difficult to prevent and their impact on the reliability of a system must be carefully assessed [12]. Therefore, we focus on this type of RIF.

At first, the impact of neutrons on a device depends on the circuit layout and on the fabrication technology. Since digital circuits are commonly designed based on standard cells, at the architecture-layer, the TCL can be constructed as a set of nodes each modeling a specific basic cell, with its reference technology and circuit structure (e.g., 6T/8T/10T SRAM cell, D-FF, latch, etc.). Each node is associated to the soft error rate of the related cell (SER_c) expressed in Failures in Time (FIT): the number of failures in one billion (10^9) device-hours of operation.

Apart from the technology and layout, several global and local parameters influence SER_c including [13]: supply voltage, operating temperature, operational location and altitude. SyRA implements the SER_c estimation workflow reported in Fig. 2.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

SER_c estimation starts from the calculation of the critical charge of the cell (Q_{crit}): the minimum charge produced by a neutron strike leading to a circuit malfunction. Q_{crit} is estimated by simulating a neutron strike modeled as a double exponential current pulse into the circuit [14].

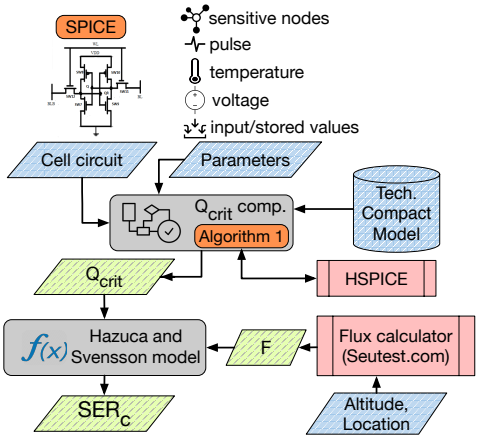

Fig. 2. SER_c estimation workflow. Colors indicate: inputs (blue), outputs (green), internal computational tasks (gray) and external tools (red).

The cell is modeled as a SPICE circuit while the target technology is fed as a technology compact model. Both (cells and compact model) are inputs of the estimation process. Cell architectures can be analyzed as far as a SPICE model is available. In the experimental campaign of this work the SRAM cell structures reported in [15] have been used; as well as, bulk planar and bulk FinFET ASU-PTM models. HSPICE, a commercial SPICE simulator from Synopsys, is used to carry out the simulations according to Alg. 1, but any SPICE simulator that can run ASU-PTM models can be employed for this purpose.

```
   Data: circuit, technology, pulse, temperature, voltage,
         input/stored values, sensitive nodes
   Result: Q_crit
1  foreach combination of input/stored values do
2      foreach sensitive node do
3          current = 0;
4          fault = 0;
5          while fault == 0 do
6              current = current + Δ;
7              setup_SPICE_simulation (circuit, technology,
                 temperature, voltage, input/stored value,
                 sensitive node);
8              run HSPICE simulation;
9              analyze HSPICE simulation;
10             if fault detected then
11                 fault = 1;
12                 Compute Q_crit as current pulse integral;
13             end if
14         end while
15     end foreach
16 end foreach
17 return Worst case or average Q_crit
```

Alg. 1 Q_{crit} simulation algorithm

Since Q_{crit} is influenced by the input values (i.e., the signal applied at the input of the cell during the simulation), the stored values (i.e., the value stored in a cell at the beginning of the simulation), and each cell may have more than one sensitive node, simulations are carried out for all combinations of these parameters (*foreach* loops at lines 1 and 2 of Alg. 1). In case of CMOS circuits (which are the ones considered in this publication), the area sensitive to neutron strikes is the drain area of the transistors [14]. Thus, "nodes" are transistors. Sensitive nodes are the transistors where the particle strike can cause a bitflip in the stored value. For each combination of stored value and sensitive node, current pulses with increasing amplitude are simulated (*while* loop at lines 5-14 of Alg. 1) until a fault (e.g., a flip of a memory cell) is detected. The SPICE simulation is performed considering the cell operating voltage and temperature. These parameters can be set cell by cell or globally for the full TCL, depending on the design. Since the shape of the pulse influences Q_{crit}, different rise times from the literature (2ps, 16ps, 33ps and 90ps) have been tested. The falling time is set to 200ps [14]. Finally, the width of the pulse is set as the interval between the start of the pulse and the instant when it decreases by 80% with respect to its maximum [14].

When a fault is detected in a simulation, the related Q_{crit} is computed by integrating the current pulse (line 12 of Alg. 1). When all simulations are carried out, the final average or worst case Q_{crit} can be returned. In the current implementation, the proposed procedure simulates the effect of a strike on a single transistor. Extending it to multiple transistors, therefore considering multiple bit flips, is part of our future work and will not affect the structure of the model but only the tool- chain used for the characterization of the technology.

Once Q_{crit} has been computed, SER_c can be calculated through the Hazucha and Svensson model [13]:

$$SER_c = C \times F \times A \times e^{-\frac{Q_{crit}}{Q_s}} \quad (2)$$

where:

- $C = 2.2 \times 10^{-5}$ is a technology independent constant computed by Hazucha and Svensson;
- F is the flux of neutrons at the specific location;
- A is the area of the circuit sensible to the neutron flux;
- Q_s is the charge collection efficiency. If a charge collected by a particle Q_{coll} is greater than Q_{crit} an RFI arises. Q_s is the mean Q_{coll} considering a range of energy particles. It is a technology dependent parameter that scales approximatively linearly with the length of the gate. It can be computed by linear regression from experimental data for CMOS technologies [13].

The neutron flux F commonly reported in literature for SER computation is from New York City at sea level. However, F is a function of the location in which the system operates and is mainly affected by two parameters [16]:

1. *altitude*: the flux increases exponentially with the altitude,
2. *vertical cutoff*: a parameter of the Earth's magnetic field depending on the geolocalization.

To account for these parameters, the proposed framework computes F exploiting the online calculator from Seutest.com [17], which is based on the empirical model proposed by Gordon et al. [18]. Resorting to the proposed TCL, designers can model several technology and circuit level design alternatives, obtaining raw SER estimations, thus enabling a fair comparison of different implementations or operational conditions of a system.

2.2 Fault Dispatcher

The *fault dispatcher* (**fd**) is the first interface node connecting the TCL with the *hardware architecture layer* (HAL). It is also the root node of the Bayesian portion of the proposed model. In a Bayesian model, each node can assume a set of states. Root nodes are described by computing the marginal probability of the node to be in each state.

The **fd** models how soft errors affecting the system are distributed among the hardware components. Since we target neu-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

tron induced RFI, it is very unlikely that multiple high-level architectural components (e.g., L1 data cache, register file, etc.) can be concurrently affected by a soft error caused by a single neutron strike. Working with this assumption that is different from the case of multiple bit upsets from a single strike in a component, the **fd** can assume as many states as the list of nodes of the HAL (V (HAL)). These nodes correspond to the components that define the hardware architecture of the system (see Section 2.3). Each state in **fd** models the event that a soft error affecting the system targets the corresponding component. The **fd** is therefore described as:

$$\boldsymbol{fd}: comp \in V(HAL) \rightarrow P(comp) \quad (3)$$

where $P(comp)$ is the probability that a soft error affecting the system is located in a specific component (*comp*). It is computed resorting to the SER of the basic hardware cells computed in the TCL as:

$$P(comp) = \frac{SER_{comp}}{\sum_{\forall cc \in V(HAL)} SER_{cc}} \quad (4)$$

SER_{comp} denotes the soft error rate of a full hardware component, which is the sum of the error rates of the cells used to build the component. It is important to stress here that the **fd** formulation provided here, is a specific case considered in this paper to model single neutron induced soft errors. Different fault distributions or fault types can be described in a similar way, given that they can be properly simulated when analyzing the hardware architecture layer.

2.3 Hardware Architecture Layer

The *hardware architecture layer* (HAL) models the hardware architecture of the system. As reported in Fig. 1, the hardware architecture is represented as a Bayesian network whose nodes model the hardware components of the system. A complex hardware component such as a microprocessor can be either modeled as a single node or split into different nodes modeling its relevant micro-architectural subcomponents. This granularity depends on several factors:

- the granularity of the results required by the designer during the reliability analysis;
- the possibility of modifying selected portions of the architecture;
- the availability of tools able to characterize the component in order to generate and populate the Bayesian model.

The memory arrays of a microprocessors are the main focus of this paper. This is motivated by the fact that microprocessors are among the most complex and important blocks of a computing system and memory arrays are highly sensible to soft erros. Therefore, the tool-chain implemented in this paper to populate the model is optimized for the memory arrays of a microprocessor. At the system-level, each node of the HAL is connected to the fault dispatcher (dashed red arcs in Fig. 1). This is used to model the event of a soft error affecting the component. At the component-level, components are connected in a hierarchy to model faults that propagate from one component to another one (solid black arcs in Fig. 1). Each node $c \in HAL$ is associated to a *conditional probability table* (cpt) defining the probability of the node to be in a given state conditioned on the states of its parent nodes.

$$\boldsymbol{cpt}(\boldsymbol{c}) = \mathrm{P(c|parents(c))} \quad (5)$$

Fig. 1 reports an example of cpt for the level-1 data cache block (L1D). The rows identify the possible states of the node:

- *noerr*: a fault in the component has been masked and is not propagated to the software layer;
- *hvul*: a fault in the component is *hardware vulnerable*, i.e., it has an impact on the execution of at least a software instruction. These faults are the ones propagated to the software layer;
- *to-<comp>*: the fault does not have a direct impact on the computation, but a corrupted entry has been propagated to another component. In this case, the cpt contains a state for each child node of the component.

