Cross-Layer System Reliability Assessment Framework for Hardware Faults

A. Vallero, A. Savino, G. Politano, S. Di Carlo
Politecnico di Torino
Torino, (Italy)

A. Chatzidimitriou, S. Tselonis, M. Kaliorakis, D. Gizopoulos,
University of Athens, Greece

M. Riera, R. Canal, A. Gonzalez
Universitat Politècnica de Catalunya
Barcelona, Spain

M. Kooli, A. Bosio, G. Di Natale
LIRMM, Montpellier
Montpellier, France

***Abstract*[1]—System reliability estimation during early design phases facilitates informed decisions for the integration of effective protection mechanisms against different classes of hardware faults. When not all system abstraction layers (technology, circuit, microarchitecture, software) are factored in such an estimation model, the delivered reliability reports must be excessively pessimistic and thus lead to unacceptably expensive, over-designed systems. We propose a scalable, cross-layer methodology and supporting suite of tools for accurate but fast estimations of computing systems reliability. The backbone of the methodology is a component-based Bayesian model, which effectively calculates system reliability based on the masking probabilities of individual hardware and software components considering their complex interactions. Our detailed experimental evaluation for different technologies, microarchitectures, and benchmarks demonstrates that the proposed model delivers very accurate reliability estimations (FIT rates) compared to statistically significant but slow fault injection campaigns at the microarchitecture level.**

Keywords—dependable computing system; reliability modeling; cross-layer reliability

I. Introduction

As we move deeply into the era of nano-scale devices, reliability becomes a key challenge for the semiconductor industry. With transistor size reaching the atomic dimensions, vulnerability to unavoidable fluctuations in the manufacturing process and environmental stress rises dramatically [1]. Failing to meet a reliability requirement may add excessive re-design costs to recover and may have severe consequences on the success of a product [2]. Worst-case design with large margins to guarantee reliable operation has been employed for long time. However, it is reaching a limit that makes it economically unsustainable due to its performance, area, and power costs [3].

Hardware faults may propagate through the hardware (HW) and software (SW) layers of the system stack, reaching the system output, or be masked during this propagation process. Different protection mechanisms can be employed at different layers implementing what is nowadays called *cross-layer reliability* enhancement [4] [5] [6]. Accurately measuring the impact on system reliability of any change in the technology, circuit, microarchitecture and software is a complex design task, involving design teams from all abstraction layers. The task has multiple objectives because reliability must be traded-off against other crucial design attributes such as performance, power, and cost [7]. Unfortunately, tools and models for cross-layer reliability analysis are still at their early stages compared to other very mature design tools (e.g., performance and power optimization tools). The current practice is to rely either on time-consuming gate-level fault injection campaigns or on simplified models that guarantee smaller computation time but deliver very coarse-grain and conservative (i.e. pessimistic) reports of the system reliability [9] [10].

This paper proposes a novel system-level cross-layer reliability assessment framework built on top of a component-based Bayesian model of the target system. In component-based system reliability modeling, the system is more than the sum of its parts. Each component affects globally perceivable properties of the entire system. By carefully integrating parameters from all layers of the system stack we are capable of accurately evaluating the failure rate of the full system. The proposed system-level reliability model and supporting tools deliver several key contributions with respect to current approaches. The target system can be described in terms of components (technology, hardware and software) and the model can efficiently describe how faults and errors propagate through components, accounting for complex interactions among them that are not modeled with simpler combinatorial models (e.g., reliability block diagrams or fault trees [8]). Components can be unplugged from the system during their characterization and the effect of their interaction can be recombined later, thus enabling reuse of information. Moreover, the model is highly parameterized and scalable. It enables including any factor that can potentially affect the reliability of the system (e.g., environmental factors such as location and temperature) by simply adding new variables to the model. The model scales efficiently to complex systems with an analysis time that is significantly shorter than traditional fault injection while maintaining adequate levels of accuracy. Other system-level approaches provide similar execution time [9] [10] but their accuracy is significantly lower and can thus lead to more costly design decisions for reliability. A statistical model itself would be useless for reliability assessment in real applications without supporting instruments to populate the model for a specific system and workload. Along with the model, we therefore present a complete framework comprising a tool-chain able to compute the Failures-In-Time (FIT) rate of the final system based on the proposed system-level reliability model. In this paper, we report experiments for transient faults (soft-errors), but if tools and models to estimate conditional failure probabilities for different classes of faults (i.e., intermittent and permanent

[1] *This paper has been fully supported by the 7th Framework Program of the European Union through the CLERECO Project, under Grant Agreement 611404.*

978-1-4673-8773-6/16/$31.00 ©2016 IEEE

faults) are developed, the proposed model can be used to study their effect as well.

II. Related Work

A. *Architectural level reliability analysis*

The *Architectural Vulnerability Factor* (AVF) of a microprocessor and its estimation has attracted significant attention by the research community. The AVF of a hardware structure is the fraction of faults in it that affect the correct program operation [11]. Most AVF estimation methods are based on offline analysis with architecture level simulators [11] [12] [13]. Offline AVF estimation is a complex process, requiring major modifications to the simulators and many resources to track values and instructions as they travel through microprocessor components. Only a limited number of instructions (short programs) can be analyzed in a reasonable amount of time because of the excessive memory requirements of ACE (Architectural Correct Execution) analysis. Online or real-time AVF estimation has been also presented [14] [15] [16] [17] [18], but it still requires heavy offline simulation and calibration for different workloads. It is not clear to what extent the parameters calibrated for one set of workloads will give accurate estimation for another set. A general drawback shared by all these methods is their AVF over-estimation due to worst-case assumptions. A 7x AVF over-estimation is reported in [9], whereas [10] reports that even a refined ACE-based analysis (which requires even more elaborate modifications of the microprocessor simulator model) leads to up to 3x over-estimation. This leads to over-designed systems. The model and method we propose in this paper aims to contribute to the design of computing systems without excessive costs for reliability.

B. *Accounting for software effects in reliability analysis*

In [20] the authors discussed a first attempt to perform static analysis of a computer system including its software. However, the approach is limited to errors affecting the instruction opcodes before they enter the microprocessor pipeline. Three seminal papers by Sridharan and Kaeli first considered the software layer in the reliability assessment of a system [19] [21] [22]. They propose to compute a Program Vulnerability Factor (PVF) to quantify the portion of AVF that is attributed to a user program. This concept has been further extended in [22] with the introduction of the concept of the *System Vulnerability Stack*. The System Vulnerability Stack is a significant advance towards the definition of a system reliability model accounting for all layers of the system stack. However, it is over-simplified and it considers that layers are statistically independent from each other, to allow computing the global AVF as a simple product of the vulnerability factors of each layer. This is obviously not the case in a real system in which there is a very intricate interaction among the different layers and among components of each layer, and this approximation leads to pessimistic predictions.

Another interesting solution that considers the impact of the application software running on embedded microprocessors was discussed in [23]. Despite the fact that it provides promising results, the method is still limited to transient faults in embedded microprocessors. Moreover, being based on static analysis of code traces, it does not capture important masking effects introduced during dynamic execution.

