Boosting Microprocessor Efficiency: Circuit- and Workload-Aware Assessment of Timing Errors

Ioannis Tsiokanos*, George Papadimitriou†, Dimitris Gizopoulos†, and Georgios Karakonstantis*

*Institute of Electronics, Communications and Information Technology, Queen's University Belfast, UK
†Dept. of Informatics and Telecommunications, University of Athens, Greece
*{i.tsiokanos, g.karakonstantis}@qub.ac.uk, †{georgepap, dgizop}@di.uoa.gr

***Abstract*—Aggressive technology scaling and increased static and dynamic variability caused by process, temperature, voltage, and aging effects make nanometer circuits prone to timing errors which threaten system functionality. Accurately evaluating the impact of those circuit-level errors on the resilience of a CPU and the executed applications remains a first-class design issue. However, existing error assessment frameworks fail to accurately model the effects of timing errors because they neglect microarchitecture- and workload-dependent parameters that critically affect the error manifestation and propagation.**

This paper provides a novel, cross-layer framework that addresses the lack of a holistic methodology for the understanding of the full system impact of hardware timing errors as they propagate from the circuit-level through the microarchitecture up to the application software. The proposed microarchitecture-aware tool is able to realistically inject timing errors considering circuit and workload features, accurately assessing timing error effects on any application binary. We estimate the location (bit position and instruction) and the time (cycle) of the injected errors via a workload-aware error model which relies on post place-and-route dynamic timing analysis. We also leverage microarchitectural error injection to access the timing error reliability of a widely deployed pipelined processor under several workloads and voltage reduction levels. To evaluate the proposed tool, our fully automated toolflow is also configured to support timing error injection based on existing workload-agnostic error models. Evaluation results for various workloads and voltage reduction levels, show that our circuit- and workload-aware error injection model improves the accuracy of the error injection ratio by $\sim 250\times$ on average compared to workload-agnostic models. Finally, we quantify the degree to which various applications are prone to timing errors using an application vulnerability metric that can be used early in the design cycle to guide the adoption of energy-efficient error mitigation strategies.

***Index Terms*—Cross-layer timing error evaluation, dynamic timing analysis, error injection, error-resilience, microarchitecture-aware modeling, voltage-scaled FPU**

I. Introduction

The scaling of microelectronics to the nanometer regime enables major design optimizations, but unfortunately circuit performance metrics such as delay, power and leakage exhibit a significant amount of variability [1]–[6]. Those variations caused by process, environmental and operating conditions, may prevent modern designs from meeting their timing specifications. Such a delay/timing uncertainty typically manifests in the form of timing errors which threaten the system functionality and output correctness [6]–[10]. Timing errors occur within a processor and may cause (among others) silent errors - known as silent data corruption (SDC)- in the application's output, without an indication of the output degradation in system event or error logs. Although prior studies attribute SDCs mainly to soft errors due to radiation and particle-strikes [11]–[14], recent studies on real systems/datacenters conducted by Facebook and Google, indicate that computational errors in central processing units (CPU), such as timing errors, lead to SDCs at a much higher rate than the soft error-induced SDCs [15], [16]. Interestingly, such studies reveal that silent errors observed in CPUs, which have minimal error protection, are orders of magnitude higher than errors in memories (DRAM, SRAM) that are equipped with effective error-correction-codes (ECC) [17], [18].

State-of-the-art. Therefore, timing errors are a growing concern for system and application resilience as manufactured devices move towards the atomic scale feature dimensions [19]. In contrast to soft errors, modeling timing errors and estimating their impact on applications early in the design cycle is extremely challenging since timing error manifestation depends on various static (e.g., supply voltage, manufacturing limitations), as well as dynamic (e.g., executed workload, lifetime degradation) parameters [1], [2], [4], [20]. The state-of-the-art in timing error evaluation is the development of high-level error injection (EI) frameworks for injecting errors and assessing their impact on program execution. Those frameworks study timing error effects and guide the implementation of energy-efficient, error-mitigation techniques that aim to depart from conventional pessimistic voltage and frequency guardbands [5], [7], [10], [21]–[23], [23]–[29]. Error injection on models of a microprocessor at any level of abstraction (from the gate and the Register-Transfer Level (RTL) [30], [31] to the microarchitecture [32]–[34] and the software [35], [36]) has been widely used for system reliability assessment. By injecting errors to the accessible resources of each level's model (gates, microarchitectural structures, architectural locations, or instructions), such approaches measure the probability for an error to affect the execution of an application. Although emulation of faulty conditions at the RTL [37]–[40] is the most accurate method in capturing the real impact of errors, it is impractical to use it for full system evaluation (including the user and the system software layers) due to the extremely low simulation throughput and high setup cost, thus it is rarely used [11], [41]. On the other hand, software-level frameworks may help to speed up the functional emulation under injected errors but such campaigns lack most, if not all, circuit and micro-architecture properties that affect the error manifestation and propagation; this can lead to serious assessment errors as it has been recently shown [42].

Error injection at the microarchitecture level employing performance simulators (e.g., gem5) offers a fast and accurate assessment of system behaviour under injected errors at the level of clock cycle [11], [13], [41]–[51] allowing the evaluation of large workloads with the hardware accuracy of the microarchitecture. Such frameworks may have been commonly exploited for evaluating the impact of soft errors, however, they have not been extended to evaluate the effects of timing errors on end-to-end application execution. This can be attributed to the fact that they rely on simple random timing error models that are agnostic of the data, circuit and microarchitecture properties that trigger such errors [12], [52]–[55]. In fact, current established methodologies inject single bit (or multiple bit) flips to the register file based on a fixed (data-agnostic) error probability, which is considered the same for any executed workload. Such data-agnostic models may be effective for emulating particle-strike soft errors [14], [56] which are random in nature, but they cannot capture the circuit-state and dynamic, data-dependent timing error manifestation. Typically, soft errors are shown to be distributed uniformly across bit positions and mainly affect a single instruction and a single bit within this instruction [14], [43], [53], [56], [57]. Conversely, timing errors tend to corrupt multiple bits and instructions depending on the input data [9], [58]–[63]. Failing to understand or underestimating the timing error effects on applications may lead to wrong design decisions and adoption of inefficient error mitigation schemes and operating points.

The manifestation of a dynamic timing error depends on the excitation of timing critical computational paths, which incurs only by specific instruction types when they are fed by certain input operands [9], [59], [62], [64]. To model such data-dependent error behavior, recently, error injection campaigns based on instruction-aware models have been proposed [58], [65], [66]. Although more accurate than data-agnostic error injection models, instruction-aware error injection models focus on injecting and evaluating timing errors rather than estimating the rate at which occurs. Moreover, they lack of detailed microarchitectural modeling (e.g., out-of-order processor models) which may result in wrong conclusions on system's reliability since microarchitecture-level masking and its features (flushes, store forwarding, dead instructions) are neglected.

Contributions. In this paper, we present an accurate and fast timing error injection framework that takes steps towards the evaluation of the impact of timing errors on applications. To achieve this, we jointly consider all different circuit, microarchitectural, and workload factors that affect the manifestation of timing errors and their propagation from the circuit up to the application through a complex pipelined microachitecture. Our main contributions are:

- We develop, for the first time, a fully automated, cross-layer, timing error injection tool that combines and enhances a widely used microarchitecture-level system simulator with detailed circuit-level dynamic timing analysis (DTA) information. Unlike existing timing error evaluation tools, our tool estimates the injection ratio, and considers the time (i.e., instruction cycle) and location (i.e., bit positions) of an injected error exploiting timing properties of the target hardware and executed workload.
- We depart from the conventional, almost purely random fixed error models, and propose accurate and realistic error models that tie the circuit's behavior with the executed input data. In contrast to prior error modeling frameworks which rely on individual functional units, our models are extracted by applying DTA on a complex, post-place-and-route, out-of-order (OoO), 5-stage CPU core in 45 nm process technology. This is the first reported comparison of different timing error models on top of a microarchitectural-based injection framework that injects timing errors considering i) random (data-agnostic), ii) statistical (instruction type-aware) and realistic (instruction type- and input operand-aware) error probabilities/distributions.
- We evaluate our error injection tool by considering benchmarks from different domains: HPC, Data Mining, Medical Imaging and Image Detection. Experimental results provide fine-grained insights of the timing error effects (SDC, Timeout, Crash, Masked) on the application outcome under different levels of delay increase induced by voltage underscaling that is one of the most popular technique to reduce power consumption. It is shown that the error injection ratio and the final program output estimation significantly vary across the comparable models: data-agnostic models inject timing errors within a ratio that fluctuates by $\sim 250\times$ on average compared to the realistic one estimated by workload-aware models using accurate dynamic timing analysis.
- We present a new Application Vulnerability Metric or AVM, which quantifies the degree to which various applications are prone to timing errors under different voltage reduction levels and error injection techniques. We provide conclusive evidence that existing error models are highly inaccurate in timing error assessment, resulting in AVM values that differ by 49.8% on average than the values obtained for the detailed workload-aware model. We also demonstrate that AVM can be used to guide energy-efficient error mitigation schemes, leading to up-to 20% energy savings when combined with a timing error prevention technique.

The rest of the paper is organized as follows. Section II presents background information, and limitations of prior injection frameworks that motivate our work. Section III discusses the implementation steps of our timing error injection tool. Section IV details our evaluation methodology and the considered injection models. Section V presents experimentation results; and Conclusions are drawn in Section VI.

II. Background & Related Work

This section provides background on how timing errors manifest in complex pipelined architectures under any variation-induced delay increase. We also discuss the most common error models that are used by existing timing error injection frameworks and analyse the challenges that motivate our work.