Even if the cpt can grow in size (see Section 2.7), not all cases are realistic. Therefore, the cpt does not need to be populated in all its parts. Gray elements represent impossible cases deriving from the fact that our model assumes that a fault affects a single architectural block at a time (see Section 2.2). Green elements are deterministic cases that do not require any computation to be characterized. Empty cells are instead the cases in which a tool-chain is required to properly compute the related conditional probabilities.

To compute these values, the implemented tool-chain resorts to microarchitecture-level fault injection. For each node, a fault injection campaign is set up to understand the behavior of the related hardware component. This approach allows us to accurately simulate, with an affordable simulation time, the behavior of array-based hardware structures such as memories that are accurately represented by different microarchitecture-level models. These structures are among the most sensitive to soft errors [19][20]. Other hardware structures (e.g., combinational circuits) are functionally approximated to speed up the simulation and cannot be considered in our framework. This approximation, from the one hand limits the amount of hardware structures that can be analyzed by our framework but from the other hand it is a key choice to spee-up the analysis in order to properly model and consider the full software stack.

The proposed tool-chain is built on top of GeFIN, a microarchitecture-level fault injection tool based on Gem5 [21]. Gem5 is a cycle-accurate full-system simulator that models two of the major Instruction Set Architectures (ISA) available on the market: (i) ARM and (ii) x86 [22].

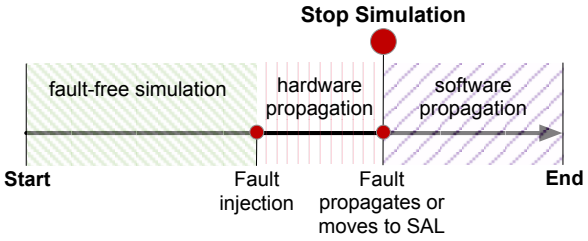

Fig. 3. Hardware soft error injection timeline.

Fig. 3 sketches the implemented simulation timeline. At this level, we simulate the program with a relevant workload, i.e., a set of input data stimulating the execution paths of the application and representative of its execution time. The fault-free simulation period represents the interval from the beginning of the application to the injection of the fault. After the fault is injected, the simulation continues until the fault propagates to another component or it becomes visible in the software execution. This last case corresponds to the clock cycle in which the first instruction affected by the fault commits to the architectural state. After that moment, the fault can be considered as propagated to the higher layer. To speed-up the analysis, several techniques have

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

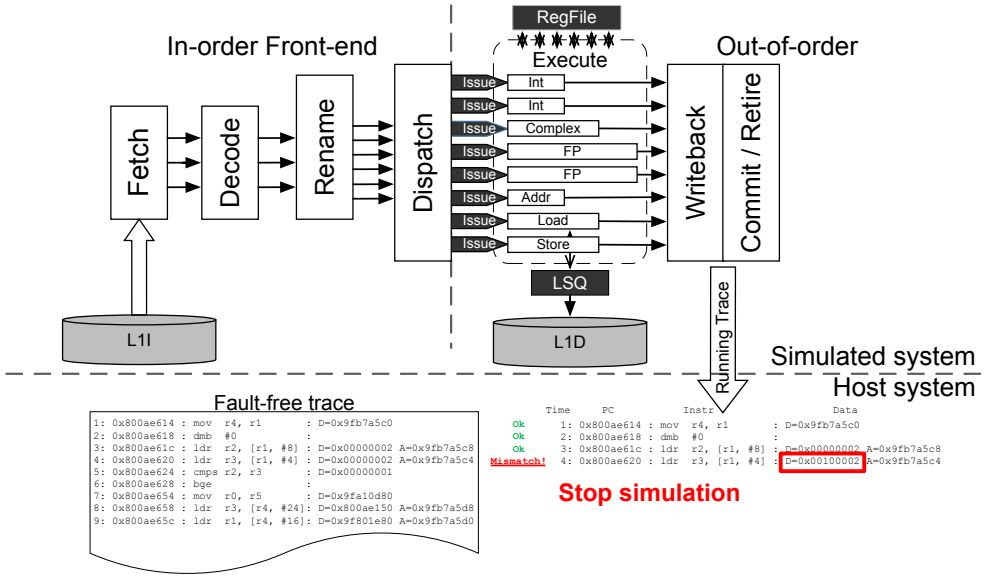

Fig. 4. Microprocessor microarchitecture simulation model.

been implemented to avoid wasting simulation time in the absence of faults, both in pre and post injection periods. More precisely, checkpointing was used to skip pre-injection period, which in average consumes up to 50% of the total simulation time, and early-stopping of the simulation was invoked in cases of fault discarding, either due to overwriting or injection on invalid entries.

To implement this simulation workflow, a trace mechanism is used to monitor the back-end of the processor pipeline (Fig. 4). The trace includes all information required to analyze the committed instructions: the decoded instruction and its operands, the data transactions in both registers and memory, the program instruction order, the processor execution mode (i.e., kernel mode vs. user mode) and the execution time of each instruction (to monitor performance deviations).

The tool-chain initially records a fault-free trace and then compares it against a faulty simulation in real-time while the simulation progresses. If the faulty entry is propagated to another component or upon a mismatch detection, the simulation immediately stops. In the latter case, the fault is marked as hardware vulnerable. All hardware vulnerable faults are analyzed and classified as will be described in Section 2.4. All memory array structures of the CPU, which occupy the vast portion of the chip's area and significantly influence the reliability of the entire chip, can be studied through the implemented tool-chain [23]. For the purposes of this study, we will present results targeting five important hardware components: Integer physical register file (RF), Store Queue (SQ), level-1 instruction cache (L1I), level-1 data cache (L1D) and level-2 cache (L2). By means of this characterization, the usage profile of the hardware structures is analyzed considering that different applications may use different portions of the structure in a different way, thus modifying the likelihood of an error to strike an active resource.

Since the characterization of each node of HAL requires the setup of a fault injection campaign, to generate the fault list for each node we use statistical fault sampling as described in [9]. Details about the parameters of the fault sampling will be provided in Section 4.

2.4 Software Fault Models

The *software fault models* (sfm) are the second interface node of the proposed model. They model how *hardware vulnerable* faults identified in the HAL can be modeled in the *software architecture layer* (SAL). This isolates the SAL from the knowledge of the underling hardware architecture.

Previous attempts to analyze software resiliency to hardware faults mainly tried to fully simulate the fault propagation from the hardware architecture up to software routines. This analysis assesses if they impact the correctness of the computation [24][25][26]. Differently, this paper focuses on modeling how the software computation perceives a hardware vulnerable fault at its abstraction layer. This allows us to decouple the HAL from the SAL.

We previously attempted to define software fault models at the level of the ISA of the microprocessor in [7]. Faults are described at this level as alterations that have an impact on the ISA of the microprocessor. The fact that these faults are defined at the ISA level, allows us to directly map them on the hardware vulnerable faults identified in the HAL. Indeed, they are faults in which a faulty instruction commits to the architectural state. Among the different sfm introduced in [7], SyRA's tool-chain currently implements the models described in Table 1.

TABLE 1
Software Fault Models (SFM)

Acronym	Software Fault Model	Description
WDO	Wrong Data in an Operand	An operand of the instruction is corrupted
DWI	Wrong Data in immediate	An immediate operand of the instruction is corrupted
OFS	Operand Forced Switched	An operand is used in place of another
IR	Instruction Replacement	An instruction is used in place of another

As reported in Fig. 1, the sfm interface node is a macro node upward connected to each hardware component of the HAL and downward connected to each node of the SAL. As described in Section 2.5, the nodes of the SAL model the set of functions of the target software. To handle the complexity of the models (see Section 2.7), the sfm node is organized at the component level

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

into a tree of nodes. The root nodes of this tree are a set of sfm dispatchers, one node for each component defined in the HAL. An example of the cpt of the sfm dispatcher associated to L2 is described in Fig. 5-A.

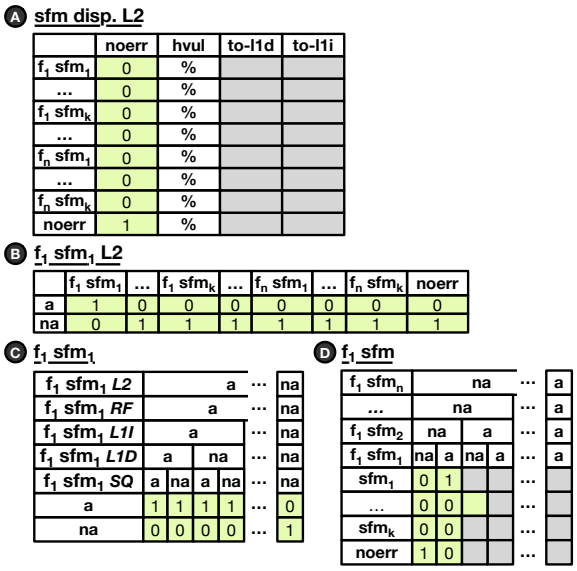

Fig. 5. Software fault models cpt organization.

The number of states of the node (i.e., the number of rows) is equal to the number of nodes of the SAL ($V(SAL)$) multiplied by the number of considered sfm. A state of the cpt models the event of the considered sfm happening in the considered function. The columns of the cpt are the possible states of the related hardware component. As in the previous examples, several columns of the cpt represent impossible cases (gray cells) or deterministic cases (green cells). The white column can be instead filled by analyzing each hardware vulnerable fault detected during the HAL characterization. These faults can be mapped to their target function by looking at the address of the corrupted instruction. Then, by looking at the type of corruption that has been detected (e.g., data corruption, instruction corruption, etc.), they can be classified into a target sfm.