C. *Bayesian models for reliability estimation*

Bayesian networks models are very useful statistical models employed in many disciplines. Weber et al. in a comprehensive review report more than 200 papers published between 1998 and 2008 in international journals on applications of Bayesian networks in different fields, i.e., dependability, risk analysis and maintenance [24]. Differently from state-space based models such as Markov models [48] Bayesian models are better suited when component-based modeling is required.

Among the different application fields, Bayesian models have been largely used to create software reliability models, i.e., to predict the probability of failure-free software operation for a specified period of time in a specified environment [25] [26] [27] [28]. Software reliability differs from system reliability considered in this paper since it reflects the software design perfection, rather than hardware manufacturing perfection and tolerance of the system to design variability and environmental stresses [29].

Finally, a large set of publications present high-level theoretical Bayesian models to predict system reliability in different fields ([31] [32] [33] and their references). The main drawback of these approaches is that they focus on models optimizations to improve the capability of the analysis but do not provide solutions to collect all parameters required to build the model in a specific application domain. Moving in this direction Vallero at al. propose in [34] to use Bayesian networks to model software traces linking each instruction with the hardware resources required for their execution. While the model is interesting, only faults in the register file of a microprocessor are considered and there is no provision on how parameters for other hardware structures can be computed.

Our model and reliability assessment method is a major step forward. We explicitly consider a Bayesian model in the field of cross-layer reliability focusing on system level effects of hardware faults. Our methodology comprises both the theoretical framework required to properly describe the target system using a Bayesian network and a complete integrated tools framework able to compute all parameters required to feed the model for virtually all realistic cases of hardware and software components.

III. System Level Reliability model

In this paper, we focus on system reliability assessment against low-level technology faults due to effects such as fabrication defects, process variations, radiations, etc. [3].

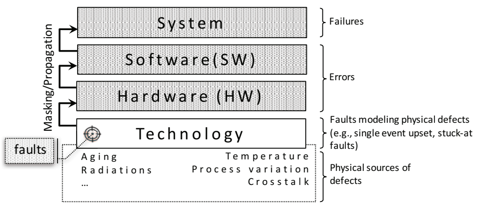


Figure 1: System stack. Faults originate at the lower layer of the system stack and are either masked or propagated to the upper layer possibly resulting in a failure at system level.

Errors resulting from low-level faults may manifest, be masked or be propagated through the HW and SW layers of the system stack, possibly resulting in partial or total failure of the system activities (Figure 1). Other reliability issues such as HW/SW design bugs are out of the scope of this paper.

We propose a reliability assessment model allowing designers to obtain reliability estimations early in the system design cycle. This supports architectural decisions and gives indications about those portions of the system that are critical and deserve customized development effort to improve reliability. Among different modeling styles [48], following the component-based system design approach we propose a component-based reliability model [38]. In component-based reliability modeling, system reliability is estimated using reliability information and other properties (e.g., size, complexity, etc.) of individual system components and their interconnections (the system architecture). Our model exploits Bayesian Networks (BNs) as a statistical foundation for full-system reliability analysis. BNs offer several interesting features for system reliability modeling: (i) efficient calculation scheme, (ii) capability of fitting on field and simulation data, (iii) intuitive and compact representation, (iv) decision support. A BN is a compact representation of a multivariate statistical distribution function encoding the Probability Density Function (PDF) governing a set of random variables by specifying a set of conditional independent statements together with a set of conditional probability functions.

The proposed system reliability model is composed of a *qualitative model* representing the architecture of the system and a *quantitative model,* representing the reliability of each component and their relations.

A. Qualitative model of the system

The system architecture is defined through a *directed acyclic graph* G (Figure 2):

$$\mathrm{G} = \big(V = (C \cup P), E = (RR \cup PR)\big) \tag{1}$$

The set of vertices V is split into two subsets: *components* (C) and *parameters* (P). Components are blocks composing the system. Depending on the architectural layer (technology, HW, SW) the component definition changes as discussed later in this section. Components are associated to Bayesian nodes, i.e., their reliability is associated to a set of random variables. Parameters are special vertices that are not direct part of the system Bayesian model. They represent implementation details of a component (e.g., operating temperature, workload, etc.) exploited by our framework to build the quantitative model of the system described later in this section. The reliability relations $RR = \{(c_i, c_j) \in C \times C\}$ is the set Bayesian arcs that define temporal or physical reliability relations among components, e.g., a failure state of a component may influence the state of another component. Finally, the parameters relations $PR = \{(ip_i, c_j) \in IP \times C\}$ is the set of relations between a component and its implementation parameters. Based on the system stack shown in Figure 1, components of a system are split into four subsets or domains (Figure 2) each requiring different techniques to be characterized for reliability.

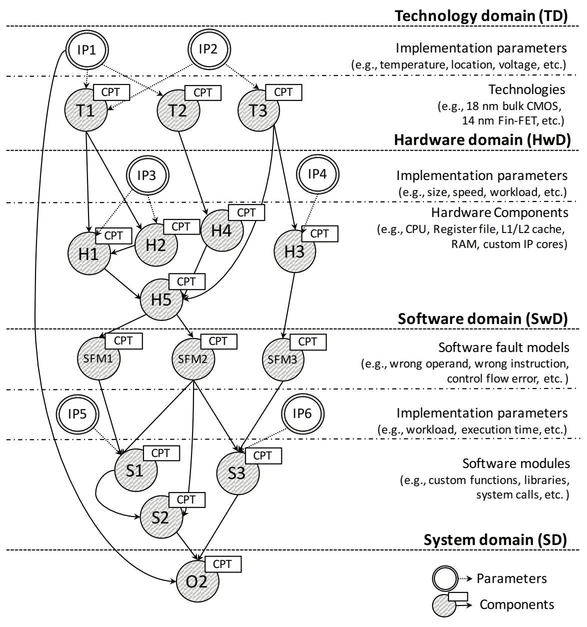

Technology domain (TD)
IP1
IP2
Implementation parameters
(e.g., temperature, location, voltage, etc.)
CPT
T1
T2
T3
Technologies
(e.g., 18 nm bulk CMOS,
14 nm Fin-FET, etc.)
Hardware domain (HwD)
IP3
IP4
Implementation parameters
(e.g., size, speed, workload, etc.)
H1
H2
H4
H3
H5
Hardware Components
(e.g., CPU, Register file, L1/L2 cache,
RAM, custom IP cores)
Software domain (SwD)
SFM1
SFM2
SFM3
Software fault models
(e.g., wrong operand, wrong instruction,
control flow error, etc.)
IP5
IP6
Implementation parameters
(e.g., workload, execution time, etc.)
S1
S3
S2
Software modules
(e.g., custom functions, libraries,
system calls, etc.)
System domain (SD)
O2
Parameters
Components

Figure 2: System reliability estimation model. System components are modeled by *component nodes*. The topology of the network provides the qualitative description of the system. Conditional probability tables (CPTs) associated to each component node of the network provide the quantitative description of the reliability of the system. *Parameter nodes* model information required to compute the CPTs of the component nodes.

The *technology domain* (*TD*) models the physical layer of the system. Components in this domain list all fabrication processes used to build the hardware structures of the system (e.g., 16nm Bulk CMOS for a microprocessor component, 14nm FinFET CMOS for external DRAM, 14nm NAND Flash for external storage, etc.). These components set the raw fault probabilities of the system. Implementation parameters in this domain model physical quantities influencing the raw fault probability of a technology (e.g., temperature, voltage, location, radiation effects, etc.).