A. Timing Properties in Pipelined Cores

Instruction pipelining is a common technique to improve the execution throughput of a CPU by allowing the simultaneous execution of several instructions [67]. Typically, a pipelined processor consists of a set of N unique timing paths $P = \{P_1, P_2, ..., P_N\}$, which are characterized by their delays $D(P_i)$ for $i = 1, 2, ..., N$ ($D(P_i)$ also considers the clock-to-output delay and the setup time of a register [68]). In such a core, each of the paths can be found within exactly one pipeline stage s, with $s = 1, 2, ..., S$, and only few of them will be excited at every clock cycle depending on the executed instruction. Each pipeline stage processes a specific part of one instruction at a time, allowing the parallel execution of multiple instructions. By the terms parallel or concurrent execution of instructions, we mean that up-to S instructions share the same hardware circuitry (i.e., pipeline) in a time-sharing fashion. At design time, the conventional static timing analysis evaluates the longest timing path across all S pipeline stages and determines the timing bound of the operation, i.e., the clock period (CLK), such as:

$$CLK = \max_{\mathbf{s=1,...,S}} \left\{ \max_{\mathbf{p \in P^s}} \{D(p)\} \right\} = \max_{\mathbf{p \in P}} \{D(p)\} \tag{1}$$

where P^s is the set of unique path-groups in any of the S pipeline stages for $s = 1, 2..., S$ such that $\cup_{s=1}^{S} P^s = P$ and $P^s \cap\ P^{s'} = \emptyset$ for

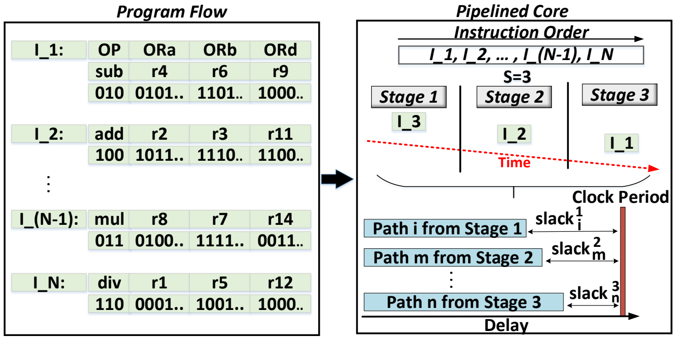


Fig. 1: Instruction sequence and delay requirements across different paths and stages in a 3-stage (S=3) pipelined core.

s $\neq$ s$'$. During the circuit operation only few of these paths get activated depending on the instruction type and its operands. Figure 1 provides an example where 3 instructions (I_1, I_2, I_3) are executed on a pipelined core with $S = 3$ stages. Any excited path P_i has a positive timing slack, $slack_i = CLK - D(P_i)$, until the so-called point of failure (PoF). In case of any path delay increase, which exceeds the available slack $slack_i$, the activated path P_i will fail since $D(P_i) > CLK$, leading to a setup timing error [68], [69]. Timing errors refer to the discrepancy between a computed value and the specified, correct value, and can propagate through microarchitecture to application-layer. As in prior work [58], faults are defined as physical events that corrupt hardware components. If a fault changes the architectural state, then it turns into an error.

B. Data-Agnostic Timing Error Injection Models

Prior frameworks [11], [12], [41], [53], which assess soft-error effects, are data-agnostic since they perform error injection using normal random distributions. To model the data-dependent path excitation, current error injection tools [53]–[55] inject errors with a fixed probability under a specific voltage/frequency setting. We refer to this model as `DA-model` which stands for data-agnostic. `DA-model` estimates the error probability, which we define as P_e, by executing a number of Monte Carlo simulations varying the input data under a given operating condition and an assumed delay increase. At the end, the average error ratio observed across all Monte Carlo trials is used as the fixed probability for *any* injection experiment at the given voltage and delay increase. Most existing tools [11], [41], [53] inject random bit flips into logical or physical registers within the processor using such a fixed error ratio. `DA-model` models the timing error probability $P_e(V)$ as a function of the target supply voltage V. This data-agnostic model is shown to be a good approximation of single-event-upsets [56] and particle strikes [14].

Nevertheless, error injection with fixed probability is also used frequently to model the impact of variations which manifest as timing errors [41], [53]. Unfortunately, this straightforward approach is obviously highly inaccurate and lacks any physical motivation: the model neglects the fact that timing errors incur dynamically only on certain instructions and bit locations once the critical or near-critical paths are excited [59], [69], [70]. Moreover, the timing error injection ratio has no direct link to the activity of the underlying hardware and it is fixed across all type of instructions executed by the target core.

C. Instruction-Aware Models

Modelling dynamic timing errors on a pipelined design is much more complex than simple non-pipelined functional units and requires consideration of the type of the executed instruction, as recently indicated [58], [63], [65], [71]. To better illustrate this, let us assume a pipelined core, as shown in Figure 1. Every pipeline stage consists of a number of timing paths which implement different functions/operations. In such a core, each of the executed instructions activates different timing paths with different delay requirements. The program flow of Figure 1 includes a set of N instructions $I = \{I\_1, .., I\_N\}$ which are concurrently executed in sequences, each consisting of d instructions with $d < N$.

In each clock cycle, both operands and especially the type of instruction that is currently in-flight in pipeline stage s determines the delay of the activated path and thus the number and the distribution of the timing errors. Instructions which activate critical long paths tend to fail more frequently [9], [60], [64], [72]. For example, one of the previous studies [61] shows that the long latency floating-point addition instructions can fail more often than their integer counterparts, which excite less critical paths. In the same direction, authors in [73] observe that timing error-prone instructions are opcode-dependent: special instruction types (e.g., floating-point multiplication) trigger timing-critical paths (i.e., long latency paths), while instructions such as MOV and XOR activate only off-critical paths (i.e., short latency paths).

To model this dynamic timing behavior, instruction-aware error models have been proposed [58], [65], which rely on dynamic timing analysis. We refer to this instruction-aware model as `IA-model`. `IA-model` considerably improves the accuracy of `DA-model` by deploying detailed DTA to extract instruction-aware statistics. This characterization is performed independently for different instructions, using gate-level characterization kernels (covering all the considered instruction types) with randomized input operands. The extracted dynamic statistics are then used to determine the probabilities of an instruction to face a timing error at a specific bit location, using the magnitude of the applied delay increase as a parameter. Unlike `DA-model`, `IA-model` considers instruction and data dependencies of path delays. The timing error probability $P_e(V, I)$ of `IA-model` is determined by the corresponding supply voltage V (which is assumed to be the reason of the delay increase) and the currently executed instruction type. Despite their statistical nature, they also suffer from inaccuracies though in a lesser degree than `DA-model`, because injection is based on an aggregate error ratio rather than the actual circuit state and its executed workload. Moreover, the timing statistics extracted by DTA requires a number of detailed, yet slow gate-level simulations for each instruction type. Thus, due to time constraints, current work performs DTA on a limited number of input operands, which may not be representative and have never validated or tuned with experimental results.

D. Workload-Aware Models

`IA-model` conditions the error statistics on the instruction type, however, depending on input operands, the same type of instruction may activate different paths of different latency requirements leading to significantly different error ratios [59], [60], [64], [74]. The diversity of input operands renders the timing error modeling very difficult. This escalates the need of workload-aware timing error models that are able to characterize the timing behavior of the target design for a given input workload and delay increase [58], [61], [75], [76]. We refer to this workload-aware model as `WA-model`. `WA-model` takes into account *all* parameters: the target supply voltage V, the instruction type I, as well as the executed workload W for estimating the timing error probability $P_e(V, I, W)$. However, those models are applied to simple functional units and thus cannot

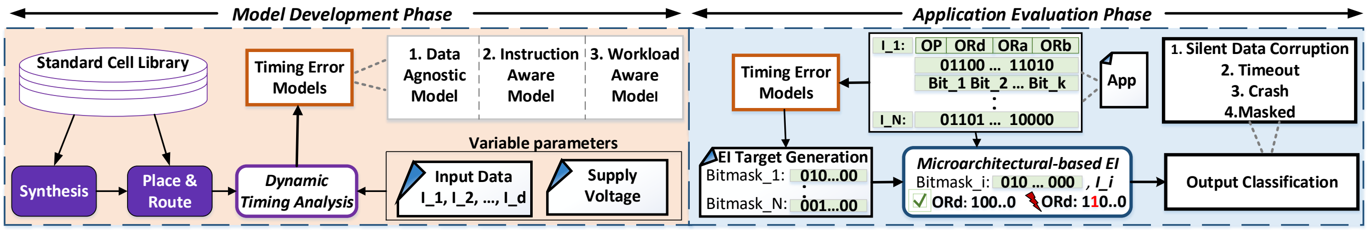


Fig. 2: Workflow for timing error injection (EI) experimentation. We develop the timing error models using circuit-level information, and use those models to perform microarchitectural error injection, evaluating the impact of timing errors on applications' outcome.

capture the complicated path activation of pipelined cores [72], [76] nor evaluate the impact of those errors on applications using detailed microarchitecure-aware timing error injection.

E. Microarchitecture-Aware Error Injection

Timing error evaluation through error injection at either hardware or software layers, is based on injecting errors and experimentally observing their effects on full program execution. Therefore, it is important to model the way that timing errors propagate from the hardware layer and manifest to the software layer. Microarchitecture-based error injection provides a comprehensive way to measure the application's vulnerability due to timing errors on top of the entire system stack, including microarchitecture, architecture, and the software layers (both user and kernel space), during a full program's execution. It reveals useful insights for the application's vulnerability on the presence of timing errors, and thus, ignoring microarhitectural features can misguide resilience studies. Moreover, microarchitecture-based timing error evaluation can be conducted in the pre-silicon stage, using early design models (called performance or microarchitectural models). In such a way, designers can take decisions early about the susceptibility of a specific hardware design on timing errors, and apply the most efficient voltage and frequency settings on their hardware designs.

Table I provides an overview of different error modeling approaches, and their features, as explained in this section. To the best of our knowledge, there is no any timing error evaluation framework to date that relies on circuit-level dynamic timing analysis and accounts for the full-features of the complex microarchitecture of a high-end CPU design.