2.5 Software architecture layer

The *software architecture layer* (SAL) models the architecture of the software executed in the system. As for the HAL, this layer is modeled at the component-level as a Bayesian network. This network represents the function call graph of the software application. In the function call graph, nodes of the graph represent software functions. At the system-level, each node is connected to the related sfm node in order to model the event of a sfm affecting the function (dashed red arcs in Fig. 1). At the component-level, functions are connected to model the function call hierarchy (solid black arcs in Fig. 1).

Fig. 1 shows an example of cpt associated to function f_n. The rows of the cpt model the impact of a sfm on the execution of the function. Based on the classification provided in [24] our tool-chain considers the following *software faulty behaviors* (sfb):

- *noerr* (*masked)*: the function produces correct results;
- *silent data corruption (SDC)*: the output of the function is different from the fault free output;
- *crash (CR):* the function generates an unrecoverable exception or enters an infinite loop becoming unresponsive.

As for the HAL, to populate the cpt of the SAL nodes, SyRA implements a dedicated tool-chain whose basic workflow is depicted in Fig. 6.

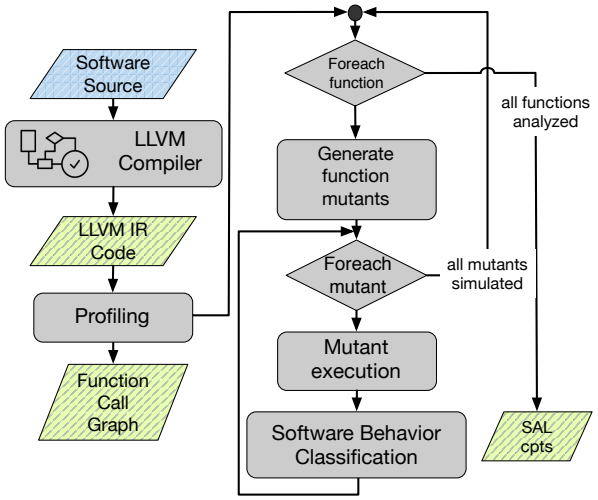

Fig. 6. SAL characterization workflow. Colors indicate: inputs (blue), outputs (green), computational tasks (gray).

One of the main goals of SyRA is to decouple the analysis of the different layers of the system as much as possible. To achieve this goal, it is necessary to analyze the software layer decoupling it from the specific execution platform and its ISA. We propose to use a virtual instruction set based on LLVM (Low Level Virtual Machine). LLVM defines an Intermediate Representation (IR) that describes the code in a form that is independent from the target machine.

According to the workflow of Fig. 6, the LLVM IR code is first executed and profiled in order to generate the function call graph required to build the SAL. A statistical software-level fault-injection campaign is then executed to characterize each function. The peculiar characteristic of the LLVM fault-injector developed in this work is that it operates by injecting the high-level sfm introduced in Section 2.4. This differentiates it from other LLVM based fault injectors such as LLFI [28] and KULFI [29] that are limited to low-level faults such as bit-flips in program data.

Software fault models are injected in the code by resorting to the concept of software mutants [30]. For each injected fault, the LLVM IR code is modified to generate a mutated version able to emulate the injection of the sfm during the execution. Every mutant is finally executed, and the result of the function is compared with a golden execution to identify one of the sfb previously defined.

The developed tool-chain enables a very high injection throughput since the software is executed at full speed on the hosting workstation. It is worth to highlight here that, while software layer fault injection alone should not be used to precisely evaluate the resiliency of a full system to soft errors [31], it is effective in our case. In fact, we only aim at analyzing how an error that already affected the execution of an instruction is propagated or masked by the software.

2.6 Observation points

The *observation points* (op) are the final interface of the proposed model. They allow us to map the information coming from the lower layers of the model to a set of system-level failure modes (fm). These failure modes can be exploited to compute a set of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

reliability metrics that are the actual output of the reliability analysis implemented in SyRA.

Differently from the previous parts of the model, this interface macro node is strictly related to the mission of the application. Therefore, its design cannot be fully automated. Nevertheless, this task is limited to the identification of the outcomes of the system and on the analysis of their importance to accomplish the mission. In the worst case, all outcomes can be considered equally important and the status of the system can be defined as correct or safe only if all the outcomes are correct.

The first role of the op is to define the software functions where the outcome of the computation can be monitored. Fig. 1 reports an example of observation points. In this case, two functions (f_4 and f_5) define the outcome of the computation. They are therefore connected to two op nodes in the model (dashed red lines). The cpt of these special nodes (see Fig. 1) maps the state of the parent node to a set of user defined system-level failure modes (e.g., unresponsive system, hazard in computation, etc.). Whenever specific system-level failure modes are not defined, the same software faulty behaviors defined in Section 2.5 can be used to define the failure modes making these cpt deterministic.

All observation points can be then aggregated into a single node to describe the final behavior of the system as a whole. Whenever a single function defines the output of the computation (e.g., the main of a program) a single level of nodes is enough to define the observation points.

2.7 Model complexity

The size of the cpt of the different nodes is the key factor that influences the complexity of the model and therefore the scalability of the tool-chain. The cpt of the fault dispatcher has a single columne and a number of rows equal to the number of hardware blocks of the system (see Fig. 1). It is therefore not critical.

Also, the nodes in the HAL are note critical (see Fig. 1). The total number of rows of the cpt of these nodes is equal to the number of children of the component plus 2. The number of columns is instead equal to the product of the number of states of its parent nodes and this number is usually low.

The fault dispatcher and the nodes of the SAL are the critical part of the model. The fault dispatcher could be, in principle, directly connected to the nodes of the SAL. Nevertheless, given the amount of states of this node, this would lead to an explosion of the size of the cpt of nodes at SAL layer whose number of columns would be equal to:

$$ncol = |sfb|^{pf} \times (|sfm| \times |V(SAL)|)^{|V(HAL)|} \quad (6)$$

This is clearly not manageable. The information of the sfm dispatchers is therefore reorganized through the following levels of the tree modeling the sfm interface node (see Section 2.4). At the second level, each state of the dispatcher is represented as a single node (see Fig. 5-B where *a* indicates that the corresponding state is active, whereas *na* indicates that the state is not active). At the third level, sfm information from different hardware components is aggregated together obtaining nodes that are associated to each function and each sfm (Fig. 5-C). Finally, for each function, information for different sfm are aggregated into a single node that can then be connected to the related function node in the SAL. An example of cpt for these nodes is reported in Fig. 5-D.

Even if this tree organization increases the number of nodes of the model, all cpt, apart for those of the sfm dispatchers, are deterministic cpt. This means that they do not require effort to be computed and can be efficiently dealt when solving the Bayesian model to perform reasoning. By applying this network organization, each node in the SAL is connected to a sfm node whose number of states is limited to the number of sfm and the number of nodes of its cpt can be computed as:

$$ncol = |sfb|^{pf} \times |sfm| \quad (7)$$

where:

- $|\cdot|$denotes the count operator;
- pf denotes the number of parent functions of the node.

The first term of (7) is the contribution of the nodes representing the parent functions. The second term is the contribution of the sfm node of the function.

Nevertheless, even with this optimization, the size of the cpt increases exponentially with the number of parent functions. In realistic software applications, this number can be high for some of the nodes (e.g., the main function of the program). This creates problems both when populating the cpt and when using the model to perform reasoning and estimations. To cope with this problem, we resort to the Noisy-MAX approach [27]. The Noisy-MAX is a generalization of the interaction of a child node with its parents that allows us to reduce the size of the computed cpt considering each parent in isolation. By means of this approach, the cpt grows linearly with the number of parents. This introduces a certain approximation that however will not significantly impact the accuracy of the model as discussed in [27] and demonstrated by the results provided in Section 4.

3 Reliability Analisys

Once built, the Bayesian reliability model presented in Section 2 is a powerful tool to analyze different reliability related aspects of a system. This section overviews the different features that have been implemented in SyRA.

3.1 Computation of reliability metrics

SyRA exploits predictive Bayesian reasoning [32] to compute a set of well-established system level reliability metrics. In predictive reasoning, starting from the fault dispatcher, i.e., the root node providing information about the fault causes, the designer is able to update its belief about the state of each component of the system. This provides information on how faults propagate across layers and between components of a layer.

The first reliability metric that can be computed using the proposed model is the Architectural Vulnerability Factor (AVF) of the system [33]. The AVF quantifies the probability of a soft error in a hardware component to manifest as a failure of the system. It jointly considers masking properties of the hardware architecture as well as of the executed software.
By looking at the state of the *sys* node in Fig. 1, the AVF of the system can be computed resorting to predictive Bayesian reasoning as:

$$AVF_S = 1 - P(sys = noerr) = 1 - \sum_{\boldsymbol{u} \in \boldsymbol{U}} P(sys = noerr|\boldsymbol{u}) \quad (8)$$

where $\boldsymbol{U}$ denotes the set of all possible instantiations $\boldsymbol{u}$, i.e., combination of states of the parent nodes of the sys node. Equation (8) is a recursive equation. It considers the noerr probability of all components from the bottom up to the top.

It is well known that solving (8) is a NP-hard problem [55]. SyRA exploits the Bayesian solvers implemented into Smile, an open library for Bayesian network analysis [34]. Two solvers can be used: (1) the exact solver proposed by Lauritzen in [35] that can be used with medium size models (i.e., tens of nodes), and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

(2) the Estimated Posterior Importance Sampling (EPIS) approximate stochastic solver proposed in [36] that can be used with very large models (i.e., thousands of nodes). Both solvers consider acyclic Bayesian networks, therefore loops cannot be represented in the model (e.g., faults that are propagated both from L2 to L1D and vice versa or recursive functions in the software applications). This represents an approximation of the real system. Bayesian networks can be extended to accommodate cycles. However, solvers for these networks such as the loopy belief propagation algorithm are more complex and may have convergence issues [37].