The *hardware domain (HwD)* models the hardware architecture. Components in this domain list the hardware blocks such as CPUs, GPUs, memories, accelerators, custom IP cores, used to build the system. Granularity at which hardware blocks are modeled in this domain depends on the level of detail the designer needs for the reliability analysis, and the degree of freedom the designer has with the design of selected components. A complex component such as a microprocessor can either be considered as a whole or split into its subcomponents (e.g., register files, ALUs, buffers, queues, speculation units, etc.) to allow a fine-grained analysis and optimization. Each hardware component is associated to a set of implementation parameters such as size (e.g., number of bits of a memory), speed, workload, etc.

The *software domain* (*SwD*) models the software architecture. To decouple the analysis of the SwD from the one of the HwD, special attention is required to define the interface between the two layers. Components of this domain are further split in two sub-domains: (1) *software fault models* (SFM), and (2) *software modules* (SM).

SFMs are the approach introduced by our model to translate hardware failures into the software domain and therefore to decouple the two domains. In this way, the corresponding supporting tools for the hardware and the software domains can operate independently. SFMs model program alterations that can be linked to alterations of the Instruction Set Architecture (ISA) of the hardware block executing the software. Table 1 lists the set of SFMs currently supported by our framework for selected microprocessor architectures. The table is not intended to be exhaustive. Additional SFMs can be plugged in the model given that proper tools for the evaluation of their occurrence and effect are designed.

Table 1: Example of SFMs taxonomy.

Software Fault Model	Description
Wrong Data in a Operand (WDO)	An operand of the instruction changes its value
Source Operand Forced Switch (SOFS)	An operand is used in place of another
Instruction Replacement (IR)	An instruction is used in place of another
Faulty Instruction (FI)	The instruction is executed incorrectly
Control Flow Error (CFE)	The control flow is not respected

On the other hand, SMs model the software architecture. The granularity of the description in this domain depends on the specific application. A node may represent a high-level library or framework, a single function, a portion of a function or even a specific data structure (e.g., an array), thus allowing for a fine-grained customization of the model. Implementation parameters in this domain mainly include the workload of each module.

Performing system level reliability analysis requires the definition of a set of observation points where the behavior of the system is evaluated and properly classified. In most applications, observation points are a set of software components whose outputs define the outcome of the system. Nevertheless, associating the concept of observation points to the software domain is limiting. To model this concept, the *system domain* (*SD*) has been split from the other domains and placed at the higher-level of our model to separately identify those components where the entire system's reliability is observed.

B. *Quantitative model of the system*

The quantitative model of the system defines the probability of occurrence of an error/fault in a component depending on the condition of its direct interacting components and on its implementation parameters. In a Bayesian model such as the one proposed in this paper, the quantitative model is a set of *Conditional Probability Tables* (CPTs): one CPT $\Theta_{c|\boldsymbol{U}}$ for each node $c \in C$ and its parent nodes $\mathbf{U}$ (Figure 3).

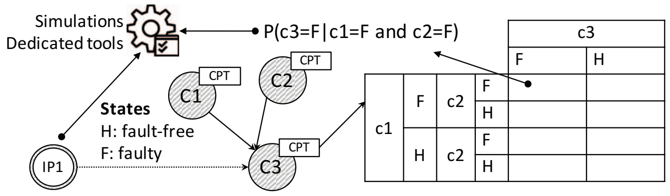


Figure 3: Example of CPT for node c3.

Each node c is associated to a set of states, which identifies possible error or error-free conditions of the node (e.g., a memory can be error free, or it can be affected by a single bit-flip, or by a double bit-flip). The set of states of the nodes depends on the node domain and the specific characteristics of the node. For each state of a node, we need to look at all combinations of states of its parent nodes. Each such combination is called an *instantiation* $\mathbf{u}$ of the parent set $\mathbf{U}$. The CPT $\Theta_{c|\boldsymbol{U}}$ maps each instantiation $c|\mathbf{u}$ to a probability $\theta_{c|\boldsymbol{u}}$ such that $\sum_c \theta_{c|\boldsymbol{u}} = 1$. Nodes without parents, called *root nodes*, are described according to their marginal probability distributions.

Computing CPTs can be both difficult and time consuming. It is typically an assignment given to a group of specialists that need to collect information and organize them according to the model. In this paper, we perform a significant step forward by providing a framework of tools, each designed to operate in one of the four node domains and able to compute conditional probabilities for major classes of software and hardware modules of modern electronic systems. The tools enable to consider each instantiation of a node and to setup a set of simulations to compute the corresponding conditional probabilities in a fast and optimized way. It is important to note that, if a node has many parents or if the parents can take a large number of states, the CPT becomes very large. The size of the CPT is, in fact, exponential in the number of parents. To cope with the exponential number of probabilities in CPT the Noisy-MAX approach can be applied [44]. The Noisy-Max is a generalization of the interaction of a child node and its direct ancestors that allows reducing the size of the computed CPT thus reducing the number of required simulations.

C. *Reasoning on the model of the system*

Once the system is described, the proposed reliability model can be used to reason about the its reliability properties.

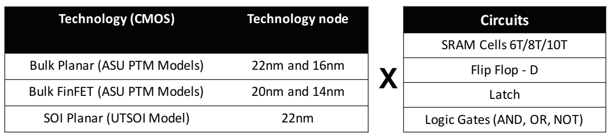
Bayesian reasoning is a well-known approach and the reader may refer to [45] for further details. Two main types of reasoning are supported. In the *predictive reasoning*, starting from information about fault causes (i.e., raw technology failure rates) the designer is able to obtain new beliefs about their effect on the system failures, following the forward directions of the network arcs. This enables early reliability analysis of the complete system. In the *diagnostic reasoning* the designer reasons from symptoms to cause (backward direction of the network arcs), i.e., the observation of a system failure updates the belief about the contribution of components to the failure. This allows us to identify weak components that most likely contributed to the failure, in order to drive the reliability design effort toward the most critical components thus optimizing the overall system at the lower cost. Moreover, the model can be used to calculate new beliefs when new information (evidence) is available. For example, by setting the evidence that a given component is in a given state (e.g., a hardware component is faulty), the model enables to update the belief of the system failure given this new information, as well as to update the belief of the root causes that lead to this component failure.

IV. Integrated Tools Framework

This section describes available tools we developed to populate quantitative information in the proposed system reliability model.

A. Technology Domain

In the technology domain, for different technologies, we developed a tool chain able to characterize the main building blocks composing a logic circuit in order to compute their marginal fault probability with respect to a given failure mode[2]. The current implementation focuses on soft-errors caused by particle strikes. Figure 4 summarizes the technologies and blocks analyzed so far for soft-errors. Further technologies and blocks can be analyzed given the availability of a proper technology and circuit model. Each block and each technology is analyzed for different combinations of run-time parameters. Supported parameters currently include combinations of voltage, temperature and geographical location. The geographical location is considered to accurately predict the error rates caused by particle strikes as detailed.