TABLE I: Overview of existing timing error injection models

Model	Injection technique	Voltage aware	Instruction aware	Workload aware	Microarchitecture aware
DA model	fixed probability	✓	*X*	*X*	*X*
IA model	statistical	✓	✓	*X*	*X*
WA model (proposed)	statistical	✓	✓	✓	✓

III. Proposed Timing Error Injection Tool

To deal with the limitations of the current timing error evaluation frameworks, we propose a cross-layer timing error injection tool for fast, accurate, and realistic timing error assessment early in the design cycle (i.e., prior to silicon prototyping). To this aim, we develop a novel microarchitectural injection framework that injects timing errors based on statistics drawn from accurate, circuit-level dynamic timing analysis. By doing so, our tool is able to incorporate all the dynamic factors (instruction types, operands distribution). These factors determine the manifestation of the timing errors at the outputs (i.e., destination registers or ORd) of functional units as well as their propagation through the underlying microarchitecture to the program output.

Figure 2 depicts the workflow of our tool split in two phases. During the model development phase, we build the timing error models through a comprehensive dynamic timing analysis. During the application evaluation phase, we perform workload- and microarchitecture-aware injections based on the timing error models extracted during the model development phase. The rest of this section describes the two phases in detail.

A. Model Development Phase

The main goal of the model development phase is to generate the timing error models that will be exploited through microarchitectural injection. The generated timing error models determine the spatial and temporal location of an injection (i.e., they characterize the rate and the point of time). The first step of timing error modeling is to estimate the manifestation of timing errors. To achieve this, we perform dynamic timing analysis, which identifies the actual timing margins of the target core at runtime by including path activation information (instruction type, operand values) that are unavailable during static timing analysis but needed for accurate error modeling. To this end, we use detailed post-place-and-route gate-level simulation supported by a commercial hardware simulator (see Section IV.B).

The post-place-and-route gate-level simulation requires the following input parameters:
1) A library file which specifies the logic and the rise and fall times of the standard cells.
2) A gate-level netlist which is stored in a Verilog format (.v). This file consists of a list of the low-level components in the circuit and a list of the nodes they are connected to.
3) A standard delay format (SDF) file which describes the cell and interconnect delays. While the first parameter is provided by the available standard cell library, the other two parameters are obtained using the typical Application-Specific Integrated Circuit (ASIC) flow [77]. It includes the Synthesis and Place and Route steps. Note that those steps are performed utilizing optimizations which aim at achieving maximum performance.

1) Performing Dynamic Timing Analysis: The post-place-and-route gate-level simulation is then used to perform dynamic timing analysis on a number of input data (i.e., a set of instructions and corresponding operands). Note that the size and the type of this instruction set depend on the level of detail of the target error model (see Section IV.C). Every instruction of the target set under nominal conditions produces an error-free output, which is used as the correct, golden output. To estimate the number of manifested dynamic timing errors under any potential delay increase, we execute two parallel instances of post-place-and-route gate-level simulation. The fist instance is executed under nominal operating conditions, while the second simulates the design using lower voltage settings (thus increased delays) than the nominal supply voltage of the evaluated model. Then, the golden output is compared with the simulation output (i.e., output of the reduced voltage) to determine the occurrence of a timing error. A timing error occurs when a simulation run output does not match to the golden output. To speed the comparison, we use a bitwise XOR on each pair of corresponding bits of the two outputs. The result of each bit position is '1' if only the two output bits are different and thus there is a timing error. A value of '0' indicates that there is no error. Therefore, each instruction corresponds to a bitmask which indicates the corrupted bit positions of the destination/output register (ORd).

B. Application Evaluation Phase

At this second phase, the timing error models extracted at the model development phase are employed to perform microarchitecture- and circuit-aware injection, evaluating the impact of timing errors on applications. As shown in Figure 2, the extracted timing error models are provided as inputs to the microarchitecture-level error injection framework (details in Section IV) along with the application. Timing error models provide essential details and information about the error injection ratio, the corrupted bit locations, and the corrupted instructions. Using such information, we can accurately model and evaluate the timing error effects for each application. Each model consists of a set of bitmasks. The number of bitmasks, the location of erroneous bits, and the instructions in which the error occurs (due to timing violations), strongly depend on the application and the voltage conditions. The injector uses this information to model the corrupted instructions of the application and the corresponding corrupted bits of the operation result. In particular, the injector applies the bitmask to the destination register and induces bit-flips depending on the bit values of the bitmask. A bit of '1' in the bitmask implies that an error will be obtained in the specific bit of the considered instruction's destination register. Our injector induces an error by flipping this bit. When the simulation campaign finishes, the results of the simulations are classified into 4 categories, as described in the next section.

For the sake of fair comparison, our toolflow (see Figure 2) is configured to support all the state-of-the-art error models and approaches (i.e., `DA-model, IA-model, WA-model`) explained in Section II. We elaborate on the details of the compared error models later (Section IV). To the best of our knowledge, this is the first reported comparison among microarchitecture-aware, timing error evaluation campaigns that inject errors based on realistic timing error distributions (`WA-model`) which are extracted based on the most accurate pre-silicon way in timing error estimation (circuit-level dynamic timing analysis), and random models (`DA-model`, `WA-model`) that use fixed error statistics.

IV. Setup & Error Models Building

In this section, we first describe the experimental setup of our cross-layer timing error injection tool. Then, we evaluate and compare different error modeling approaches.

A. Software Platform & Benchmarks

We evaluate the impact of timing error on different applications by harnessing the microarchitecture-level accuracy of the most popular performance simulator gem5 [78]. Unlike lower-level simulation models (e.g., gate-level and RTL), gem5 allows cycle accurate and deterministic end-to-end execution of large workloads on top of an operating system, i.e., full system analysis which is impossible at lower levels. We have extended gem5 cycle-accurate simulator to support accurate timing error injections, and employ it to deliver, for first time, a full-system microarchitecture-level timing error injection framework. It consists of a modified gem5 version that allows timing error injection at the microarchitecture level along with instrumentation for running and controlling simulation campaigns on full-system setup. To evaluate the efficacy of our framework, we use a diverse set of 7 benchmarks from the Rodinia [79] and NAS [80] benchmark suites, and an open-source image filter application [81]. Specifically, those programs are *k-means*, *hotspot* and *cg*, *is*, *mg* and *bt* and *sobel*. All applications execute sequentially on a single thread and are compiled with gcc v9.3 using the original building scripts.

TABLE II: Input, size and error classification across the benchmarks.

App	Input	Number of Instructions	Classification Criteria
sobel	123 x 456	632×10^6	Image Output
cg	S	318×10^6	Verification checking
k-means	300_34f	35.5×10^9	Clustering
srad_v1	100 0.5 502 458 1	2.3×10^9	Image Output
hotspot	512 512 1	4.9×10^9	File Output
is	S	73×10^6	Verification checking
mg	S	36×10^6	Verification checking

Table II summarizes the benchmarks, their inputs and what we check in order to classify the injection outcome. Injection outcomes are classified into 4 categories:

- **Masked**. Masked includes the error injection runs in which the timing error does not affect the execution of the application (which is executed till the end) or the system. The result of a simulation with a masked error is identical to the error-free simulation in terms of the output of the application.
- **Silent Data Corruption (SDC)**. The simulation finished normally, but the program output was different compared to the error-free simulation, without any observable indications of this effect.
- **Crash**. A simulation that did not reach the end of the execution, as it was disturbed by an unrecoverable event. As a result, no program output was produced. A crash may refer to a process crash (killed process), a system crash (kernel panic), or a floating-point exception.
- **Timeout**. The simulation did not finish within a certain amount of time, equal to two times the error-free execution time. These simulations are externally stopped to resolve potential deadlock or livelock situations.

B. Hardware Platforms

To extract error models, we perform timing analysis of the open-source, 32-bit *marocchino* general-purpose embedded microprocessor which is based on OpenRISC instruction set architecture (ISA) [82]. The microarchitecture of the core includes a 5-stage, out-of-order-pipeline and supports a single and double precision, IEEE-754 compatible [83], floating-point unit (FPU). Figure 4 shows the

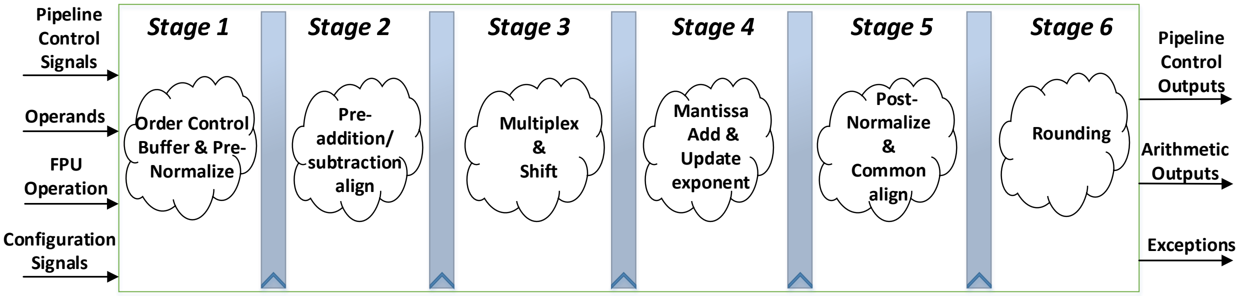


Fig. 3: Microarchitecture of the floating-point addition/subtraction operations.

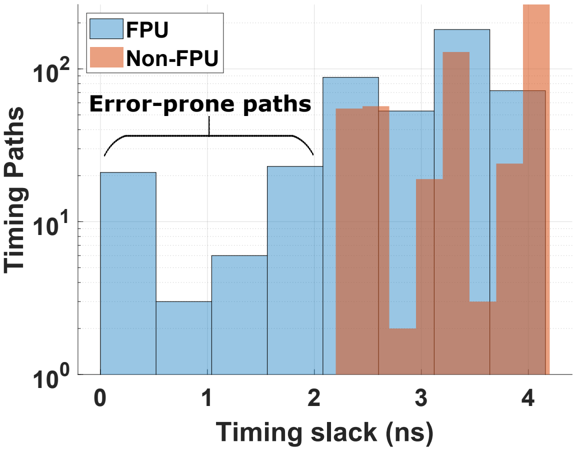


Fig. 4: Distribution of the 1000 longest timing paths (i.e., paths with the lowest slack) in the marocchino pipeline.

distribution of the 1000 longest (i.e., most prone to timing errors) paths of the post-placed-and-routed core. The figure shows that in this implementation only the FPU-related paths are prone to timing errors (they have the lowest timing slack), while all the paths in non-FPU instructions (e.g., integer addition, MOV, etc.) are short enough (they have sufficient timing slack) to be safe under potential levels of delay increase due to voltage scaling. We further assume that memory is protected with Error Correcting Codes [17].