SyRA also offers the possibility to compute the contribution of each hardware component to the AVF of the system. This is possible by setting into the model the *evidence* that a soft error is affecting that component. This in turns means to condition the state of the sys node to a state of the fault dispatcher:

$$AVF_{comp} = 1 - P(sys = noerr | fd = comp) \quad (9)$$

The AVF of each component can then be used in conjunction with the soft error rate of the component defined in (4) to compute the FIT rate of the system:

$$\lambda_S = \sum_{\forall comp \in V(HAL)} SER_{comp} \cdot AVF_{comp} \quad (10)$$

SyRA can be easily extended to compute other reliability metrics that are related to AVF_S and λ_S as the Mean Time To Failure (MTTF). Additional work would be instead required to compute the Mean Time Between Failure (MTBF) that also accounts for the repair time in reparable systems.

Both the AVF and the λ_S are pure reliability metrics. However, as discussed in Section 1, designers must be able to trade-off reliability with other design parameters such as the performance of the system. To enable a joint analysis of reliability and performance, SyRA enables to compute the *executions per failure* (EPF) of the system [34]. EPF is the number of times an application must be executed before observing a system failure. It is computed as:

$$EPF = \frac{EIT}{\lambda_S} \quad (11)$$

where EIT (Executions in Time) is the number of executions of an application in 10^9 hours of device operation. This can be estimated by looking at the duration of a golden execution trace of the application. This implies to simulate the execution of the application by applying a representative workload for the mission of the application. When a single workload cannot be identified, multiple execution times can be averaged, or worst-case execution times can be considered. This is however a parameter that is demanded to the reliability engineer that has a deep knowledge of the target application.

As for pure reliability metrics, the EPF is not the only possible measure. Another option that could be considered is the failures per executions. It is a function of λ_S and of the application execution time, which are both available in the framework. In SyRA we decided to work with the EPF since, instead of providing a failure rate, it looks at the length of time a system is expected to last in operation before observing a failure. In our opinion this provides a different and interesting point of view compared to the failure rates.

3.2 Design exploration and diagnostic analysis

The capability of updating the belief of the state of each node based on the evidence of the state of selected nodes, opens several opportunities to analyze the system.

This mechanism can be used to quickly perform a simplified *early design exploration* to understand the impact on the AVF of the system of different combinations of cross-layer error protection techniques and their insertion points. Let us consider the example provided in Fig. 1. What is the benefit (i.e., AVF_S reduction) of a single-Error Correcting Code (ECC) in L2? To answer this question, it is enough to set the evidence that L2 = *noerr* then updating the beliefs of all other nodes:

$$AVF_S = 1 - P(system = noerr | L2 = err) \quad (12)$$

Whenever the effect of a protection mechanism can be quantified in terms of its masking probability on the related component, its effect in the system can be analyzed quickly without any new simulation. Of course, this is not a full design space exploration framework in which complete implementations are instantiated and simulated and an algorithm to efficiently explore the space is provided. As for the other parts of SyRA it is intended to give quick feedbacks to the designers in the early design phases.

In a similar way, SyRA allows to implement what we call *diagnostic analysis*. The diagnostic analysis evaluates the model in the backward direction starting from the leaf (i.e., the sys node) back to the root. This in turns means reasoning from symptoms to cause. If we set the evidence that the system has failed (i.e., set the evidence that *sys = fmi*), we can update our belief about the contribution of each node (hardware or software component) to this failure. This enables us to isolate those nodes that likely contribute to the failure. This has the potential to drive the reliability design effort toward the most critical components, thus optimizing the overall system at the lower cost. A similar approach can be used to study the impact of a failure in a given component on the status of the full system.

By resorting to design exploration and diagnostic analysis, system designers are provided with a powerful tool that enables early reliability analysis of the complete system.

3.3 Dealing with error margins of the layers

Conditional probabilities populating the reliability model presented in Fig. 1 are computed resorting to statistical fault-injection campaigns at different layers (see Section 2.3 and Section 2.4). Whenever statistical fault-injection is applied, estimated information is accompanied with an error margin that depends on the amount of injections that have been performed [9]. This means that each parameter of the different cpts is not an exact value but is uniformly distributed within the related error interval. These variations must be considered since their combined effect can influence the system level reliability metrics computed through the proposed model.

For this reason, SyRA exploits Monte Carlo simulation to evaluate the impact of this uncertainty. The workflow of the simulation is summarized in Alg. 2.

```
1 n = 0;
2 while n < MAX_SIM do
3 |   Sample all probabilities with error margin;
4 |   Create a model using the sampled values;
5 |   Perform the target analysis;
6 |   Save the computed metric;
7 |   n = n+1;
8 end while
9 return Distribution of the computed metric
```

Alg. 2. Monte Carlo simulation to account for error margins in the cpt.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

It consists on a repeated execution of the target analysis (e.g., AVF computation) by randomly sampling at each iteration all conditional probabilities within their error margins. The proposed Monte Carlo analysis enables to understand how the reliability metric is distributed based on the uncertainty of the parameters of model. The sampling process for each column of a generic cpt with three possible states is reported in Fig. 7.

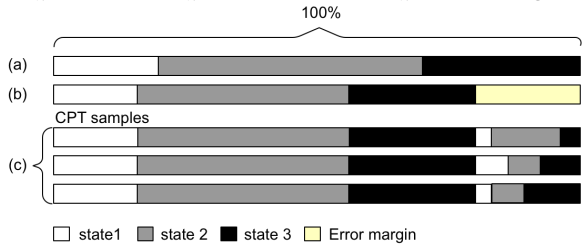

Fig. 7. Sampling process of one column of a cpt

The procedure starts from the measured probabilities for each state (a). The probabilities are scaled to consider that there is a certain uncertainty due to the error margin of each measure (b). Several samples for the same column are then generated randomly assigning portions of the error margin area to the possible states (c). This procedure is repeated for each column of the cpt and for all tables of the network.

4 Experimental Results

This section describes a set of experiments to show how SyRA can be used to analyze the resilience of complex microprocessor-based systems to soft error.

4.1 Experimental design

Each use-case considered in this experimental setup is a microprocessor-based system. The characteristics of the selected use-cases are:

- *Hardware architecture*: three relevant out-of-order superscalar microprocessor architectures are considered: (i) ARM Cortex-A9 (A9), (ii) ARM Cortex-A15 (A15) and (iii) Intel-like i7-skylake (x86). Table 2 reports their relevant architectural parameters.
- *Operating system*: a mix of bare-metal and Linux applications.
- *Application software*: eight benchmarks from the MiBench[1] suite [10], two realistic industrial applications and one open-source HPC application.

The first industrial application is a control application from the avionic domain (rta). It is a real-time application with different activation periods for different tasks running on the Linux operating system. It comprises about 16K lines of C code including 567 functions. The second industrial application (bm) is a bare metal application for DC motor controllers. It comprises about 3K lines of C code including 14 functions. Finally, the third realistic application is an open source software to solve hyperbolic equations on dynamically changing fully-adaptive conforming 2D triangular grids (hpc[2]). It is used in applications such as simulations of the propagation of a tsunami waves over the sea. With about 300K lines of C code and 419 functions this represents a very complex application able to stress the scalability of SyRA.

TABLE 2
Single-core hardware architecture parameters

	A9	A15	X86
L1 I/D	32KB	32KB	32KB
L2	512KB	1MB	1MB
RF	56 32bit reg	128 32bit reg	168 64bit reg
SQ	8 32bit reg	16 32bit reg	72 64bit reg.
Tech. Node	65/45nm	32/28nm	14 nm FinFET
Clock	0.8-2GHz	1-2GHZ	4GHz

For all fault-injection campaigns, at all layers, we use statistical fault sampling as described in [9] in order to reach a 5% error margin with 99% confidence level for all estimated parameters. Finally, to account for the uncertainty of the model we performed Monte Carlo simulation as described in Section 3.3 with 100,000 samples.

4.2 Reliability analysis

To show the capabilities of SyRA, we start by assessing the accuracy of the estimated reliability metrics. Fig. 8 compares, for each use-case, the AVF_S computed with our model to the one computed with a full micro-architectural statistical fault injection campaign. We selected AVF_S for the comparison since it is the main metric on which the other metrics are based. Micro-architectural fault injections are performed using GeFIN [21]. GeFIN is a micro-architecture level fault injector used in previous studies for reliability assessment [21][38][39][40][42][53]. Microarchitecture-level injection (to which we compare SyRA in terms of speed and accuracy) is a good reference for measuring the AVF or FIT rate of the memory arrays considered in this study since they are precisely modeled in the Gem5 simulator on which GeFIN is based. Moreover, GeFIN allows us to run the applications on top of a real operating system thus emulating a complete and realistic execution environment.

In Fig. 8, all applications except hpc are analyzed on A9. Hpc is analyzed on x86 since it cannot be executed on a low-end pro-

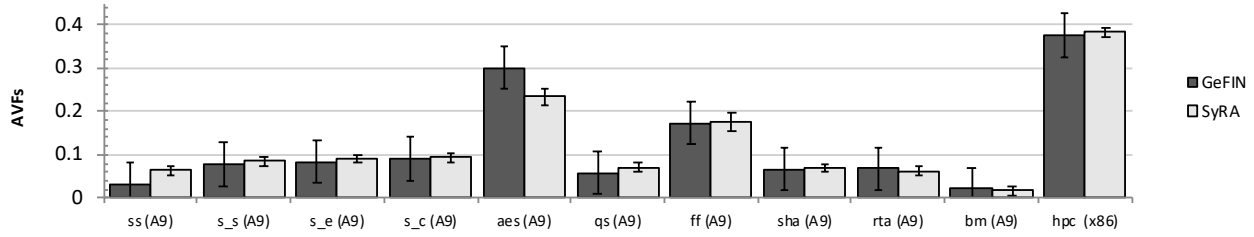

Fig. 8. Comparison of AVF_S estimated with SyRA and GeFIN considering the joint contribution of RF, L1D, L1I, L2, SQ.