Technology (CMOS)	Technology node
Bulk Planar (ASU PTM Models)	22nm and 16nm
Bulk FinFET (ASU PTM Models)	20nm and 14nm
SOI Planar (UTSOI Model)	22nm

X

Circuits
SRAM Cells 6T/8T/10T
Flip Flop - D
Latch
Logic Gates (AND, OR, NOT)

Figure 4: Building blocks and technologies analyzed

Figure 5 summarizes the simulation workflow.

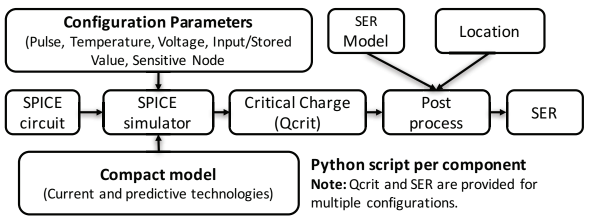

Configuration Parameters
(Pulse, Temperature, Voltage, Input/Stored Value, Sensitive Node
SER Model
Location
SPICE circuit
SPICE simulator
Critical Charge (Qcrit)
Post process
SER
Compact model
(Current and predictive technologies)
Python script per component
Note: Qcrit and SER are provided for multiple configurations.

Figure 5: Technology Domain characterization workflow

A Python simulation engine drives an exhaustive design space characterization for a given component implemented in a given technology. The analysis is organized into a set of nested loops to simulate an element subject to different operating temperatures, voltage, locations and technology models. To study the impact of particle strikes, in the inner loop we iteratively increase the current injected in the sensitive nodes of an element until a flip or glitch is detected measuring the stored value (SRAM) or the output (Logic Gates). Each electrical simulation is performed in HSPICE. The minimum charge generated from a pulse that causes a malfunction is stored and defined as the Q_{crit} of that element. Finally, for each Q_{crit}, a raw soft error rate (SER) is computed using the model in [36].

Through the use of this framework we have been able to build a technology library of SERs for different blocks under different technologies, geographical locations, and voltage/temperature. Data from this library can be fed to our system model any time a new system must be analyzed, without repeating the underlying simulations. The reader may refer to [35] for additional implementation details.

B. Hardware Domain (HwD)

At the hardware domain we focused our effort on the development of a tool-chain able to characterize the different micro-architectural blocks of a microprocessor. Microprocessors are our main target since they are one of the most complex and important blocks of a system. This analysis starts from the assumption that a fault (e.g., a single event upset) affects one of the blocks of the microprocessor. Whether this transforms into an error for other blocks depends on several parameters that are analyzed at this level and in particular on the microprocessor workload (i.e., the executed software).

To compute CPTs for the different structures composing a complex microprocessor we resort to microarchitecture-level fault injection, which delivers very accurate results for array-based structures, unlike ACE analysis that is faster but suffers from over-estimation of the final reliability assessments [9] [13] [37]. We improved an existing microarchitecture level fault injection tool GeFIN [39] based on Gem5 [43] (a cycle-accurate full-system simulator) with an extra operation mode designed to cover the requirements of this work[3]. GeFIN gives us the opportunity to run experiments for two of the major ISAs (ARM and x86). The new GeFIN operation mode developed in this paper is the *IRS (Injection Runs up to the Software level)* mode (for several different modes of operation of GeFIN see also [49]. This mode, enables to run fast injection experiments in which simulations end at the first visible fault effect at software layer; i.e., the moment that the first instruction affected by the fault commits to the architectural state. This isolates the fault in the HwD before it is transferred to the SwD. By running this fast operation mode it is possible to detect how errors propagate among microprocessor hardware structures and how they translate into the SFMs presented in Table 1, thus enabling to compute CPTs for all these nodes. Another operation mode provided by GeFIN is the *IRE (Injection Runs up to the End)* mode. This mode implements a full fault injection campaign running experiments to the end of a benchmark. It allows us to observe the fault effect at the application output. This mode of operation is very time consuming and is only used at the end of our paper to compare the reliability estimations that our model delivers with the reliability estimations of a very detailed fault injection campaign.

Figure 6 summarizes the behavior of the two GeFIN fault injector modes. The pre-fault period represents the interval from the start of the benchmark until the fault injection. This period consumes an average of ~50% of the simulation lifetime. We can further speedup both IRE and IRS modes by skipping the pre-fault period.

[2] The marginal distribution of a random variable is the probability distribution of the variable without reference to the values of the other variables (i.e., opposite to conditional probabilities). Since technology nodes are root nodes they are described by marginal probabilities rather than conditional probabilities.

[3] Any other similar tool can be also used with the appropriate modifications (e.g., MaFIN in [39]).

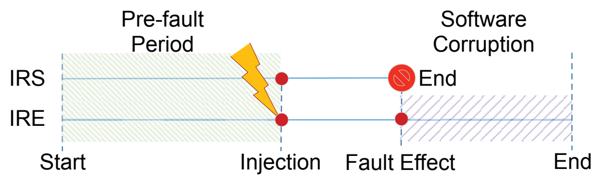


Figure 6: IRE and IRS modes of GeFIN operation.

All array structures of the CPU, which occupy the vast portion of the chip's area and mainly determine the reliability of the entire chip, can be studied through fault injection using GeFIN. For the purposes of this study, we present results targeting five important hardware components: Integer physical register file, Load/Store Queue, L1I cache, L1D cache, L2 unified cache. Finally, to generate the initial fault mask list of all our campaigns at the microarchitecture level we used statistical fault sampling as described in [40]. The number of required fault injection runs per campaign depends on the size of the structure (in bits), the number of execution cycles of the benchmark, the confidence level and the error margin. The error margin and the confidence level mainly affect the total number of the injections. For all the hardware structures and all benchmarks of our study, the number of fault injection runs was rounded up to 2000, which corresponds to 2.88% error margin and 99% confidence level. Increasing the error margin and decreasing the confidence level of the sample, leads to decrease of the execution time of our fault injection campaigns (running less injection runs), decreasing consequently the accuracy of the final estimation results.

C. *Software-level (SwD)*

The SwD models the software architecture by considering SFMs and SMs. SFMs are the error sources for this domain and their CPTs are computed as the final outcome of the HwD analysis using GeFIN. To compute CPTs of SMs we developed a software-level fault injector, which, naturally, is very fast compared to the hardware-level injection tools. To study the SFM effects in the SwD, we need to investigate methods to describe the software independently from the target hardware architecture. This in turns means decoupling this layer from the ISA of the software execution platform. We exploit LLVM (Low Level Virtual Machine), a compiler framework that uses virtualization with virtual instruction sets to abstract the software analysis from the target microprocessor ISA, thus enabling to reuse the same results to characterize systems based on different hardware architectures. LLVM uses the Intermediate Representation (IR) as a form to represent code in the compiler. IR is an intermediate representation between the assembly level and the source code level, and it is independent from the source language and the target machine.

Two relevant LLVM based fault injection frameworks have been proposed LLFI [41] and KULFI [42]. However, both injectors are not designed to work with the high-level SFMs described in our model and focus on simpler fault models such as bit-flips. For this reason, we developed a software-level LLVM based fault injector named LIFILL that targets the SFMs presented in Table 1. Starting from the original target source code, the tool generates the LLVM code that will be used later in the whole process of injection and analysis. Four main steps are performed:

1) *Trace Execution:* The fault free program is first executed to generate an instruction trace, i.e., the list of executed LLVM instructions. Moreover, in this step, the function call graph of the program is generated (including calls to system functions) in order to automatically construct the nodes topology of the analyzed software.