Hence, for this study, we can limit our modeling to arithmetic floating-point instructions, following the representation: $-1^S \times M \times 2^E$, where S denotes the sign, E is the exponent and M is the mantissa. In a double (single) precision floating-point number, defined by the IEEE-754 Standard, the most significant bit (MSB) indicates the sign, the next 11 (8) bits represent the exponent and the mantissa consists of the last 52 (23) bits. Previous studies have also demonstrated that floating-point instructions are much more prone to timing errors due to their long computation timing paths [58], [60], [61], [64], [76]. Also, floating-point operations typically determine the clock period (they have the lowest timing slack as shown in Figure 4) and emerge as a major contributor to the energy consumption (>30%) [84], [85] Note that the ARM FPU and CPU model on which we perform our microarchitecture-level experiments on gem5 have an 1-to-1 correspondence of instructions to the OpenRISC FPU used by model development phase (see Figure 2) to estimate timing errors. Although we demonstrate the features and accuracy of our cross-layer error injection framework on the floating point subsystem of the CPU, *exactly the same approach* can be employed to evaluate the impact of timing errors in other subsystems of the CPU.

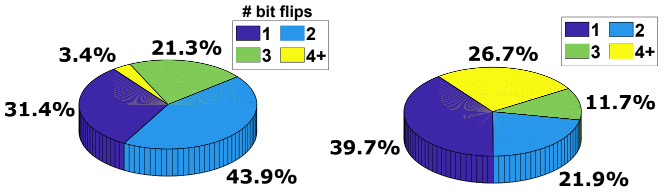
For the experiments in this study, the following 12 floating-point instructions (6 single precision instructions + 6 double precision instructions) are implemented: multiplication, division, addition, subtraction, integer to floating-point and floating-point to integer conversions. Figure 3 highlights the microarchitecture of the floating-point addition/subtraction operations. At Stage 1, an Order Control Buffer (OCB) and a Pre-Normalize block are implemented, which permit data dependencies detection and adjustment of the exponent and mantissa, respectively. Stage 2 is responsible for the pre-addition/subtraction alignment, while Stage 3 performs the necessary multiplexing and shifting of the operands. Mantissa addition and exponent update are performed at Stage 4; post-normalization and rounding occur in the last two stages. The target FPU also handles exceptions such as denormalized numbers, overflow, underflow, or Not-a-Number (NaN) and generates exception signals.

1) Gate-level Operating Conditions: The FPU design is implemented using the typical corner of the Composite Current Source (CCS) NanGate 45 nm library (supply voltage = 1.1V, temperature = 25°C) [86]. For hardware Synthesis and Place and Route, we use the Design Compiler (version: N-2017.09-SP3) from Synopsys and Innovus (version: v16.13-s0451) from Cadence, respectively. For the dynamic power measurements, we invoke Voltus (version 16.2) from Cadence. DTA is performed using detailed post place-and-route simulation supported by ModelSim (version 10.7c) from Mentor Graphics. The fastest (minimum) clock period or *CLK* achieved is 4.5 ns. In this work, we focus on timing errors resulting from different voltage reduction (VR) levels. We use two different VR levels, VR15 and VR20 that correspond to 15% and 20% supply voltage reduction, respectively. Particularly, the nominal operating voltage of target design is at 1.1 V which is defined by the typical corner of the available library. To evaluate the performance of the design across the two VR levels, we perform accurate library characterization under iso-temperature and process conditions. To do so, we use SiliconSmart from Synopsys. Although our evaluation focuses on timing errors as a result of supply voltage reduction, our injection tool can be used to evaluate timing errors due to different sources of delay increase (e.g., overclocking, temperature fluctuations, transistor aging, process variation).

C. Timing Error Injection Models Comparison

To analyze the impact of different error models on timing error evaluation, we perform microarchitectural error injection using the following state-of-the-art models discussed in Section II.

1) Timing Error Injection using a Data-Agnostic Model: We begin our analysis with the widely used random error injection which is agnostic of the executed data [12], [14], [41], [45], [46]. `DA-model`



(a) VR15 V

(b) VR20 V

Fig. 5: Distribution of the number of bit flips at faulty instructions outputs under 15% (VR15) and 20% (VR20) supply voltage reduction (VR).

introduces random bit flips into all or a limited subset of registers within the processor using a fixed error injection ratio. To estimate this, we perform DTA on 10 millions instructions which are randomly extracted from the considered benchmarks. We estimate the timing error ratio (ER) of the target instructions when subjected to 15% (VR15) and 20% (VR20) voltage reduction, respectively. We define ER as follows:

$$ER = \frac{Faulty\ Instructions}{Total\ Instructions} \tag{2}$$

ER under VR15 is equal to 10^{-3} and under VR20 it is 10^{-2}. Hence, based on DA-model, the number of the injected timing errors to each program is predefined, it depends only on its execution time and can be estimated as follows: $\#errors = \lceil \#Instructions \times fixed\ ER \rceil$.

DA-model randomly selects the target instruction and each bit flip occurs with a uniform, fixed probability. In particular, this model corrupts a single bit of the target instruction and the corrupted bit is randomly selected across the bits of the destination register. This simple model has originally been motivated by the analysis of single-event-upsets, such as soft errors, which affect all resources uniformly and independently of the processor state or timing properties. However, timing errors tend to flip more that one bits [58], [61], [64] and in a non-uniform pattern. Using dynamic timing analysis (see Section III.A,1), we characterize how many bits of the faulty instructions are flipped and at which ratio. Figure 5 shows that timing errors flip multiple bits in most cases (64.5% on average under the two VR levels). Such a finding is consistent with existing studies indicating that timing errors tend to affect more than 2 bits [58].

2) Timing Error Injection using an Instruction-Aware Model: To improve the inaccuracy of DA-model and more accurately link the microarchitecture-level error injection to the physical circuit behavior in the presence of timing errors, IA-model has been proposed which performs error injection using instruction-aware dynamic statistics. We have adopted this model from existing error injection tools [65], [66] which extract bit error ratios by applying accurate yet slow dynamic timing analysis. Due to the very low throughput of circuit/gate-level timing analysis, such a characterization can be only based on a limited number of input operands the number of which may not be statistically significant. This may lead to inaccurate error statistics, misleading timing error behavior of circuits. To investigate how many instructions are more important that others in estimating accurate bit error ratios (BERs) (i.e., BERs close to the actual ones), we conduct the following experiment. First, we extract the input operands of all the floating-point multiplication (fp-mul) instructions found in is program and estimate the ER of each bit (i.e., BER) at VR20. The BER of the full trace is then compared with the one measured by a set of $K : K = 10K, 100K, 1M$ randomly extracted fp-mul instructions.

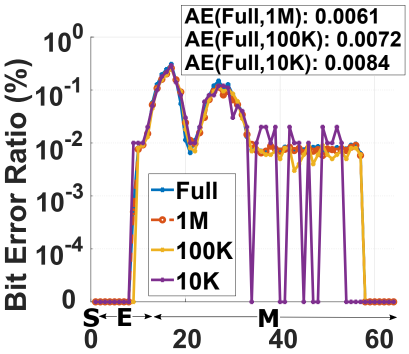


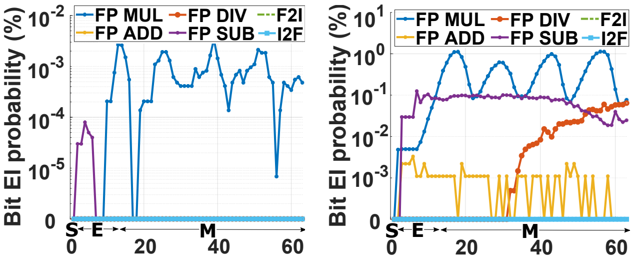
Fig. 6: Bit error ratio (BER) for the floating point multiplication (fp-mul) instruction of the is program under different number of instructions and 20% voltage reduction. X axis depicts the sign (S), exponent(E) and mantissa (M) bits. The average absolute error (AE) across the BER extracted by the different number of fp-mul instructions and the BER extracted by the full trace, is also depicted.

Each step records the BER and the average absolute error (AE), defined as:

$$AE = \frac{\sum_{i=1}^{K} \left| \frac{BER_{full}(i)-BER_{sim}(i)}{BER_{full}(i)} \right|}{K} \tag{3}$$

where BER_{full}(i) denotes the exact BER output value obtained from the full trace and BER_{sim}(i) represents the BER obtained by the simulation using a set of K instructions for a specific output bit position. For this experiment, we use input operands with a 32-bit value range (single precision floating-point instruction) and a 32-bit output result, and with 64-bit operands (double precision floating-point instructions) giving a 64-bit result.