[1] (1) string search (ss), (2) susan image smoothing (s_s), (3) susan image edge detection (s_e), (4) susan image corner algorithm (s_c), (5) rijndael encoding performing AES encryption (aes), (6) quick sort (qs), (7) Fast Fourier Transform (ff), (8) Secure Hash Algorithm (sha).

[2] https://www5.in.tum.de/sierpinski/index.php

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

cessor such as the A9. As shown in the figure, estimations provided by SyRA are consistently contained within the error margin of the estimations computed using GeFIN, even when considering the most complex use-cases. The aes benchmark is the only outlier reporting the highest deviation of 7 percent points (pp), which is slightly higher than the 5% error margin of GeFIN. We have identified the source of this deviation. Aes stores its key as a constant array. GeFIN is able to inject faults in this data during the fault injection campaign as it is loaded on the system memory. However, our LLVM analysis workflow currently does not support the corruption of constant structures. This is a limitation of the available tool-chain that needs to be improved in future releases.

To further strengthen the validation of the accuracy delivered by SyRA, we exploited a commercial Cortex-A9 RTL model and an RTL fault injection tool from Yogitech s.p.a. [56] to set up an RTL fault injection campaign. As discussed in the introduction of this paper RTL fault injection has important strengths in terms of accuracy and number of modeled hardware structures but introduces limiations in terms of simulation throughput. Instead, being based on a micro-architecture level model of the hardware, SyRA limits the type of hardware structures that can be analyzed but better scales with the complexity of the software stack. The goal of this comparison is not to decide which method is better. Both can be applied in a complementary way. Instead, we want to demonstrate that, for the common ground, estimations provided by SyRA are in line with precise RTL simulations.

Due to the complexity of these simulations, not all hardware components could be analyzed. Very large memory arrays require significant computational effort to be precisely simulated during fault injection at the RTL level. For this reason, the RTL campaign was limited to two components: RF and L1D. Not all software applications were analyzed in this comparison. To fairly compare results with the one delivered by SyRA applications need to be simulated for their full duration. Performing this at the RTL level for complex benchmarks is excessively time consuming. The application code was modified to work as bare metal and I/O transactions were emulated by the simulator. We performed about 1000 injections for evey hardware structure in order to reach a 5% error margin with 99% confidence level for all estimated parameters. The analysis with SyRA was executed in order to resemble the characteristics of the new setup.

Fig. 9 reports the results of this analysis. As in the comparison with GeFIN, estimations performed by SyRA are consistently within the error margins of those obtained using RTL fault injection, further confirming the accuracy of the proposed model.

Once the accuracy of the model has been discussed, the capability of the framework can be further stressed. Fig. 10 shows the use of SyRA to perform hardware design exploration. It compares the AVF_S of rta when executed on the A9 or A15 microprocessor. The figure also reports the contribution of each hardware structure to the AVF_S. As expected the two architectures report different AVF_S and SyRA is able to quantify this difference (about 4pp). This difference can be further analyzed by looking at the contribution of each hardware block. In details, it can be in good part explained by the contribution of L2 and RF. Looking at Table 2 we can see that A15 has a bigger L2 cache and a bigger register file. These blocks are therefore less utilized and the probability that a fault will target a live entry is lower. However, the bigger size increases the fault probability. This trade-off can be analyzed looking at the FIT rate of the system (λ_S).

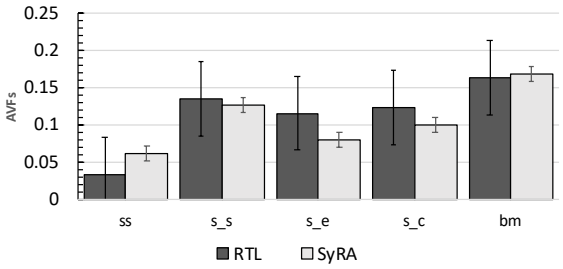

Fig. 9. Comparison of AVF_S estimated with SyRA and RTL fault injection considering the joint contribution of RF and L1D with applications configured to work as bare metal.

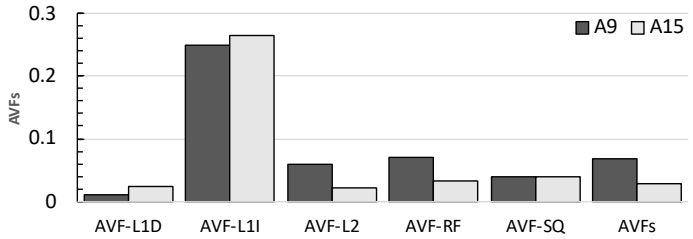

Fig. 10. Comparison of the rta AVF executed on A9 and A15.

To compute λ_S, Table 3 reports SER_C for a single 6T SRAM cell for the different technologies considered in Table 2. These results have been computed resorting to the cell characterization procedure described in Section 2.1.

TABLE 3
Soft error rate for 1 6T SRAM cell computed with typical conditions (1V, 50C, New York at sea level).

	14nm FinFET	28nm planar	32nm planar	40nm planar	65nm planar
SER_C	8.55E-9	1.2E-3	1.19E-3	1.14E-3	1.09E-3

Fig. 11 reports λ_S for all use-cases executed on the A9. The figure investigates the impact of the two production technologies commercially available for this microprocessor. As expected, since the raw error rate per bit of the two technologies is similar, the impact of the technology on the failure rate of the system (λ_S) is marginal. Nevertheless, it is interesting to see the

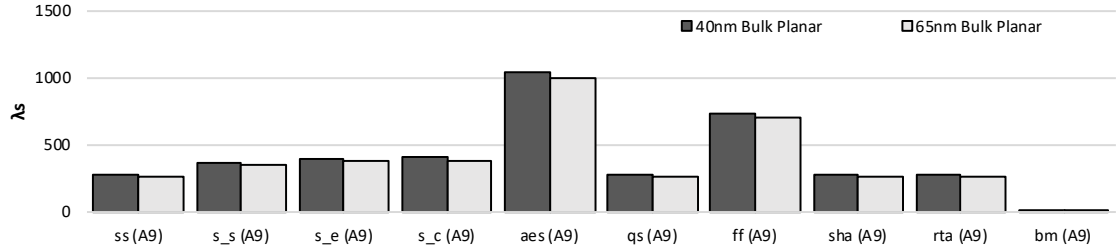

Fig. 11. λ_S for all benchmarks executed on different commercial technologies implementations of A9.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

significant variance of λ_S depending on the executed application that follows the same trend as AVF_S. It is especially interesting to note that bm has by design a very low λ_S. This bare-metal application is inherently highly resilient to soft errors. Therefore, it would require low design effort to be used in mission-critical applications. This type of analysis may help saving resources while designing the system.

If we repeat the same analysis for hpc executed on x86, we obtain $\lambda_S = 8.632\text{E}-3$. This is significantly lower that the one of all other systems. This is a use-case in which the benefit of the technology that has a raw error-rate several orders of magnitude lower than the others is able to compensate the higher AVF.

Architectural and technological design exploration can be put together to understand their joint benefit. Fig. 12 compares λ_S for rta when executed on two different microprocessor architectures featuring different technologies. It is interesting to note that, the difference in the AVF of the two architectures reported in Fig. 10, considering that the technologies have quite similar raw error rates is compensated by the bigger size of the memory arrays of the A15. This is an interesting example to show how, from the one hand changing the architecture gives benefits in terms of AVF but, at the same time it makes the system more susceptible to raw errors. Thanks to the developed tool-chain we are able to carefully analyze this trade-off and take the appropriate design decisions.

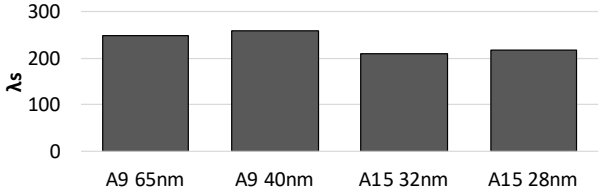

Fig. 12. Comparison of λ_S for rta executed on different microprocessors characterized by different architectures and fabrication technologies.

As discussed in Section 3.1, λ_S is a pure reliability metric that does not fairly compare CPU architectures that have different clock frequencies. To provide additional insides on the reliability/performance trade-off SyRA offers the possibility to compute the EPF. We exploit this metric to compare rta and bm under two implementations: (1) A9 65nm Bulk Planar CMOS clocked at 800-MHz and (2) A15 28nm Bulk Planar CMOS clocked at 2.5GHz. Fig. 13 reports the computed EPF for the two applications. The figure reports a very interesting result.

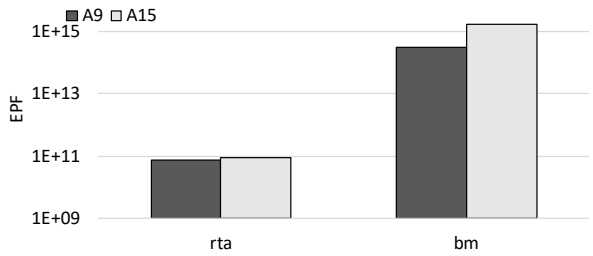

Fig. 13. Comparison of the EPF (logarithmic scale) of rta and bm executed on two ARM microprocessors: A9 - 65nm clocked at 800-MHz and A15 - 28nm clocked at 2.5GHz.