2) *Fault Injection:* SFMs are injected into the LLVM code respecting the input parameters. The injection technique consists in mutating the LLVM code. For each injection a mutated program is generated.

3) *Execution:* the golden LLVM code and each faulty program are executed with their output redirected to a log file. We log all information concerning the LLVM execution (execution time, software outputs, value of each data structure to monitor, etc.).

4) *Analysis:* the log files corresponding to the outputs of the faulty programs are compared to the golden output in order to evaluate the software behavior and determine the CPTs of each node. The considered faulty behaviors are defined in the *Table 2.*

Table 2: Software Faulty Behavior Classifications.

Fault Classification	Description
Masked	The application execution terminates normally. All the application outputs are correct.
Silent Data Corruption (SDC)	The application execution terminates normally. However, the application outputs are different from the fault free outputs.
Unresponsive	The application execution does not terminate normally: it stops working or it never stops.

D. *System domain (SD)*

All tools described in the previous sections are integrated into a system level reliability analyzer that implements the high-level BN model. The tool whose GUI is shown in Figure 7 is written in C++ and QT and provides the following functionalities to the user:

- *System architecture design*: the graph based system architecture can be easily designed. The architecture of complex hardware components (e.g., complex microprocessors) can be selected from a library of components and customized in terms of parameters, whereas the software architecture can be automatically derived from the function call graph provided by the LIFILL tool.
- *Quantitative model:* the output of the different supporting tools can be directly imported in the system analyzer to automatically compile the quantitative model of the system.
- *Reliability analysis:* once a system is described, Bayesian analysis can be performed on the resulting network. Both predictive and diagnostic reasoning is implemented and available to the user.

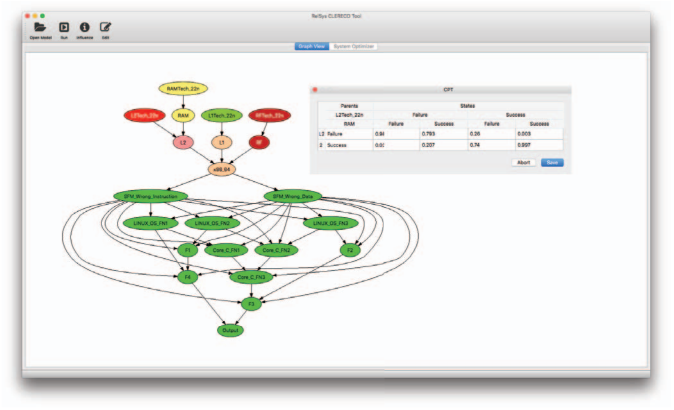


Figure 7: System level reliability analyzer interface.

V. EVALUATION

To demonstrate the capability of the proposed reliability assessment framework we have setup an extensive campaign of experiments (Table 3). We analyzed a set of microprocessor-based systems, denoted hereinafter as *configurations*, each one characterized by: (i) a technology process, (ii) a microprocessor ISA and microarchitecture and (iii) an executed software benchmark.

Table 3: Experimental setup

TECHNOLOGY DOMAIN
T1: 22nm Bulk Planar (FIT: 194.7E-7[*]), **T2**: 22nm Bulk FinFET (FIT: 177.6E-9[*]), **T3**: 22nm SOI Planar (FIT: 111.2E-8[*])
HARDWARE DOMAIN
x86 out-of-order CPU and ARM out-of-order CPU
Register file (256 regs each 64-bits, 2KB) L1 Instruction Cache (32KB) L1 Data Cache (32KB) L2 Cache (1MB) Load/Store Queue (128B)
Main memory
DRAM protected with ECC (i.e., no fault injected)
SOFTWARE DOMAIN
Linux operating system executing one of the following MiBench programs: (1) susan smooth, (2) susan edges, (3) susan corners, (4) qsort, (5) string search, (6) sha, (7) fft
* single bit FIT rate 6T SRAM cells with typical conditions (1V, 50C)

We considered three technology processes taken from our technology library and two microprocessor ISAs: (i) x86 out-of-order model and (ii) ARM Cortex-A15 out-of-order model. This covers two of the main microprocessor families used in commercial products in most computing markets. Each CPU has been split into a set of relevant components as reported in Table 3. We assume the main memory is ECC protected and the analysis focuses on soft-errors (single event upsets) in the microprocessor structures (the model can of course be use with an unprotected main memory). Each system runs the Linux operating system and executes one of the considered applications. All applications have been selected from the MiBench benchmark suite [46]. MiBench benchmarks have been extensively used in reliability studies [37] [39] [9] [47]. In total, considering 7 applications, 2 microprocessor ISAs and 3 technologies we analyzed a total of 42 different configurations.

Figure 8 shows the implemented validation flow. The FIT rate of every configuration (FIT_{BN}) has been computed through our system reliability model. This required to: (i) retrieve SERs for the target technology from our technology library, (ii) analyze the microprocessor structures using the GeFIN fault injector in IRS mode, and (iii) analyze the SwD using LIFILL. We considered three SFMs: WDO, IR and CFE. In parallel, the same analysis has be performed through a full fault injection campaign using the GeFIN fault injector in the IRE mode, which is very accurate but also very time consuming. Results of this campaign have been derated considering the raw SER of the technology and used to compute a fault-injection FIT rate of the system (FIT_{FI}). FIT_{FI} is the reference value to evaluate the accuracy of our model. Finally, we performed an additional *fast analysis* using our reliability model but considering approximated CPTs for nodes in the HwD computed as *average* values over all benchmarks. Populating CPTs of the HwD is the most computational intensive task. By avoiding this task using average values we were able to perform very fast even if not highly accurate analysis (FIT_{FBN}) that can be useful for designers to take very early design decisions. For all fault injection campaigns at all levels (hardware and software) we used statistical fault sampling as described in [40] in order to reach a 2.88% error margin and 99% confidence level for all estimated parameters. Finally, since parameters of the system model are affected by an error margin, Monte Carlo simulation (100,000 samples) is used to analyze the effect of this error on the final estimations.