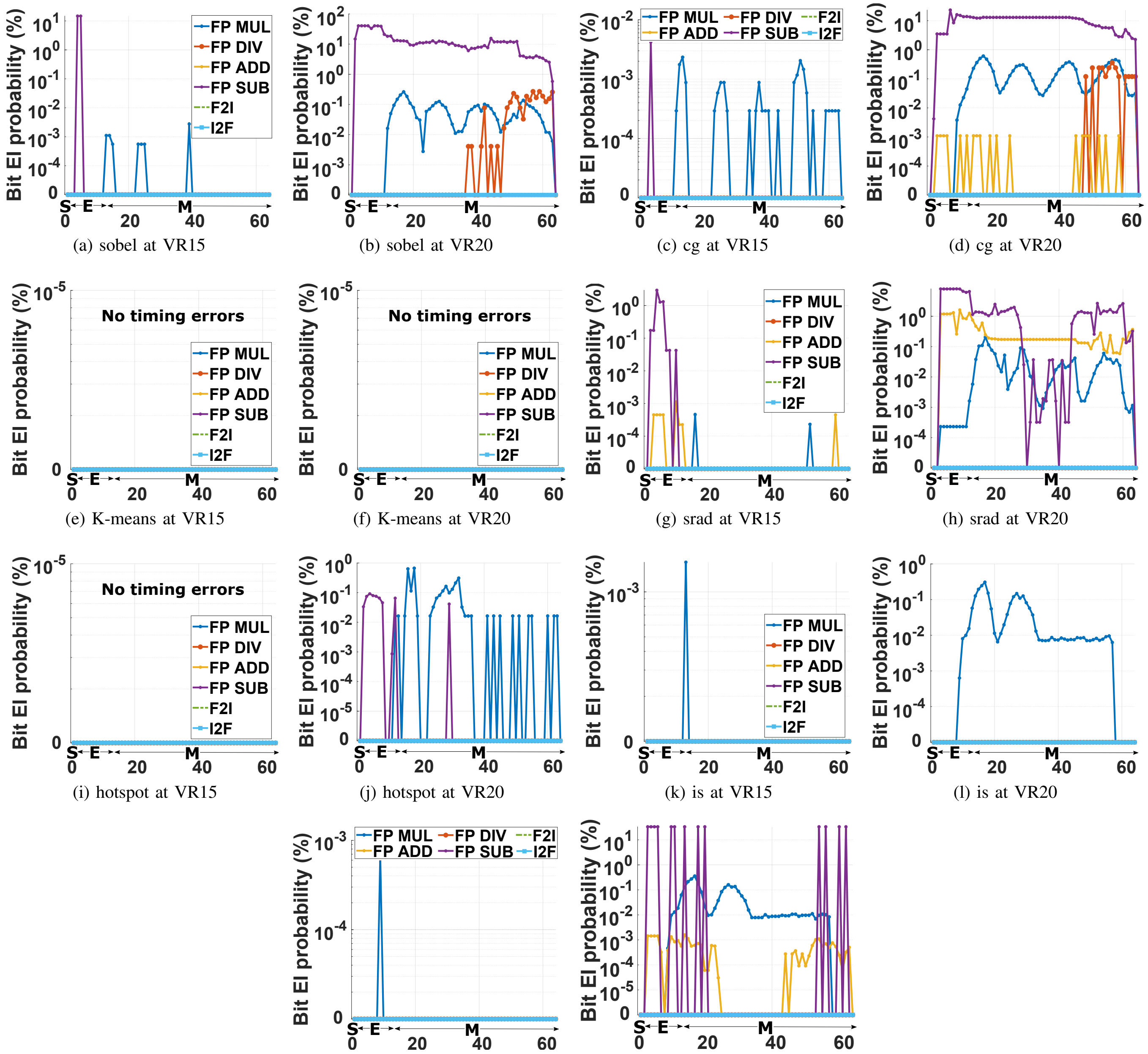
Figure 6 shows the output BER and AE in the sign, mantissa and exponent bits across different number of considered instructions. As observed, the BER produced by both 1M randomly extracted fp-mul instructions and full trace is almost identical with low AE. Similar observations were made when we perform dynamic timing analysis using different instruction types, however due to space limitations, we plot insights only from fp-mul instructions which as we explain next are the most error-prone ones. Thus, we choose to simulate $1M$ input data of each instruction type to extract instruction-aware statistics for the IA-model. Specifically, we use in total $12M$ uniformly distributed random operands, covering all the considered 12 floating-point instructions ($1M$ operands per operation). Unlike



(a) VR15

(b) VR20

Fig. 7: Bit error injection (EI) probabilities for an instruction-aware timing error injection model under (a) 15% voltage reduction (VR15) and (b) 20% voltage reduction (VR20). X axis depicts the sign (S), exponent (E) and mantissa (M) bits.



(a) sobel at VR15 (b) sobel at VR20 (c) cg at VR15 (d) cg at VR20

(e) K-means at VR15 (f) K-means at VR20 (g) srad at VR15 (h) srad at VR20

(i) hotspot at VR15 (j) hotspot at VR20 (k) is at VR15 (l) is at VR20

(m) mg at VR15 (n) mg at VR20

Fig. 8: Bit error injection (EI) probabilities in the considered benchmarks under the instruction- and workload-aware timing error model ($WA-model$). All these outcomes are evaluated under 15% (VR15) and 20% (VR20) supply voltage reduction. X axis depicts the sign (S), exponent (E) and mantissa (M) bits.

the `DA-model`, the `IA-model` models instruction and data dependencies of path delays.

Figure 7 depicts the bit error injection probabilities extracted for each instruction type and VR level for the `IA-model.` As shown, in the target design, fp-MUL is the most error-prone instruction. Particularly, for VR15, fp-mul and floating-point subtraction (fp-sub) are the only two instructions that may lead to a timing error. The integer to floating-point (I2F) and floating-point to integer (F2I) conversions incur no errors under any of the assumed VR level, while floating-point divisions (fp-div) and additions (fp-add) result in errors only under VR20. It is important to note that we observe no timing errors for the single precision floating-point instructions under the target operating conditions (reduced supply voltage).

3) Timing Error Injection using an Instruction- & Workload-Aware Model: The `IA-model` extracts different statistics depending on the type of the executed instruction. Then, it injects errors based on those statistics into every input workload. Unfortunately, despite the significant effort of `IA-model`, timing errors cannot be accurately modeled due largely to lack of consideration of workload variation (input operand values). It is recently shown that both the instruction type and mainly the input values can significantly influence the distribution and ratio of the dynamic timing errors [9], [58], [61], [75]. Therefore, we develop a more accurate timing model, namely $WA-model$ that considers instruction type and operands distribution

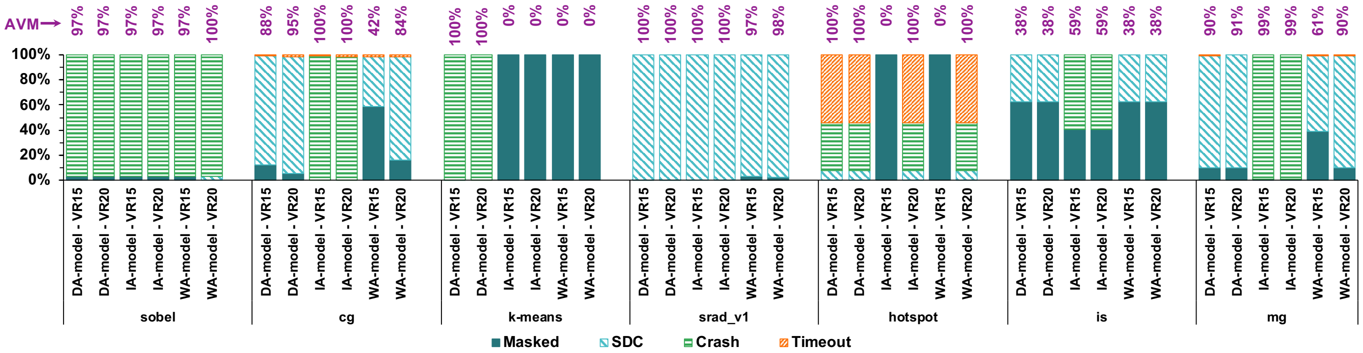


Fig. 9: Injection outcome distributions in the considered benchmarks under data-agnostic model ($DA-model$), instruction-aware model ($IA-model$), and instruction- and workload-aware model. All these outcomes are evaluated under 15% (VR15) and 20% (VR20) voltage reduction.

of the target benchmark. $WA-model$ estimates the bit error ratio of each benchmark by applying dynamic timing analysis on $1M$ randomly extracted instructions from the executed workload. Note that none of the prior workload-aware models have been used in the context of microarchitectural timing error injection that takes into account data- and microarchitecture-dependent error manifestation and propagation in the full system stack.

Figure 8 depicts the bit error ratio across the floating-point instruction types in the 7 benchmarks at VR15 and VR20 voltage reduction levels, where several interesting observations can be made. Figure 8 illustrates the error behaviour of each of the 64 individual output bits (sign, mantissa and exponent bits) of the floating-point unit (FPU) under test; it clearly shows that different workloads exhibit vastly different bit error ratios. For example, the ten MSBs (i.e., first 10 bits from the left) of the `mg` under 15% voltage reduction (see Figure 8(m)) exhibit a nearly zero BER, while the same bit positions under the `srad` program (see Figure 8(g)) exhibit BER up-to 8%. Furthermore, it is evident that, in contrast to soft errors, each bit has its own timing error ratio and, thus, multiple bits can have timing errors simultaneously. This justifies the importance of using different error injection probabilities for each bit. It can be also observed that the mantissa's bits are more prone to timing errors than the exponent's bits. This is an interesting finding as it indicates that techniques tailored for error mitigation on the mantissa's bits could improve fault tolerance of applications.

V. Experimental Results

This section demonstrates the effectiveness of our cross-layer tool in accurate timing error evaluation. We first showcase the application outcomes obtained using error injection for different voltage reduction levels to visually contrast the results from different error models. Next, we report the error ratio along with the application vulnerability to errors across different voltage reduction, or VR, levels and error injection models.

For the statistical significance (error injection accuracy for a 3% error margin with a 95% confidence interval) of the evaluation results, we run 1068 different executions [87] for each benchmark and at each VR level. The plots related to `DA-model, IA-model` and `WA-model` reflect this setting. For the 7 target programs, the 2 VR levels, and the 3 error models of our comparison, we conducted a total of $(1068 \cdot 7 \cdot 2) \cdot 3 = 44856$ injection experiments. For every program execution, we apply the bitmasks in a random clock cycle. Note that the instructions that are corrupted during an experiment are guided through the timing error models provided by the Development Phase (see Figure 2).