While the EPF of bm increases of one order of magnitude, the EPF of rta remains almost constant. This behavior is explained by the behavior of the two applications. Rta is a real-time application. Even by selecting a faster microprocessor its execution time remains constant and therefore the EIT remains constant. The only change obtained moving from the A9 to A15 is the lower λ_S as reported in Fig. 12. Differently, the performance of bm is affected by the higher performance of the A15, with a significant impact on the EPF. These two very interesting examples deriving from the analysis of two of the selected use-cases clearly show the benefit of the EPF metric that can be computed using SyRA. Using EPF the trade-off between performance and reliability can be easily analyzed in a single measure providing a significant support for the designers.

4.3 Diagnostic analysis

One of the main features offered by SyRA is the possibility of supporting quick design exploration to optimize the target system. For this purpose, we study the application of a cross-layer combination of two relevant protection mechanisms to the most complex of the available use-cases (hpc). The two protection mechanisms are ECC [43] that works at the architecture layer, and Fault Tolerance via control and data flow assertions [44] that works at the software layer. The aim of the diagnostic analysis is to identify the weak portions of the system and to understand how the reliability changes when these portions are protected. We want to remark that the aim of these experiments is not to demonstrate which technique or which combination of techniques is superior. Being SyRA a framework for early reliability analysis, we are not interested in delivering a precise implementation of all protection mechanisms in the different simulation environments, as this will be costly and time consuming. At this stage, we are interested in estimating the impact of the application of the protection mechanisms on the system to actually decide whether it is worth or not their application. Therefore, Table 4 characterizes the two protection mechanisms with information extracted from the literature. Of course, if an instrumented version of the software is available, or a library of protection mechanisms is implemented in Gem5, precise results can be obtained. However, this is out of the scope of the paper.

TABLE 4
Characterization of the selected hardware and software fault tolerance techniques

Technique	AVF reduction
ECC	100%
Fault Tolerance via control and data flow assertions	85%

We exploited SyRA's to analyze 600 variations of the original system in which randomly selected groups of hardware/software components have been protected using the two considered protection techniques. Protecting a component means updating its cpt according to the masking capability of the protection technique calculating a new value for AVF_S. Fig. 14 shows the results of this analysis.

The horizontal axis reports the number of protected components. For instance, when the number of protected components is equal to 5 it means that 5 nodes of the system have been randomly selected to be protected using the two selected protection mechanisms. Candidate nodes are the five memory arrays considered in the HAL and all functions of the software applications. The protection mechanisms described in Table 4 are applied to the full component (e.g., if RF is selected the ECC is applied to all its entries). We use a random selection of the components to give an overview of the characteristics of the design space that SyRA can explore. The vertical axis instead reports AVF_S. Each gray point represents a considered system. At a first

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

look this figure shows that, protecting a high number of components does not always translates into benefits on the AVF_S. Even considering a fixed number of protected components, depending on the selected nodes the AVF_S changes. Interestingly, we marked in red a set of systems in which the protected components have been selected using SyRA's diagnostic analysis as follows. The original system has been incrementally protected through several iterations. At each iteration, the most critical component of the system has been identified and protected (see Section 3.2). The figure shows that, by protecting just 9 components selected carefully, it is possible to reach a very low AVF equal to 0.008 (green dot).

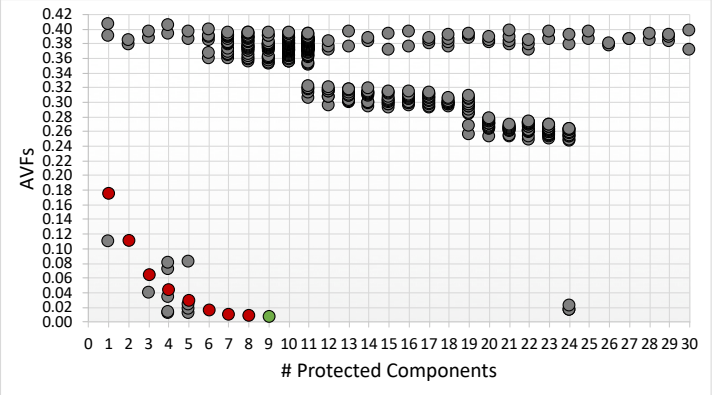

Fig. 14. SyRA diagnostic analysis for the hpc benchmark. Gray dots show the AVF_S of 600 combinations of randomly protected components. Red dots identify the AVF_S obtained by incrementally protecting the worst-case component identified through SyRA diagnostic analysis. The green dot represents the best system.

The example provided in this section is a simple demonstration of the potential that SyRA offers to the designers that want to tune the system to the specific applications.

The capability of SyRA to preform sensitivity analysis and to estimate quickly the effect of different protection mechanisms on selected portions of a system suggests that it could be exploited as a building block for the construction of automatic algorithms to perform design space exploration to identify best combinations of cross-layer protection mechanisms to apply. Nevertheless, this requires strategies to efficiently explore the space of solutions which are out of the scope of this paper.

4.4 Performance

After showing the capability of SyRA, it is interesting to analyze its performance. Fig. 15 reports this information expressed in hours of simulation on a Xeon workstation running simulations with 12 parallel threads. The figure shows the full time required to perform simulations, build the model, and compute the metrics. The only time not considered here is the characterization of the technology. The time to simulate a single cell (e.g., a SRAM cell) with a specific technology, voltage and temperature ranges between 10 to 20 minutes depending on the size of the cell. This time is mainly occupied by SPICE simulations. The analysis of a full technology for a single cell considering all different operational points reported in this paper can take from 1 to 3 hours. If several technologies should be analyzed the cost of the analysis can take some days. Even if this part of the analysis can be costly when several alternatives must be evaluated (i.e., different technologies, cell architectures, operational points, etc.), it is worth to remember that, once a single cell is characterized, the obtained data can be stored and reused several times to analyze different design.

Fig. 15 clearly shows that the simulation time for SyRA is significantly reduced with respect to GeFIN even when considering complex applications (on average 68% lower). This is a key requirement during the design phase to explore different cross-layer reliability techniques. It is important to remark that the simulation time has been computed resorting to a single workstation. Its absolute value can be reduced exploiting parallel machines or machines with higher parallelism. Since all applications run on the same hardware the difference between the considered benchmarks is mainly due to the complexity of the software. Larger software applications require longer simulation time during the HAL/SAL characterization. Finally, they generate larger models that require more time to be solved.

Fig. 16 shows the different contributions to the global analysis

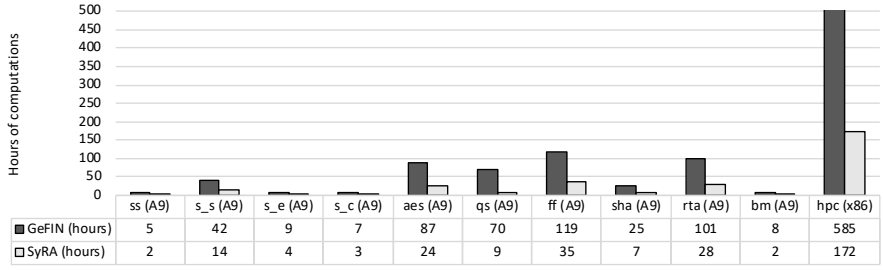

Fig. 15: Comparison of the simulation time (hours of simulation) required to compute AVF_S using SyRA and GeFIN.

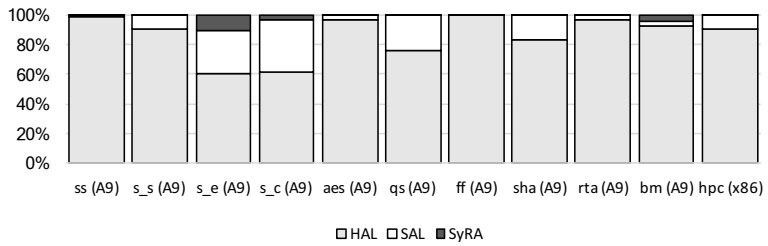

Fig. 16: Distribution of the simulation time between the HAL/SAL characterization and the model analysis (SyRA).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

time. HAL and SAL indicate the percentage of time required to perform simulations to compute the cpt of the two layers, while SyRA indicates the time required to build and analyze the Bayesian model and to compute the metrics.

The benefit of using SyRA is even more evident when comparing its performance with respect to RTL fault injection. In this case, the analysis time for the benchmarks considered in Fig. 9 was 2 orders of magnitude higher.

5 Related work

The analysis of the reliability at the circuit and technology layer is a very well-established research field. A comprehensive survey of different gate-level circuit reliability analysis techniques can be found in [45].

The study of the contribution of the hardware architecture to the resilience of a system to soft errors has received significant attention by the research community. Several AVF estimation methods are based on Architectural Correct Execution (ACE) analysis using architecture level simulators [20][33][46]. These approaches are complex. They require significant modifications to the simulators to track resources during the execution of the program. Therefore, they are limited to the analysis of small programs. Apart for the complexity, accuracy is a general limitation of these approaches. A 7x AVF over-estimation is reported in [19]. Even with refined approaches, which require additional complexity in the simulation, ACE analysis still provides 3x overestimation. This has a detrimental effect on the system leading to system over-design [20]. According to [20], the conservatism of ACE analysis is due to two key sources: (1) lack of details in the employed abstract performance models and (2) the presence of what we call Y-Bits that are a result of the single-pass simulation methodology that is typical of ACE analysis. Some transient faults dramatically alter the course of execution in a processor, yet they do not affect the correct execution. These are Y-Bits. SyRA simulates the full propagation of faults from hardware up to software. At the hardware layer, it precisely models the hardware structures in which soft errors are injected and the propagation of each soft error is simulated until the end of the application to evaluate its fate. Therefore, it does not suffer from the same approximation of ACE analysis

The first attempt to model the contribution of the software to the AVF of the system is provided in three seminal papers by Sridharan and Kaeli [47][48][49]. They introduce the concept of Program Vulnerability Factor (PVF) to quantify the portion of the AVF that can be attributed to the executed software. This concept has been further extended in [49] with the introduction of the concept of the System Vulnerability Stack. The System Vulnerability Stack is a significant advance towards the definition of a cross-layer system reliability model. However, its main drawback is that it oversimplifies the definition of the layers. In particular, the basic assumption is that the layers are statistically independent from each other. This allows to compute the AVF of the system simply as the product of the vulnerability factors of each layer. Moreover, the layers are not further split into their composing components, preventing a fine-grained analysis of the architecture of the system.