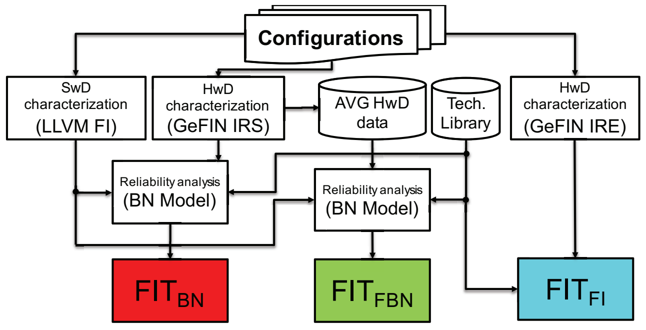
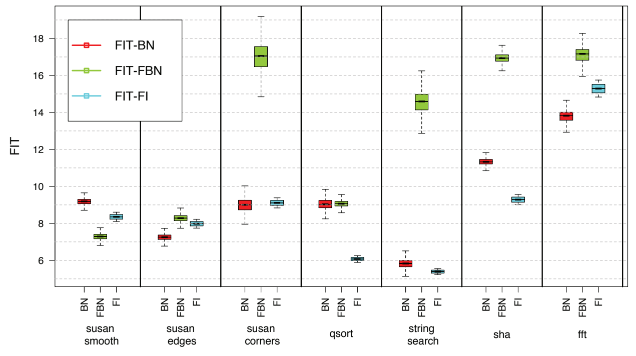
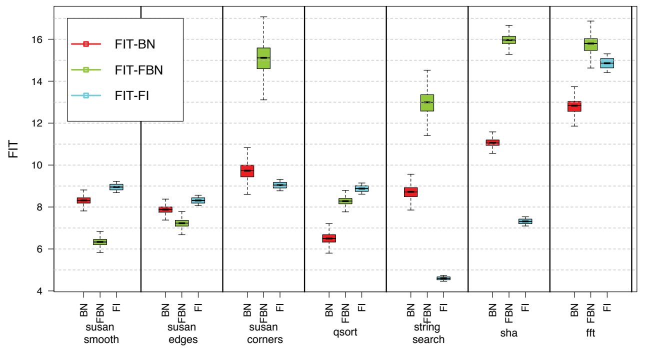

Configurations
SwD characterization (LLVM FI)
HwD characterization (GeFIN IRS)
AVG HwD data
Tech. Library
HwD characterization (GeFIN IRE)
Reliability analysis (BN Model)
Reliability analysis (BN Model)
FIT_BN
FIT_FBN
FIT_FI

Figure 8: Summary of the validation flow used to validate the proposed reliability estimation framework.

Figure 9 shows a set of boxplots reporting for each configuration the three FIT rates reported in Figure 8 for the two ISAs and technology T1. Figure 9 shows the accuracy of the proposed model compared to the fault injection case, which we consider as the baseline. When comparing the FIT_{BN} (red boxes) with the FIT_{FI} (cyan boxes) we can observe for all configurations very small differences in the order of a few units, confirming the accuracy of the proposed framework. Figure 9 shows also interesting results when considering the “fast analysis” (FIT_{FBN}). Interestingly, obtained results even if not accurate as the one provided by FIT_{BN} are still able to give a rough estimation of the reliability of the system, confirming the value of this fast analysis. A bigger library of characterized systems could be a good base for further improvements of this type of analysis.



(A) Technology T1 and x86 microprocessor model



(B) Technology T1 and ARM microprocessor model.

Figure 9: FIT estimation for the 7 selected benchmarks: (1) FIT_{BN} computed with the proposed model, (2) FIT_{FBN} computed with the proposed model with fast analysis, (3) FIT_{FI} computed using full fault injection campaigns.

The analysis of the proposed framework would not be fair without a comparison with ACE-based AVF calculation methods. Unfortunately, available ACE-based AVF reports are related to different benchmarks, different ISAs and different microarchitectures and simulators. George et. al [9] report (among others) the AVF of the integer register file for the closest system configuration to the one we adopt, making it suitable for a fair comparison. The authors employ MiBench benchmarks in an x86 Out-Of-Order microprocessor with a 1MB L2 cache, 64KB L1 data cache, 64KB L1 instruction cache and 256-entries integer register file (same register file size as in our case). The integer register file's AVF computed with ACE analysis is equal to 15% for the string search benchmark.

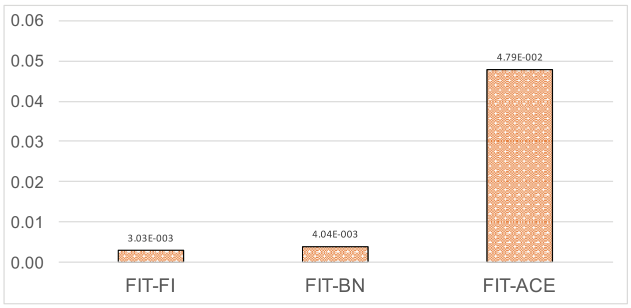


Figure 10: Comparison with AVF computed through ACE analysis for the string search benchmark computed with technology T1.

Figure 10 uses this result and compares it with results from our model assuming that all hardware components are fully protected apart from the integer register file. It is evident from the figure that ACE analysis is much less accurate w.r.t. our model, whose FIT is very close to the one computed by the fault injection. This represents an important step forward in the realization of a framework for cross-layer reliability evaluation.

Apart from the accuracy our experiments also focus on demonstrating the use of our model when performing design exploration. Figure 11 reports the FIT of the susan smooth x86 benchmark estimated for the three considered technologies T1, T2, T3. Since technology related information is recomputed and stored in our technology library, performing this analysis only requires to change the CPT of the technology nodes and to analyze again the BN. This operation is very fast even for very big networks.

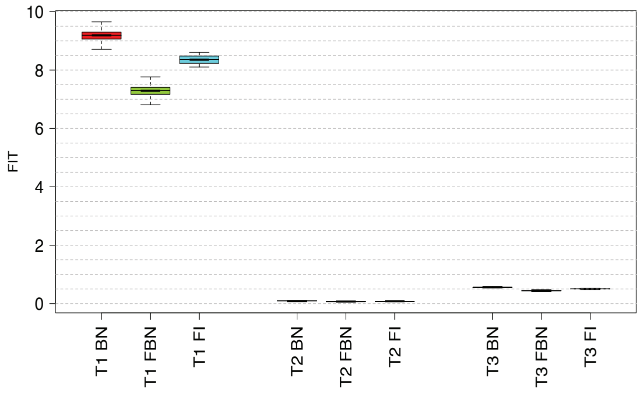
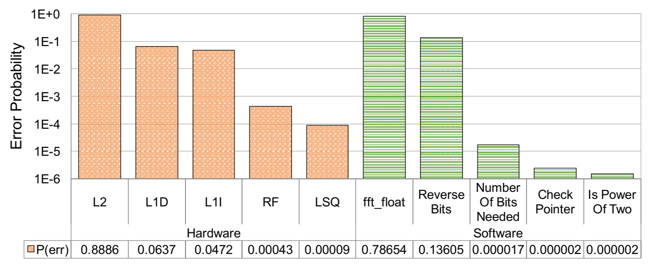


Figure 11: FIT rate of the susan smooth on x86 for 3 technologies (T1: 22nm Bulk Planar, T2 22nm, Bulk FinFET, 22nm SOI Planar)

Figure 12 shows the power of the diagnostic reasoning provided by the framework for the x86 fft configuration with technology T1. By setting the evidence of a failure on all the output nodes of the model we have updated our belief of the causes of this failure. Figure 12 reports the 5 top critical hardware components (orange bars) and the 5 top critical software components (green bars). The reported probability indicates the belief that the component is in an error state given that we observed a failure at the output of the system. These components are preferred targets to improve the reliability of the system.



	L2	L1D	L1I	RF	LSQ	fft_float	Reverse Bits	Number Of Bits Needed	Check Pointer	Is Power Of Two
	Hardware					Software				
▨ P(err)	0.8886	0.0637	0.0472	0.00043	0.00009	0.78654	0.13605	0.000017	0.000002	0.000002

Figure 12: Example of diagnostic analysis for the x86 fft configuration.