A. Distribution of Error Injection Outcomes

Here, we discuss the outcomes of the injection experiments at the application-level, and report them in terms of Crashes, SDCs, Timeouts and Masked outcomes, for the different error injection models and VR levels.

Figure 9 shows the distributions of the error injection outcomes in the considered benchmarks under the different timing error models and VR levels. The results are obtained using the developed toolflow (Figure 2) and configuring it to inject errors based on the 2 traditional errors models (`DA-model, IA-model`) and the proposed `WA-model`. Then, we compare the output of the target application applying different error models with the error-free output. Note that the injection outcome distributions are largely application-dependent. As we see, in most of the cases the `WA-model` provides significantly different results that the `DA-model` and `IA-model`. As we have demonstrated in Section II, the `DA-model` and `IA-model` models are applied to simple functional units and cannot capture the complicated path activation of pipelined cores, and thus, cannot be considered as a representative way for accurately evaluating the impact of timing errors due to low-voltage operation on different applications. Assume for example the *cg* benchmark shown in Figure 9. Evaluating the *cg* application for timing errors for two different voltage levels (VR15 and VR20) based on the `DA-model`, we can see that there is a high probability of SDCs to occur as the effect of the timing errors in these voltage levels. Specifically, there is only 12% and 5% masking probability for VR15 and VR20, respectively. `IA-model`, on the other hand, show that the masking probability is 0% and the application primarily suffers from Crashes (instead of SDC which is the primary cause of the timing error effects according to `DA-model`). Since both models (i.e., `DA-model, IA-model`) cannot capture all required information for evaluating the effects of timing errors, these divergences are expected. On the contrary, the proposed `WA-model`, which takes into account all necessary aspects from the circuit-level analysis, is able to provide accurate evaluations of the timing error effects for each application. The large magnitude of the masking probability of *cg* is obvious in the two voltage levels (i.e., VR15 and VR20). Specifically, while `WA-model` shows 58% and 16% masking effects for VR15 and VR20, respectively, the `DA-model` and `IA-model` show virtually zero (or negligible) masking probability. It is clear that such diverging results that the `DA-model` and `IA-model`

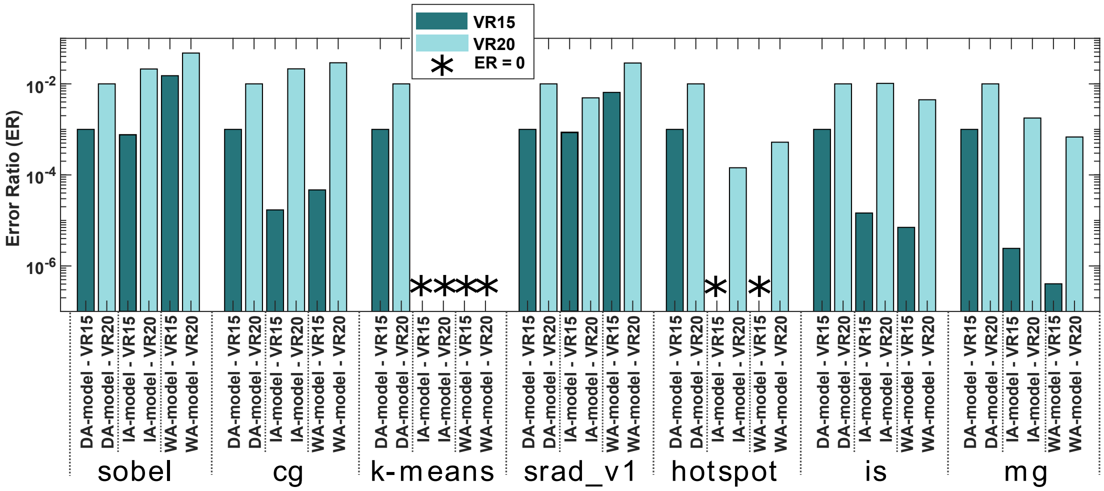


Fig. 10: Timing error ratio (ER) in the considered benchmarks under data-agnostic error injection ($DA-model$), instruction-aware error injection ($IA-model$), and instruction- and workload-aware error injection. All outcomes are evaluated under 15% (VR15) and 20% (VR20) voltage reduction.

deliver, can easily *mislead any protection or energy-efficient decisions for a specific design*. Note that for our target design, VR15 and VR20 voltage levels are intentionally chosen because they provide aggressive voltage reduction (and thus, power reduction), so we can show the significance of the divergences provided by `DA-model` and `IA-model` compared to the `WA-model`. Similar observations can be also drawn for *is* and *mg* applications.

Another important observation derived by Figure 9 is that there are also cases in which a specific voltage level will not produce any errors for some applications. Assume for example the *hotspot* application, in which we can see that in VR15 there is zero probability of timing errors to occur (i.e., `WA-model`). This means that there is a potential for more aggressive power savings. However, the `DA-model`'s evaluation results hide this opportunity, and very simplistically show that this application has 100% probability to be corrupted in this voltage level; a misleading result. The same observation also exists for the *k-means* benchmark, which can safely operate at both VR15 and VR20 according the the `WA-model`, while `DA-model` shows the exact opposite result. These findings indicate that `DA-model` and `IA-model` are inaccurate in evaluating dynamic timing errors since both models lead to pessimistic results concerning the estimation of timing error effects, and thus, prevent the system for being more energy efficient.

B. Timing Error Injection Ratios

A more informative indicator of the divergence or the similarity among measurements with the different error models is the ratio of injected errors. Figure 10 compares the ratio of errors injected by our microarchitectural error injection tool with the ratios obtained for `DA-model`, `IA-model` and `WA-model` under different VR levels. For each experiment, we estimate the timing error ratio (see (2)). There are several interesting observations in Figure 10. First, as expected, all the compared error injection models inject more errors under VR20 than VR15. This difference in the number of errors between the two VR levels is consistent with the timing wall phenomenon [64], [70], [88]. Second, different applications experience different error ratios. This is because different input data activate different paths. However, this is not true for the instruction- and operand-agnostic `DA-model` which assumes a fixed error ratio across the considered workloads. `DA-model` injects errors with a ratio that differs (higher or lower) by $\sim 250\times$ on average from the ratio obtained when applying the accurate error model, i.e. `WA-model`; this is a clear indication of the accuracy the `WA-model` brings.

To make error injection more realistic, `IA-model` extracts instruction-aware error probabilities for each distinct bit. `IA-model` injects errors with a ratio that varies by $\sim 230\times$ on average from the ratio obtained when we apply error injection based on $WA-model$.

C. Analysis of Application Vulnerability

The proposed error injection framework enable us to early check if a particular application is susceptible to timing errors when the target CPU operates at reduced voltages. For this reason, we introduce the *Application Vulnerability Metric* (AVM), which can measure the probability of different voltage reduction levels to affect the application's correct execution. AVM is defined as follows:

$$AVM = \frac{\#SDC\ +\ \#Crash\ +\ \#Timeout}{Total\ Injected\ Errors} \qquad (4)$$

AVM takes into account all non-masked probabilities (i.e., SDC, Crash, and Timeout) and aggregates in a single number the severity of each voltage level and each application to timing errors. For example, as shown at the top of Figure 9, in the case of k-means, AVM for `IA-model` and `WA-model` indicates that this program is highly tolerant to errors (AVM = 0%) under 15% and 20% voltage reduction. Accordingly, we can reduce the voltage from 1.1 V down to 0.88 V without influencing the program output and thus improve power efficiency by up to 56%. Notably, for the same workload, the `DA-model`-driven error injection suggests no voltage reduction beyond 10% of the nominal voltage, which is translated to 21% of power savings. It is clear how severe the effect of pessimistic voltage reduction is due to the inaccurate evaluations of `DA-model` and `IA-model`.

VI. Conclusion & Future Work

We presented a novel, cross-layer framework to accurately evaluate timing error effects for any application binary prior to silicon prototyping and thus guide optimal points of operation. To achieve this, we consider, for the first time, the workload- and microarchitecture-dependent error manifestation and propagation in any program. Our fully-automated toolflow allows the designer to assess the impact of timing errors on programs, with realistic input sizes consisting of billions of instructions, and identify reliability bottlenecks in microarchitectures (by pinpointing hardware structures that cause frequent errors), while determining efficient operating settings under a desired output quality target. Our tool can be also used for post-silicon analysis. It helps application/infrastructure developers to i) detect code regions that are vulnerable to timing errors due to the existence of error-prone instructions, and ii) select efficient error recovery schemes.

The proposed framework injects timing errors at runtime to the actual program execution by leveraging and enhancing a widely-used microarchitecture-level system simulator, gem5. We estimate the injected timing error considering workload and microarchitecture properties that significantly affect how timing errors manifest and propagate. For fair comparison and evaluation of the proposed workload-aware model which relies on detailed circuit-level gate-level simulation, our toolflow allows the evaluation of timing errors which are injected via different error models which are data-agnostic. When compared to existing error injection models, our workload-aware model enhances the accuracy of error injection ratio by $\sim 250\times$ on average in different applications under various degrees of voltage reduction. Additionally, our framework is able to quantify the degree to which several applications are prone to timing errors and guide energy-efficient operating settings by introducing the new Application Vulnerability Metric, AVM.

In our future research, we aim to characterize the behavior of several CPU models in the presence of timing errors. Although we focus on timing errors as a result of undervolting, the proposed tool can be easily extended to assess timing errors due to several sources of delay increase such as temperature variations, overclocking, transistor aging, and process fluctuations.

Acknowledgment

This research effort has been supported by European Union's H2020 Tetramax project (Grant 761349) through a Technology Transfer Experiment, the H2020 UniServer project (Grant 688540), and the FP7 Clereco project (Grant 611404).

References

[1] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proceedings of the 40th Design Automation Conference, DAC 2003, Anaheim, CA, USA, June 2-6, 2003*. ACM, 2003, pp. 338–342.