Another interesting solution that considers the impact of the application software running on embedded microprocessors was discussed in [50]. As in the case of ACE analysis, it is based on the use of program traces. Therefore, it suffers from inaccuracies due to the fact that it cannot capture important masking effects introduced during dynamic execution of the software. Moreover, it is limited to bare metal applications.

Rehman et al. [51] introduce: the Instruction Vulnerability Index (IVI) that calculates the vulnerability of an instruction during its execution in the pipeline, the Function Vulnerability Index (FVI) that joins the IVI of all instructions of a function into a single index, and the Application Vulnerability Index (AVI) that joins the FVI of the functions of an application into a single index. This is an interesting idea to provide indications on how to optimize the software to improve reliability. The main limitation of the proposed reliability analysis flow is that the contribution of the hardware is not well defined. Instructions are labeled into critical and not-critical a priori and not on an application base. Moreover, the computation of the failure probability and incorrect output probability of each instruction is demanded to fault-injection experiments without detailing how this process should be carried out. Eventually, the contribution of the program level error masking is not considered. The same group in [52] shows how the contribution of the program-level error masking can be analyzed to generate reliability optimized programs, thus overcoming some of the limitations of the previous publication.
Overall, SyRA makes a step forward. In a single framework, we are able to model the technology, the hardware micro architecture and the software architecture including the contribution of the program level error masking.

A very comprehensive study aimed at understanding how different cross-layer design options influence the reliability of a system has been presented by Cheng et al. in [2]. They performed an impressive and massive simulation campaigns identifying combinations of selected protection techniques that overall work very well together across different hardware designs (two processors SPARC Leon3 and Alpha IVM), different software benchmarks and 798 cross-layer combinations. 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 provides a clear picture of how cross-layer reliability techniques can work but still every design has its own peculiarity and, therefore, the same analysis should be repeated every time a new design is constructed. The complexity of the simulations presented in the paper guarantees high accuracy, but, in our opinion is not affordable in the early stages of the design. In these stages, design decisions should be taken based on reasonably accurate models but still consuming affordable computing resources as proposed in SyRA.

A first attempt of using a Bayesian cross-layer reliability model to estimate the system level failure in time was presented by the authors of this paper in [53]. Despite providing preliminary good results, the model proposed in the paper is simplified especially when it comes to the way the different layers are interfaced. The basic improvements of SyRA with respect to this work have been already discussed in the introduction of this paper. We would like to mention here that, without the improvements provided in this paper, the analysis of complex applications such as rta and hpc would have created scalability issues. Since the possibility of analyzing complex systems running complex applications is a key characteristic of our framework, working on its scalability in order to handle large applications has been a significant step forward.

7 Conclusions

In this paper we presented SyRA, a complete framework for early system reliability analysis for cross-layer soft errors resilience of microprocessor systems. The proposed framework provides a powerful tool to analyze the resilience of a system to neutron induced soft errors, thus understanding its weak components. These components can be good candidates to take focused actions when implementing fault tolerance mechanisms.

The approach is defined to analyze the system in the early stages of the design flow when the designer needs to take key decisions about the implementation of the system. Therefore, speed of the analysis has been one of the driving factors while developing the approach. We demonstrated the capability of SyRA through a very extensive set of experiments using realistic microprocessor architectures, demonstrating the accuracy of the tool. Besides its accuracy in reliability assessments, one of the key capabilities of SyRA is the possibility to perform early diagnostic analysis to identify reliability-critical components of the system and to support design exploration to quickly evaluate the effect that different cross-layer protection mechanisms at the technology, hardware and software layer will have on the system's reliability.

6 Acknowledgment

This paper has been fully supported by the 7th Framework Program of the European Union through the CLERECO Project, under Grant Agreement 611404. Interested researchers willing to contribute to the extension to this framework can visit the project's website at http://www.clereco.eu. This paper is part of the results of a three-year Ph.D. project on cross-layer reliability analysis [54].

7 References

[1] J. A. Abraham, "Cross-layer resilience: are high-level techniques always better?" in 2016 IEEE International High Level Design Validation and Test Workshop (HLDVT), Oct 2016, pp. 78-78.

[2] 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.

[3] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, Tahoori, and N. Wehn, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends," in Proceedings of the 50th Annual Design Automation Conference, ser. DAC '13. New York, NY, USA: ACM, 2013, pp. 99:1-99:10.

[4] N. P. Carter, H. Naeimi, and D. S. Gardner, "Design techniques for cross-layer resilience," in Proceedings of the Conference on Design, Automation and Test in Europe, ser. DATE '10. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2010, pp. 1023-1028.

[5] A. DeHon, H. M. Quinn, and N. P. Carter, "Vision for cross-layer optimization to address the dual challenges of energy and reliability," in 2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010), March 2010, pp. 1017-1022.

[6] D. W. Coit, T. Jin, and N. Wattanapongsakorn, "System optimiza-tion with component reliability estimation uncertainty: a multi-criteria approach," IEEE Transactions on Reliability, vol. 53, no. 3, 369-380, Sept 2004.

[7] A. Vallero, S. Tselonis, N. Foutris, M. Kaliorakis, M. Kooli, A. Savino, G. Politano, A. Bosio, G. D. Natale, D. Gizopoulos, and S. D. 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.

[8] L. Entrena, M. Garcia-Valderas, R. Fernandez-Cardenal, A. Lin-doso, M. Portela, and C. Lopez-Ongil, "Soft error sensitivity evaluation of microprocessors by multilevel emulation-based fault injection," IEEE Transactions on Computers, vol. 61, no. 3, pp. 313- 322, March 2012.

[9] 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.

[10] 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.

[11] A. Filieri, C. Ghezzi, V. Grassi, and R. Mirandola, Reliability Analy-sis of Component-Based Systems with Multiple Failure Modes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1-20.

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

[13] P. Hazucha and C. Svensson, "Impact of CMOS technology scaling on the atmospheric neutron soft error rate," IEEE Transactions on Nuclear Science, vol. 47, no. 6, pp. 2586-2594, Dec 2000.

[14] 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 Exhibi-tion (DATE), March 2016, pp. 217-222.

[15] S. Ozdemir, N. Aymerich, M. Riera, R. Canal, A. Gonzalez, M. Kaliorakis, S. Tselonis, N. Foutris, D. Gizopoulos, S. Di Carlo, P. Prinetto, "D2.2.2 - Characterization of failure mechanisms for future systems," [online] http://www.clereco.eu/images/deliverables/D2.2.2_Characterization-of-failure-mechanisms-for%20future-systemsv1.0.pdf

[16] J. F. Ziegler, "Terrestrial cosmic rays," IBM journal of research and development, vol. 40, no. 1, pp. 19-39, 1996.

[17] J. Wilkinson, "Soft-error testing resources," [online] http://ww.seutest.com.

[18] M. S. Gordon, P. Goldhagen, K. P. Rodbell, T. H. Zabel, H. H. K. Tang, J. M. Clem, and P. Bailey, "Measurement of the flux and energy spectrum of cosmic-ray induced neutrons on the ground," IEEE Transactions on Nuclear Science, vol. 51, no. 6, pp. 3427-3434, Dec 2004.

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

[20] N. J. Wang, A. Mahesri, and S. J. Patel, "Examining ACE analysis reliability estimates using fault-injection," in Proceedings of the 34th Annual International Symposium on Computer Architecture, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 460-469.

[21] 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.

[22] 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, 1-7, Aug. 2011.

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

[24] S. Mukherjee, Architecture design for soft errors. Morgan Kaufmann, 2011.

[25] N. J. Wang and S. J. Patel, "ReStore: Symptom-based soft error detection

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2887225, IEEE Transactions on Computers

in microprocessors," Dependable and Secure Computing, IEEE Transactions on, vol. 3, no. 3, pp. 188-201, 2006.

[26] S. Mirkhani, M. Lavasani, and Z. Navabi, "Hierarchical fault simulation using behavioral and gate level hardware models," in Test Symposium, 2002. (ATS'02). Proceedings of the 11th Asian. IEEE, 2002, pp. 374-379.

[27] A. Zagorecki and M. J. Druzdzel, "Knowledge engineering for bayesian networks: How common are noisy-max distributions in practice?" IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 43, no. 1, pp. 186-195, 2013.

[28] A. Thomas and K. Pattabiraman, "LLFI: An intermediate code level fault injector for soft computing applications," in Workshop on Silicon Errors in Logic System Effects (SELSE), 2013.

[29] V. C. Sharma, A. Haran, Z. Rakamaric, and G. Gopalakrishnan, "Towards formal approaches to system resilience," in Dependable Computing (PRDC), 2013 IEEE 19th Pacific Rim International Sympo-sium on. IEEE, 2013, pp. 41-50.

[30] D. Wu, M. A. Hennell, D. Hedley, and I. J. Riddell, "A practical method for software quality control via program mutation," in [1988] Proceedings. Second Workshop on Software Testing, Verification, and Analysis, Jul 1988, pp. 159-170.

[31] H. Cho, S. Mirkhani, C. Y. Cher, J. A. Abraham and S. Mitra, "Quantitative evaluation of soft error injection techniques for robust system design," 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, 2013, pp. 1-10.

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

[33] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnera-bility factors for a high-performance microprocessor," in Microar-chitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on. IEEE, 2003, pp. 29-40.