Results obtained in Figure 12 have been employed in Figure 13 to perform design exploration. This figure reports the FIT rate computed by our model for the six variants of the x86 fft system reported in Table 4.

Table 4: Design variants for reliability design exploration

Variant	Reverse Bits	fff_float	L2
v1	U	U	U
v2	FT	U	U
v3	U	FT	U
v4	U	U	FT
v5	U	FT	FT
v6	FT	FT	FT

FT: fault tolerant (100% masked faults), U: unreliable

In the different variants, by working on the CPT of some of the critical nodes identified in Figure 12 we emulated the introduction of fault tolerance mechanisms able to fully mask single errors in the node and we evaluated their effect on the FIT rate of the system.

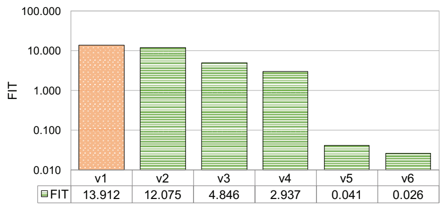


Figure 13: Comparison of the FIT rate for different variants of the x86 fft benchmark with protection mechanisms applied to critical components.

The orange bar refers to the unprotected system. It is interesting to note that acting on individual hardware or software components does not significantly improve the system FIT rate. However, by protecting just two critical components (v5) the FIT rate drops by more than one order of magnitude. Protecting additional components does not produce further significant improvements. This type of analysis can be performed easily by modeling the effect of a protection mechanism at the CPT level and does not require long simulation time. Results such as the one provided in Figure 13 are a valuable instrument to reduce the overhead introduced by reliability oriented design solutions.

To complete our analysis Figure 14 compares the simulation time between the full fault injection, and the BN model on a high-end workstation (Intel(R) Core(TM) i7-4771 CPU @ 3.50GHz, RAM 16 GB) running Linux Slackware.

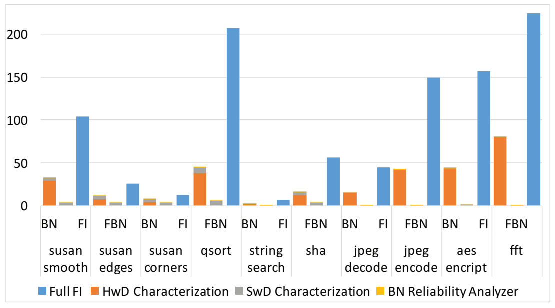


Figure 14: Performance comparisons (hours of simulation).

On average, the BN models of the considered benchmarks are composed of about 20 nodes. For each benchmark we report the full simulation time required to analyze the two CPU architectures. Simulation time to analyze the technology is not included here since data in the technology library are pre-compiled. On average we observe a very significant 65% simulation time reduction for the BN results compared to the FI results, reaching a maximum of ~78% for the qsort configuration. The reader can also observe that, when considering the fast analysis the computation time is negligible.

It is important to remind that, differently from full fault injection, when multiple architectures are analyzed only those components that change from one architecture to another must be characterized more than once. This is the case of our experimental setup where two CPU architectures have been considered. When moving from the x86 CPU to the ARM CPU, the SwD nodes are not influenced and do not need to be characterized twice.

VI. Conclusions

In this paper we proposed a scalable, cross-layer methodology and supporting tools ecosystem for accurate and fast estimations of computing systems reliability based on a component-based Bayesian network model. The model and related tools address all layers of a complex system from the technology up to the application software. The model efficiently calculates the system reliability (FIT rate) driven by the masking probabilities of individual hardware and software components and considering their complex interactions. Through an extensive set of experiments we demonstrated the features and the accuracy of the proposed framework. Besides its accuracy in reliability assessments, one of the key capabilities of the framework is the possibility to perform early diagnostic analysis to identify reliability-critical components of the system and to perform 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.

References

[1] T. Nigam, "Scaling to the final frontier: Reliability challenges in sub 20 nm technologies," in *2011 IEEE International Integrated Reliability Workshop Final Report (IRW)*, 16-20 Oct. 2011.

[2] J.Yoshida. "Toyota Case: Single Bit Flip That killed." *EETimes*, Oct. 2013.

[3] T.Austin, V.Bertacco, S.Mahlke, and Y.Cao. "Reliable Systems on Unreliable Fabrics" *IEEE Design & Test of Computers*, 25(4): 322-333, July 2008.

[4] H.M. Quinn, A. De Hon, and N. Carter. "CCC visioning study: system-level cross-layer cooperation to achieve predictable systems for unpredictable components", *Technical report, Los Alamos National Laboratory (LANL)*, 2011.

[5] S. Mitra, K. Brelsford, and P.N. Sanda. "Cross-layer resilience challenges: Metrics and optimization". In *Conference on Design Automation & Test in Europe* (DATE), 2010.

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

[7] D.W. Coit, J. Tongdan, N. Wattanapongsakorn, "System optimization with component reliability estimation uncertainty: a multi-criteria approach," in Reliability, *IEEE Transactions on Reliability*, vol.53, no.3, pp.369-380, Sept. 2004

[8] S. Distefano, A. Puliafito, "Dependability Evaluation with Dynamic Reliability Block Diagrams and Dynamic Fault Trees," in *IEEE Transactions on Dependable and Secure Computing*, vol.6, no.1, pp.4-17, Jan.-March 2009

[9] N.J. George, C.R. Elks, B.W. Johnson, J. Lach, "Transient fault models and AVF estimation revisited," in *IEEE/IFIP International Conference on Dependable Systems and Networks* (DSN), 2010, pp.477-486, June 28 2010-July 1 2010

[10] N. J. Wang, A. Mahesri, and S. J. Patel. 2007. "Examining ACE analysis reliability estimates using fault-injection". *In Proceedings of the 34th annual international symposium on Computer architecture* (ISCA '07). New York, NY, USA, 460-469

[11] S. Mukherjee, C. Weaver, J. Emer, S.K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," in *Proceedings of International Symposium on Microarchitecture* (MICRO), 2003.

[12] X. Li, S. Adve, P. Bose, and J. Rivers, "SoftArch: An Architecture-Level Tool for Modeling and Analyzing Soft-Errors," in *Proceedings of International Conference on Dependable Systems and Networks* (DSN), 2005.

[13] N. Wang, A. Mahesri, and S.J. Patel, "Examining ACE Analysis Reliability Estimates Using Fault-Injection," in *Proceedings of International Symposium on Computer Architecture* (ISCA), 2007.

[14] N. Soundararajan, A. Parashar, and A. Sivasubramaniam, "Mechanisms for Bounding Vulnerabilities of Processor Structures," in *Proceedings of International Symposium on Computer Architecture* (ISCA), 2007.

[15] K. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic Prediction of Architectural Vulnerability from Microarchitectural State," in *Proceedings of International Symposium on Computer Architecture* (ISCA), 2007

[16] L. Duan, B. Li, and L. Peng, "Versatile Prediction and Fast Estimation of Architectural Vulnerability Factor from Processor Performance Metrics," in *Proceedings of International Symposium on High Performance Computer Architecture* (HPCA), 2009.

[17] A. Biswas, N. Soundararajan, S. Mukherjee, and S. Gurumurthi, "Quantized AVF: A Means of Capturing Vulnerability Variations over Small Windows of Time," in *Proceedings of Workshop on Silicon Errors in Logic-System Effects* (SELSE), 2009.