[2] K. A. Bowman, J. W. Tschanz, S. Lu, P. A. Aseron, M. M. Khellah, A. Raychowdhury, B. M. Geuskens, C. Tokunaga, C. Wilkerson, T. Karnik, and V. K. De, "A 45 nm resilient microprocessor core for dynamic variation tolerance," *J. Solid-State Circuits*, vol. 46, no. 1, pp. 194–208, 2011.

[3] S. Dighe, S. R. Vangal, P. Aseron, S. Kumar, T. Jacob, K. A. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. K. De, and S. Borkar, "Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraflops processor," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 184–193, 2011.

[4] P. Gupta, Y. Agarwal, L. Dolecek, N. D. Dutt, R. K. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. S. Rosing, M. B. Srivastava, S. Swanson, and D. Sylvester, "Underdesigned and opportunistic computing in presence of hardware variability," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 8–23, 2013.

[5] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, P. Lawthers, and S. Das, "Harnessing voltage margins for energy efficiency in multicore cpus," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2017, pp. 503–516. [Online]. Available: https://doi.org/10.1145/3123939.3124537

[6] K. A. Bowman, J. W. Tschanz, S.-L. L. Lu, P. A. Aseron, M. M. Khellah, A. Raychowdhury, B. M. Geuskens, C. Tokunaga, C. B. Wilkerson, T. Karnik, and V. K. De, "A 45 nm resilient microprocessor core for dynamic variation tolerance," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 194–208, 2011.

[7] Y. Zhang, M. Khayatzadeh, K. Yang, M. Saligane, N. R. Pinckney, M. Alioto, D. T. Blaauw, and D. Sylvester, "irazor: Current-based error detection and correction scheme for PVT variation in 40-nm ARM cortex-r4 processor," *J. Solid-State Circuits*, vol. 53, no. 2, pp. 619–631, 2018. [Online]. Available: https://doi.org/10.1109/JSSC.2017.2749423

[8] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold RISC-V core with DSP extensions for scalable iot endpoint devices," *IEEE Trans. VLSI Syst.*, vol. 25, no. 10, pp. 2700–2713, 2017.

[9] I. Tsiokanos, L. Mukhanov, and G. Karakonstantis, "Low-power variation-aware cores based on dynamic data-dependent bitwidth truncation," in *DATE*. IEEE, 2019, pp. 698–703.

[10] D. M. Bull, S. Das, K. Shivashankar, G. S. Dasika, K. Flautner, and D. T. Blaauw, "A power-efficient 32b ARM ISA processor using timing-error detection and correction for transient-error tolerance and adaptation to PVT variation," in *IEEE International Solid-State Circuits Conference, ISSCC 2010, Digest of Technical Papers, San Francisco, CA, USA, 7-11 February, 2010*. IEEE, 2010, pp. 284–285.

[11] M. Kaliorakis, D. Gizopoulos, R. Canal, and A. Gonzalez, "Merlin: Exploiting dynamic instruction behavior for fast and accurate microarchitecture level reliability assessment," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, June 2017, pp. 241–254. [Online]. Available: https://doi.org/10.1145/3079856.3080225

[12] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: approximate data types for safe and general low-power computation," in *PLDI*. ACM, 2011, pp. 164–174.

[13] G. Georgakoudis, I. Laguna, H. Vandierendonck, D. S. Nikolopoulos, and M. Schulz, "Safire: Scalable and accurate fault injection for parallel multithreaded applications," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2019, pp. 890–899.

[14] H. Schirmeier, C. Borchert, and O. Spinczyk, "Avoiding pitfalls in fault-injection based comparison of program susceptibility to soft errors," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2015, pp. 319–330.

[15] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, "Silent data corruptions at scale," 2021.

[16] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, "Cores that don't count," in *HotOS '21: Workshop on Hot Topics in Operating Systems, Ann Arbor, Michigan, USA, June, 1-3, 2021*, S. Angel, B. Kasikci, and E. Kohler, Eds. ACM, 2021, pp. 9–16.

[17] H. Farbeh, H. Kim, S. G. Miremadi, and S. Kim, "Floating-ecc: Dynamic repositioning of error correcting code bits for extending the lifetime of stt-ram caches," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3661–3675, 2016.

[18] X. Wang, M. Mao, E. Eken, W. Wen, H. Li, and Y. Chen, "Sliding basket: An adaptive ecc scheme for runtime write failure suppression of stt-ram cache," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 762–767.

[19] M. T. Bohr and I. A. Young, "Cmos scaling trends and beyond," *IEEE Micro*, vol. 37, no. 6, pp. 20–29, 2017.

[20] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel, "Towards aging-induced approximations," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2017, pp. 1–6.

[21] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29 - June 07, 2013*. ACM, 2013, pp. 113:1–113:9.

[22] B. Moons and M. Verhelst, "Dvas: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, July 2015, pp. 237–242. [Online]. Available: https://doi.org/10.1109/ISLPED.2015.7273520

[23] G. Karakonstantis, A. Chatterjee, and K. Roy, "Containing the nanometer "pandora-box": Cross-layer design techniques for variation aware low power systems," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 1, no. 1, pp. 19–29, 2011.

[24] L. Rashid, K. Pattabiraman, and S. Gopalakrishnan, "Characterizing the impact of intermittent hardware faults on programs," *IEEE Transactions on Reliability*, vol. 64, no. 1, pp. 297–310, March 2015.

[25] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J. A. Tierno, J. B. Carter, and R. W. Berry, "Active guardband management in power7+ to save energy and maintain reliability," *IEEE Micro*, vol. 33, no. 4, pp. 35–45, July 2013.

[26] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, G. Favor, K. Sankaran, and S. Das, "A System-Level Voltage/Frequency Scaling Characterization Framework for Multicore CPUs," in *Silicon Errors in Logic - System Effects (SELSE)*, 2017. [Online]. Available: https://arxiv.org/pdf/2106.09975

[27] G. Papadimitriou, A. Chatzidimitriou, D. Gizopoulos, V. J. Reddi, J. Leng, B. Salami, O. S. Unsal, and A. C. Kestelman, "Exceeding conservative limits: A consolidated analysis on modern hardware margins," *IEEE Transactions on Device and Materials Reliability*, vol. 20, no. 2, pp. 341–350, 2020. [Online]. Available: https://doi.org/10.1109/TDMR.2020.2989813

[28] P. Koutsovasilis, C. Antonopoulos, N. Bellas, S. Lalis, G. Papadimitriou, A. Chatzidimitriou, and D. Gizopoulos, "The impact of cpu voltage margins on power-constrained execution," *IEEE Transactions on Sustainable Computing*, pp. 1–1, 2020. [Online]. Available: https://doi.org/10.1109/TSUSC.2020.3045195

[29] G. Papadimitriou, A. Chatzidimitriou, and D. Gizopoulos, "Adaptive voltage/frequency scaling and core allocation for balanced energy and performance on multicore cpus," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 133–146. [Online]. Available: https://doi.org/10.1109/HPCA.2019.00033

[30] X. Iturbe, B. Venu, and E. Ozer, "Soft error vulnerability assessment of the real-time safety-related arm cortex-r5 cpu," in *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2016, pp. 91–96.

[31] J. Blome, S. Mahlke, D. Bradley, and K. Flautner, "A microarchitectural analysis of soft error propagation in a production-level embedded microprocessor," in *In Proceedings of the First Workshop on Architecture Reliability*, 2005.

[32] A. Chatzidimitriou and D. Gizopoulos, "Anatomy of microarchitecture-level reliability assessment: Throughput and accuracy," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2016, pp. 69–78. [Online]. Available: https://doi.org/10.1109/ISPASS.2016.7482075

[33] O. Goloubeva, M. Rebaudengo, M. Sonza Reorda, and M. Violante, "Soft-error detection using control flow assertions," in *Proceedings 18th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*, 2003, pp. 581–588.

[34] A. Chatzidimitriou, G. Papadimitriou, C. Gavanas, G. Katsoridas, and D. Gizopoulos, "Multi-bit upsets vulnerability analysis of modern microprocessors," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, 2019, pp. 119–130. [Online]. Available: https://doi.org/10.1109/IISWC47752.2019.9042036

[35] J. Carreira, H. Madeira, and J. Gabriel, "Xception: Software fault injection and monitoring in processor functional units1," 1995.

[36] H. Madeira, D. Costa, and M. Vieira, "On the emulation of software faults by software fault injection," in *Proceeding International Conference on Dependable Systems and Networks. DSN 2000*, 2000, pp. 417–426. [Online]. Available: https://doi.org/10.1109/ICDSN.2000.857571

[37] D. Kammler, J. Guan, G. Ascheid, R. Leupers, and H. Meyr, "A fast and flexible platform for fault injection and evaluation in verilog-based simulations," in *2009 Third IEEE International Conference on Secure Software Integration and Reliability Improvement*, 2009, pp. 309–314.

[38] V. Sieh, O. Tschache, and F. Balbach, "Verify: evaluation of reliability using vhdl-models with embedded fault descriptions," in *Proceedings of IEEE 27th International Symposium on Fault Tolerant Computing*, 1997, pp. 32–36.

[39] N. J. Wang, A. Mahesri, and S. J. Patel, "Examining ACE analysis reliability estimates using fault-injection," in *34th International Symposium on Computer Architecture (ISCA 2007), June 9-13, 2007, San Diego, California, USA*, D. M. Tullsen and B. Calder, Eds. ACM, 2007, pp. 460–469. [Online]. Available: https://doi.org/10.1145/1250662.1250719

[40] M. Maniatakos, N. Karimi, C. Tirumurti, A. Jas, and Y. Makris, "Instruction-level impact analysis of low-level faults in a modern microprocessor controller," *IEEE Transactions on Computers*, vol. 60, no. 9, pp. 1260–1273, Sep. 2011.

[41] K. Parasyris, G. Tziantzoulis, C. D. Antonopoulos, and N. Bellas, "Gemfi: A fault injection tool for studying the behavior of applications on unreliable substrates," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2014, pp. 622–629.