[34] M. J. Druzdzel, "SMILE: Structural modeling, inference, and learn-ing engine and genie: a development environment for graphical decision-theoretic models," in Aaai/Iaai, 1999, pp. 902-903.

[35] C. Huang and A. Darwiche, "Inference in belief networks: A procedural guide," International journal of approximate reasoning, vol. 15, no. 3, pp. 225-263, 1996.

[36] C. Yuan and M. J. Druzdzel, "An importance sampling algorithm based on evidence pre-propagation," in Proceedings of the Nine-teenth conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann Publishers Inc., 2002, pp. 624-631.

[37] Y. Weiss, "Correctness of Local Probability Propagation in Graphical Models with Loops," in Neural Computation, vol. 12, no. 1, pp. 1-41, Jan. 1 2000.

[38] M. Kaliorakis, D. Gizopoulos, R. Canal, and A. Gonzalez, "MeRLiN: Exploiting Dynamic Instruction Behavior for Fast and Accurate Microarchitecture Level Reliability Assessment," 44th Annual International Symposium on Computer Architecture (ISCA '17), pp. 241-254.

[39] A. Chatzidimitriou, M. Kaliorakis, S. Tselonis and D. Gizopoulos, "Performance-aware reliability assessment of heterogeneous chips," 2017 IEEE 35th VLSI Test Symposium (VTS), Las Vegas, NV, 2017, pp. 1-6.

[40] 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.

[41] A. Vallero, S. Di Carlo, S. Tselonis, and D. Gizopoulos, "Microar-chitecture level reliability comparison of modern gpu designs: First findings," in Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on. IEEE, 2017, pp. 129-130.

[42] A. Chatzidimitriou, M. Kaliorakis, D. Gizopoulos, M. Iacaruso, Pipponzi, R. Mariani, and S. D. Carlo, "RT level vs. microarchitecture-level reliability assessment: Case study on ARM(R) cortex(r)-a9 cpu," in 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), June 2017, pp. 117-120.

[43] H. Amrouch and J. Henkel, "Self-immunity technique to improve register file integrity against soft errors," in Proc. 24th Int. Conf. VLSI Des., 2011, pp. 189–194.

[44] 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.

[45] R. Xiao and C. Chen, "Gate-level circuit reliability analysis: A survey," VLSI Des., vol. 2014, pp. 4:4-4:4, Jan. 2014. [Online]. Available: http://dx.doi.org/10.1155/2014/529392

[46] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "SoftArch: an architecture-level tool for modeling and analyzing soft errors," in 2005 International Conference on Dependable Systems and Networks (DSN'05), June 2005, pp. 496-505.

[47] V. Sridharan and D. Kaeli, "Eliminating microarchitectural de-pendency from architectural vulnerability," in High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on, Feb 2009, pp. 117-128.

[48] V. Sridharan and D. R. Kaeli, "Using pvf traces to accelerate avf modeling," in Proceedings of the IEEE Workshop on Silicon Errors in Logic System Effects, 2010.

[49] V. Sridharan and D. R. Kaeli, "Using hardware vulnerability factors to enhance AVF analysis," in Proceedings of the 37th Annual International Symposium on Computer Architecture, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 461-472. [Online]. Available: http://doi.acm.org/10.1145/1815961.1816023

[50] 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.

[51] S. Rehman, M. Shafique, F. Kriebel and J. Henkel, "Reliable software for unreliable hardware: Embedded code generation aiming at reliability," 2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Taipei, 2011, pp. 237-246.

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

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

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

[55] G. F. Cooper, *"The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks (Research Note)"*, Artif. Intell., vol. 42, no. 2-3, Mar 1990, pp.393-405

[56] A. Benso, A. Bosio, S. D. Carlo and R. Mariani, "A Functional Verification based Fault Injection Environment," *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, Rome, 2007, pp. 114-122.

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.

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 an assistant professor in 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.

Athanasios Chatzidimitriou (S'16) received his B.Sc. degree on Informatics engineering from Technological Educational Institute of Athens, Greece and his M.Sc. degree on Embedded Computing Systems from University of Piraeus, Greece. Currently, he is a Ph.D. student in the Department of Informatics and Telecommunications at University of Athens. His research interests focus on microprocessor reliability.

Manolis Kaliorakis (S'14) received his B.Sc. degree in Electrical and Computer Engineering from Democritus University of Thrace (DUTH), Greece and his M.Sc. degree in Microelectronics and Integrated Circuits Design from Department of Informatics & Telecommunications, University of Athens, Greece. Currently, he is a Ph.D. student in the Department of Informatics and Telecommunications at University of Athens. His research interests focus on computer architecture, microprocessor reliability, testing and fault tolerance

Maha Kooli (M'14) Maha Kooli (S'14) received her Ph.D. in Computer Engineering and Microelectronics from the University of Montpellier, France in 2016, and her engineer diploma in computer science and applied mathematics from the engineering school ENSEEIHT-INP Toulouse, France in 2013. From December 2016 to March 2017, she was a postdoc at the LIRMM laboratory in Montpellier with the National Scientific Research Center (CNRS) of France. Since April 2017, she has been a post doc at CEA Leti in Grenoble. Her research interests are related to reliability, software systems, in-memory computing and security. She participated in CLERECO project (FP7).

Marc Riera Villanueva received his B.Sc. degree in Computer Engineering in 2013, and his M.Sc. degree in MIRI: High Performance Computing in 2015, both from Universitat Politecnica de Catalunya (UPC - BarcelonaTech). He joined the ARCO research group at UPC in July 2014 and he is currently pursuing a Ph.D. at UPC. His research focuses on the area of Reliability and recently started working on Resilient and Low Power Accelerators for Cognitive Computing.

Marti Anglada received his B.Sc. degree in Computer Engineering in 2013, and his M.Sc. degree in MIRI: High Performance Computing in 2015, both from Universitat Politecnica de Catalunya (UPC - BarcelonaTech). He joined the ARCO research group at UPC in July 2014 and he is currently pursuing a Ph.D. at UPC. His research is focused on low-power, resilient architectures.

Giorgio Di Natale received the Ph.D. in Computer Engineering from the Politecnico di Torino in 2003 and the HDR (Habilitation a Diriger les Recherches) in 2014 from the University of Montpellier II (France). He is currently a researcher for the National Research Center of France at the LIRMM laboratory in Montpellier. His research interests include: hardware security and trust, reliability, fault tolerance, test. He is the Action Chair of the COST Action IC1204 (TRUDEVICE) on Trustworthy Manufacturing and Utilization of Secure Devices. He is a Golden Core member of the Computer Society.

Alberto Bosio Alberto Bosio got his PhD from the Politecnico di Torino (Italy) in 2006. From September 2018 he is a Full Professor at INL - Ecole Centrale de Lyon, France. He is a co-author of 1 book, 35 international journal papers, 3 patents, 7 invited papers, 2 embedded tutorials and more than 100 papers in international conferences. He served as committee and organizing member in several international conferences as well as reviewers for many international journals. He is a member of the IEEE and the Chair of the European Test Technical Technology Council (ETTTC).

Ramon Canal received the M.Sc. and Ph.D. degrees from the Universitat Politecnica de Catalunya, Barcelona, Spain, where he joined the Faculty of the Computer Architecture Department in 2003. He was with Sun Microsystems in 2000, and was a Fulbright Visiting Scholar with Harvard University, Cambridge, MA, USA, in 2006 and 2007. His research focuses on system reliability and the memory hierarchy. He has an extensive list of publications and several invited talks. He has been a Program Committee Member in several editions of HPCA, ISCA, MICRO, HiPC, IPDPS, ICCD, ICPADS, and CF. He has been the Co-General Chair of IOLTS 2012 and HPCA 2016.

Antonio Gonzalez (Ph.D. 1989) is a Full Professor at the Computer Architecture Department of the Universitat Politecnica de Catalunya, Barcelona (Spain), and the director of the Microarchitecture and Compiler research group. He was the founding director of the Intel Barcelona Research Center from 2002 to 2014. His research has focused on computer architecture, compilers and parallel processing, with a special emphasis on microarchitecture and code generation. He has published over 350 papers and has served as associate editor of five IEEE and ACM journals, program chair for ISCA, MICRO, HPCA, ICS and ISPASS, general chair for MICRO and HPCA, and PC member for more than 100 symposia. He is an IEEE Fellow.

Dimitris Gizopoulos (S'93-M'97-SM'03-F'13) is professor at the Department of Informatics & Telecomm., University of Athens where he leads the Computer Architecture Lab. His research focuses on the dependability, performance and energy of computer architectures. Gizopoulos has published more than 170 papers in peer reviewed IEEE and ACM journals and conferences, is an inventor of US patents, author of a book and editor of two books on dependability. He has served as associate editor and special issues guest editor for several IEEE Transactions and Magazines and currently serves on the Editorial Board of IEEE Transactions on Computers and IEEE Transactions on Sustainable Computing. He is an IEEE Fellow, a Golden Core Member of IEEE Computer Society and an ACM Senior Member.

Riccardo Mariani holds a Ph.D. in Microelectronics from the University of Pisa. He is widely recognized as an expert in functional safety and integrated circuit reliability. In his current role as chief functional safety technologist at Intel Corporation, he oversees strategies and technologies for IoT applications that require functional safety, high reliability and performance, such as autonomous driving, transportation and industrial systems. Mariani has contributed to multiple industry standards efforts throughout his career, including leading the ISO 26262-11 part specific to semiconductors. He has also won the SGS-Thomson Award and the Enrico Denoth Award for his engineering achievements. He holds a bachelor's degree in electronic engineering and a Ph.D. in microelectronics from the University of Pisa in Italy.

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 Com-puting cOntinuum. Di Carlo has published more than 170 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.