[18] X. Fu, J. Poe, T. Li, and J. Fortes, "Characterizing Microarchitecture Soft Error Vulnerability Phase Behavior," in *Proceedings of International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), 2006

[19] V. Sridharan and D.R. Kaeli, "Eliminating microarchitectural dependency from Architectural Vulnerability," in *IEEE 15th International Symposium on High Performance Computer Architecture*, 2009, pp. 117-128.

[20] S. Di Carlo, G. Di Natale, and P. Prinetto A. Benso, "Static analysis of seu effects on software applications ," in *Proceedings of the International Test Conference*, 2002, pp. 500-508.

[21] 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

[22] V. Sridharan and D. R. Kaeli. 2010. "Using hardware vulnerability factors to enhance AVF analysis". In *Proceedings of the 37th annual international symposium on Computer architecture* (ISCA '10). ACM, New York, NY, USA, 461-472

[23] A. Savino, S.Di Carlo,, G. Politano, A. Benso, G. Di Natale, A. Bosio "Statistical Reliability Estimation of Microprocessor-Based Systems". In *IEEE Transactions on Computers*, vol. 61 n. 11, (2012), pp. 1521-1534. - ISSN 0018-9340

[24] P. Weber, G. Medina-Oliva, C. Simon, B. Iung, "Overview on Bayesian networks applications for dependability, risk analysis and maintenance areas", *Engineering Applications of Artificial Intelligence*, Volume 25, Issue 4, June 2012, Pages 671-682

[25] R. Roshanak, N. Medvidovic, and L. Golubchik. "A Bayesian model for predicting reliability of software systems at the architectural level." *Software architectures, components, and applications*. Springer Berlin Heidelberg, 2007. 108-126.

[26] S. Yang, M. Lu and L. Ge, "Bayesian Network Based Software Reliability Prediction by Dynamic Simulation," in *IEEE 7th International Conference Software Security and Reliability* (SERE), 2013 pp.13-20, 18-20 June 2013

[27] H. Singh, V. Cortellessa, B. Cukic, E. Gunel, E.; V. Bharadwaj, "A Bayesian approach to reliability prediction and assessment of component based systems," in *Proceedings. 12th International Symposium on Software Reliability Engineering*, 2001. pp.12-21, 27-30 Nov. 2001

[28] H. Okamura, M. Grottke, T. Dohi, K.S. Trivedi, "Variational Bayesian Approach for Interval Estimation of NHPP-Based Software Reliability Models," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2007. DSN '07. pp.698-707, 25-28 June 2007

[29] A.L. Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability," in *IEEE Transactions on Software Engineering* , vol.SE-11, no.12, pp.1411-1423, Dec. 1985

[30] M.R. Choudhury, K. Mohanram, "Accurate and scalable reliability analysis of logic circuits," in *Design, Automation & Test in Europe Conference & Exhibition*, 2007. DATE '07 , vol., no., pp.1-6, 16-20 April 2007

[31] J. Yu, Z. Hehua, S. Xiaoyu, J. Xun and W.N.N Hung, G. Ming, S. Jiaguang, "Bayesian-Network-Based Reliability Analysis of PLC Systems," in IEEE Transactions on Industrial Electronics, vol.60, no.11, pp.5325-5336, Nov. 2013

[32] Y. Wu, Z. Ren, "Mission reliability analysis of multiple-phased systems based on Bayesian network," in *Prognostics and System Health Management Conference* (PHM-2014 Hunan), 2014 , vol., no., pp.504-508, 24-27 Aug. 2014

[33] D. Marquez, M. Neil, N. Fenton, "Improved reliability modeling using Bayesian networks and dynamic discretization", *Reliability Engineering & System Safety*, Volume 95, Issue 4, April 2010, Pages 412-425, ISSN 0951-8320

[34] A. Vallero, A. Savino, S. Tselonis, N. Foutris, M. Kaliorakis, G. Politano, D. Gizopoulos, S. Di Carlo, "Bayesian network early reliability evaluation analysis for both permanent and transient faults," in *IEEE 21st International On-Line Testing Symposium* (IOLTS), 2015, pp.7-12, 6-8 July 2015

[35] M. Riera, R. Canal, J. Abella, A. Gonzalez "A Detailed Methodology to Compute Soft Error Rates in Advanced Technologies**"**, Proceedings of the Design, Automation and Test in Europe (DATE), 2016, Mar. 14-18 2016.

[36] P. Hazucha and C. Svensson. "Impact of cmos technology scaling on the atmospheric neutron soft error rate". *IEEE Transactions on Nuclear Science*, 47(6):2586–2594, Dec 2000.

[37] M. Ebrahimi, N. Sayed, M. Rashvand, and M.B. Tahoori, "Fault Injection Acceleration by Architectural Importance Sampling", in *IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis* (CODES), Oct. 2015

[38] A. Filieri, C. Ghezzi, V. Grassi, R. Mirandola, "Reliability analysis of component-based systems with multiple failure modes". In *Component-Based Software Engineering* (Springer), pp. 11-20, 2010.

[39] M. Kaliorakis, S. Tselonis, A. Chatzidimitriou, and D. Gizopoulos, "Differential Faut Injection on Microarchitectural Simulators", in *IEEE International Symposium on Workload Characterization* (IISWC), Oct. 2015.

[40] R. Leveugle, A. Calvez, P. Maistri, P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence", DATE 2009.

[41] T. Anna, and K. Pattabiraman. "LLFI: An Intermediate Code Level Fault Injector For Soft Computing Applications." *9th Workshop on Silicon Errors in Logic* (SELSE-09). IEEE, 2013

[42] V. C. Sharma, A. Haran, Z. Rakamaric, and G. Gopalakrishnan, "Towards formal approach-es to system resilience," in Proceedings of the *19th IEEE Pacific Rim International Symposium on Dependable Computing* (PRDC), 2013.

[43] 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", in *ACM SIGARCH Computer Arch. News*, vol. 39, no. 2, May 2011.

[44] A. Zagorecki and M. J. Druzdzel (2013) "Knowledge engineering for Bayesian networks: How common are noisy-MAX distributions in practice?" *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* 43 (1). pp. 186- 195.

[45] D. Heckerman, D. Geiger, D.M. Chickering "Learning Bayesian networks: the combination of knowledge and statistical data" *Mach. Learn.*, 20 (3) (1995), pp. 197–243

[46] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, R.B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE International Workshop on Workload Characterization*, 2001. WWC-4. 2001, pp.3-14, 2 Dec. 2001

[47] Z. Zhao, D. Lee, A. Gerstlauer, and L.K.John, "Host compiled reliability modeling for fast estimation of architectural vulnerabilities", in *Workshop on Silicon Errors in Logic System Effects* (SELSE), MarchApril 2015.

[48] D.M. Nicol, W.H. Sanders, K.S. Trivedi, "Model-Based Evaluation: From Dependability to Security", IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 1, Jan.-Mar. 2004, pp. 48-65

[49] A. Chatzidimitriou, D. Gizopoulos "Anatomy of Microarchitecture-Level Reliability Assessment: Throughput and Accuracy", Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, Apr. 17-19, 2016