[42] G. Papadimitriou and D. Gizopoulos, "Demystifying the system vulnerability stack: Transient fault effects across the layers," in *48th International Symposium on Computer Architecture (ISCA 2021), June 14-19, 2021, Worldwide Online Event*. IEEE, 2021, pp. 902–915. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00075

[43] A. Chatzidimitriou, P. Bodmann, G. Papadimitriou, D. Gizopoulos, and P. Rech, "Demystifying soft error assessment strategies on arm cpus: Microarchitectural fault injection vs. neutron beam experiments," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 26–38. [Online]. Available: https://doi.org/10.1109/DSN.2019.00018

[44] P. N. Sanda, J. W. Kellington, P. Kudva, R. Kalla, R. B. McBeth, J. Ackaret, R. Lockwood, J. Schumann, and C. R. Jones, "Soft-error resilience of the ibm power6 processor," *IBM Journal of Research and Development*, vol. 52, no. 3, pp. 275–284, May 2008.

[45] M. Li, P. Ramachandran, S. K. Sahoo, S. V. Adve, V. S. Adve, and Y. Zhou, "Understanding the propagation of hard errors to software and implications for resilient system design," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2008, Seattle, WA, USA, March 1-5, 2008*, S. J. Eggers and J. R. Larus, Eds. ACM, 2008, pp. 265–276. [Online]. Available: https://doi.org/10.1145/1346281.1346315

[46] N. J. Wang and S. J. Patel, "Restore: Symptom-based soft error detection in microprocessors," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 3, pp. 188–201, July 2006.

[47] N. Foutris, D. Gizopoulos, J. Kalamatianos, and V. Sridharan, "Assessing the impact of hard faults in performance components of modern microprocessors," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, Oct 2013, pp. 207–214.

[48] G. Georgakoudis, I. Laguna, D. S. Nikolopoulos, and M. Schulz, "REFINE: realistic fault injection via compiler-based instrumentation for accuracy, portability and speed," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2017, Denver, CO, USA, November 12 - 17, 2017*, B. Mohr and P. Raghavan, Eds. ACM, 2017, pp. 29:1–29:14. [Online]. Available: https://doi.org/10.1145/3126908.3126972

[49] A. Shye, T. Moseley, V. J. Reddi, J. Blomstedt, and D. A. Connors, "Using process-level redundancy to exploit multiple cores for transient fault tolerance," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, June 2007, pp. 297–306.

[50] V. C. Sharma, A. Haran, Z. Rakamaric, and G. Gopalakrishnan, "Towards formal approaches to system resilience," in *2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing*, Dec 2013, pp. 41–50.

[51] R. Venkatagiri, K. Ahmed, A. Mahmoud, S. Misailovic, D. Marinov, C. W. Fletcher, and S. V. Adve, "gem5-approxilyzer: An open-source tool for application-level soft error analysis," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 214–221.

[52] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *ASPLOS*. ACM, 2012, pp. 301–312.

[53] K. Parasyris, V. Vassiliadis, C. D. Antonopoulos, S. Lalis, and N. Bellas, "Significance-aware program execution on unreliable hardware," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, Apr. 2017. [Online]. Available: https://doi.org/10.1145/3058980

[54] J. Sartori, J. Sloan, and R. Kumar, "Stochastic computing: Embracing errors in architecture and design of processors and applications," in *2011 Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, 2011, pp. 135–144.

[55] E. Krimer, P. Chiang, and M. Erez, "Lane decoupling for improving the timing-error resiliency of wide-simd architectures," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 237–248.

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

[57] P. Bodmann, G. Papadimitriou, D. Gizopoulos, and P. Rech, "The impact of soc integration and os deployment on the reliability of arm processors," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021, pp. 223–225. [Online]. Available: https://doi.org/10.1109/ISPASS51385.2021.00040

[58] C. Chang, W. Yin, and M. Erez, "Assessing the impact of timing errors on HPC applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019, Denver, Colorado, USA, November 17-19, 2019*, M. Taufer, P. Balaji, and A. J. Peña, Eds. ACM, 2019, pp. 70:1–70:19. [Online]. Available: https://doi.org/10.1145/3295500.3356184

[59] A. Rahimi, L. Benini, and R. K. Gupta, "Application-adaptive guardbanding to mitigate static and dynamic variability," *IEEE Trans. Computers*, vol. 63, no. 9, pp. 2160–2173, 2014.

[60] G. Tziantzioulis, A. M. Gok, S. M. Faisal, N. Hardavellas, S. Ogrenci-Memik, and S. Parthasarathy, "b-hive: A bit-level history-based error model with value correlation for voltage-scaled integer and floating point units," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[61] X. Jiao, A. Rahimi, Y. Jiang, J. Wang, H. Fatemi, J. P. de Gyvez, and R. K. Gupta, "Clim: A cross-level workload-aware timing error prediction model for functional units," *IEEE Transactions on Computers*, vol. 67, no. 6, pp. 771–783, June 2018.

[62] J. Constantin, L. Wang, G. Karakonstantis, A. Chattopadhyay, and A. Burg, "Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*, W. Nebel and D. Atienza, Eds. ACM, 2015, pp. 381–386.

[63] I. Tsiokanos and G. Karakonstantis, "Exhero: Execution history-aware error-rate estimation in pipelined designs," *IEEE Micro*, vol. 41, no. 1, pp. 61–68, 2021.

[64] I. Tsiokanos, L. Mukhanov, D. S. Nikolopoulos, and G. Karakonstantis, "Significance-driven data truncation for preventing timing failures," *IEEE Transactions on Device and Materials Reliability*, vol. 19, no. 1, pp. 25–36, March 2019.

[65] J. Constantin, A. P. Burg, Z. Wang, A. Chattopadhyay, and G. Karakonstantis, "Statistical fault injection for impact-evaluation of timing errors on application performance," in *Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, TX, USA, June 5-9, 2016*. ACM, 2016, pp. 13:1–13:6.

[66] S. Roy and K. Chakraborty, "Predicting timing violations through instruction-level path sensitization analysis," in *DAC*. ACM, 2012, pp. 1074–1081.

[67] J. Gaudiot, J. Kang, and W. Ro, *Techniques to Improve Performance Beyond Pipelining: Superpipelining, Superscalar, and VLIW*, ser. Advances in Computers, 2005, pp. 1–34.

[68] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*, New York, USA: Springer, 2009.

[69] D. Garyfallou, I. Tsiokanos, N. Evmorfopoulos, G. Stamoulis, and G. Karakonstantis, "Accurate estimation of dynamic timing slacks using event-driven simulation," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*, 2020, pp. 225–230.

[70] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Slack redistribution for graceful degradation under voltage overscaling," in *Proceedings of the 15th Asia South Pacific Design Automation Conference, ASP-DAC 2010, Taipei, Taiwan, January 18-21, 2010*. IEEE, 2010, pp. 825–831.

[71] Y. Fan, T. Jia, J. Gu, S. Campanoni, and R. Joseph, "Compiler-guided instruction-level clock scheduling for timing speculative processors," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[72] G. Hoang, R. B. Findler, and R. Joseph, "Exploring circuit timing-aware language and compilation," in *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2011, Newport Beach, CA, USA, March 5-11, 2011*, R. Gupta and T. C. Mowry, Eds. ACM, 2011, pp. 345–356. [Online]. Available: https://doi.org/10.1145/1950365.1950405

[73] T. Jia, R. Joseph, and J. Gu, "An instruction-driven adaptive clock management through dynamic phase scaling and compiler assistance for a low power microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 8, pp. 2327–2338, 2019.

[74] I. Tsiokanos, L. Mukhanov, D. S. Nikolopoulos, and G. Karakonstantis, "Variation-aware pipelined cores through path shaping and dynamic cycle adjustment: Case study on a floating-point unit," in *Proceedings of the International Symposium on Low Power Electronics and Design, ISLPED 2018, Seattle, WA, USA, July 23-25, 2018*. ACM, 2018, pp. 52:1–52:6.

[75] X. Jiao, D. Ma, W. Chang, and Y. Jiang, "Levax: An input-aware learning-based error model of voltage-scaled functional units," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2020.

[76] I. Tsiokanos, L. Mukhanov, G. Georgakoudis, D. S. Nikolopoulos, and G. Karakonstantis, "DEFCON: generating and detecting failure-prone instruction sequences via stochastic search," in *DATE*. IEEE, 2020, pp. 1121–1126.

[77] J. N. Rodrigues, M. Kamuf, H. Hedberg, and V. Owall, "A manual on asic front to back end design flow," in *2005 IEEE International Conference on Microelectronic Systems Education (MSE'05)*, 2005, pp. 75–76.

[78] 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, p. 1–7, Aug. 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718

[79] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2009, pp. 44–54.

[80] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The nas parallel benchmarks summary and preliminary results," in *Supercomputing '91:Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, Nov 1991, pp. 158–165.

[81] Implementation of the Sobel Filter in C, [online], available: https://github.com/petermlm/SobelFilter.

[82] OpenRISC Community, "OpenRISC 1000 architecture manual.".

[83] IEEE 754-2008. IEEE 754-2008 Standard for Floating-Point Arithmetic.

[84] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benin, "A transprecision floating-point platform for ultra-low power computing," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018, pp. 1051–1056.

[85] A. Rahimi, A. Marongiu, R. K. Gupta, and L. Benini, "A variability-aware openmp environment for efficient execution of accuracy-configurable computation on shared-fpu processor clusters," in *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Sept 2013, pp. 1–10. [Online]. Available: https://doi.org/10.1109/CODES-ISSS.2013.6659022

[86] NanGate FreePDK45 Open Cell Library, [online], available: http://nangate.com.

[87] 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*, 2009, pp. 502–506. [Online]. Available: https://doi.org/10.1109/DATE.2009.5090716

[88] J. Patel, "Cmos process variations: A critical operation point hypothesis," April 2008 [Online].