2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)

gem5-MARVEL: Microarchitecture-Level Resilience Analysis of Heterogeneous SoC Architectures

Odysseas Chatzopoulos George Papadimitriou Vasileios Karakostas Dimitris Gizopoulos
Department of Informatics & Telecommunications, University of Athens, Greece
{od.chatzopoulos, georgepap, vkarakos, dgizop}@di.uoa.gr

Abstract—**In this paper, we present gem5-MARVEL, the first consolidated microarchitecture-level fault injection infrastructure for heterogeneous System-on-Chip architectures comprising CPUs of all major Instruction Set Architectures (ISAs) and different types of domain-specific accelerators. The proposed framework is based on a modular design that facilitates flexible fault injection scenarios that correspond to different fault models and system configurations. gem5-MARVEL includes a set of libraries for the automation of fault injection and the analysis of the effects of hardware faults at full system execution. We evaluate the proposed framework on several 64-bit CPU ISAs: x86, Arm, and RISC-V, as well as on different designs of domain-specific accelerators. The case studies we present unveil important insights and demonstrate the effectiveness of the proposed infrastructure in the analysis of the impact of faults on different types of heterogeneous computing systems. gem5-MARVEL facilitates broad design space exploration for entire heterogeneous computing systems at the microarchitecture level, where resilience under realistic fault scenarios can be simultaneously analyzed with performance (the typical use of microarchitectural simulators).**

Index Terms—**Reliability, CPUs, accelerators, heterogeneous architectures, transient faults, permanent faults, silent data corruptions, microarchitecture-level fault injection**

I. Introduction

With Moore's Law slowing down [1], domain-specific accelerators (DSAs) have become increasingly important due to their superior performance and efficiency on specific tasks compared to general-purpose CPUs [2]. Accelerators are specialized hardware engines designed for specific domains, such as graphics [3], deep learning [4], bioinformatics [5], image processing [6], and simulation [7]. They deliver high performance gains by reducing overheads, offering fast specialized operations, optimized memory systems, and parallelism. General-purpose CPUs perform better at control-intensive tasks, but are less efficient than DSAs for specific tasks [8]. As modern computing systems become more complex and heterogeneous, multiple CPUs of different ISAs, and a diverse set of DSAs need to cooperate for optimized performance and energy efficiency.

However, the reliability of integrated circuits deteriorates with increasing design complexity, decreasing feature sizes of transistors, and the wide manufacturing variability; all these make modern computing systems severely vulnerable to various types of faults, such as transient and permanent faults [9]. Transient faults are temporary and can be caused by environmental factors, such as radiation or power fluctuations, while permanent faults are caused by manufacturing defects or aging effects over time. Both types of faults can significantly impact the system's reliability, and it is crucial to evaluate and mitigate their effects. Hardware faults of all types can lead to unexpected behavior, system crashes, or even worse, data corruptions [10]–[15]. More importantly, DSAs are often critical components for many applications (e.g., [4]–[6]), and thus, the loss of data or incorrect processing due to a fault in a DSA can have severe consequences, especially in safety- and mission-critical systems [16].

Reliability assessment of computing systems is crucial to guarantee correct operation. This assessment is particularly significant at **early design stages** for complex Systems-on-Chip (SoCs) with CPUs and DSAs. Early identification of weak hardware structures that are more vulnerable to faults can prevent system failures and data corruptions, guiding effective countermeasures. Calculating the Architectural Vulnerability Factor (AVF) for each microarchitectural component is the comprehensive way to assess the vulnerability of the entire system stack [17]–[20]. Two prevailing methods to estimate the AVF are Architecturally Correct Execution (ACE) analysis [21] and statistical fault injection (SFI) [18]. While ACE analysis is fast, it can overestimate AVF and has implementation difficulties. SFI, although slower, provides accurate results [17], [20], and thus, SFI is the focus of this work. Still, both methods operate at the microarchitecture-level and calculate the cross-layer AVF [17] of the system.

In the rising popularity of the RISC-V ISA [30], [31], enabling diverse designs from various vendors, reliability evaluation becomes ever more critical. The expanding utilization of RISC-V in safety-critical applications [32], [33], as well as its emerging adoption in cloud computing and HPC, makes this assessment essential to ensure resilience against hardware faults when compared to classic ISAs like x86 and Arm. To this end, a heterogeneous microarchitecture-level fault injection framework is necessary to assess CPU and DSA resilience jointly. Currently, as shown in Table I, there is no fault injection framework that operates at the microarchitecture-level and targets all major aspects of modern computing systems, including microarchitecture-level fault injection to both CPU and DSAs (jointly as an SoC or independently), support for multiple ISAs, and for multiple fault models, etc. Such a framework is essential for chip designers and researchers to assess system vulnerability against hardware faults, guide design changes for improved resilience, and identify critical components needing extra protection early in the design cycle.

TABLE I
STATE-OF-THE-ART AND CONTRIBUTIONS OF THIS WORK REGARDING RESILIENCE ANALYSIS FRAMEWORKS.

State-of-the-Art	Simulation			Fault Injection			ISA Support			Fault Models		Bit-Flips Support		Metrics	
	uArch	gem5	FS	CPU	DSA	SoC	x86	Arm	RISC-V	Transient	Permanent	Single	Multiple	AVF	HVF
FIMSIM [22]	✓	✓*		✓						✓	✓	✓	✓	✓	
GeFIN [23]	✓	✓	✓	✓			✓	✓		✓	✓	✓	✓	✓	✓
MaFIN [23]	✓		✓	✓			✓			✓	✓	✓	✓	✓	
GemFI [24]		✓		✓			✓†			✓	✓	✓			
Thales [25]/Fidelity [26]										✓		✓	✓		
LLFI [27]/LLTFI [28]				✓			✓	✓		✓		✓			
gem5-Approxilyzer [29]		✓	✓	✓			✓§			✓		✓			
This Work	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

*FIMSIM uses M5 simulator, the predecessor of gem5, and is based on Alpha ISA. †Not tested ISA. §It also works for SPARC.

Developing this consolidated resilience analysis framework for complex computing systems is challenging, particularly for heterogeneous systems with diverse CPUs and DSAs. Thus, a customizable microarchitecture-level fault injection framework is crucial to evaluate the resilience of CPUs and accelerators, regardless of their architecture or specific use case [34].

In this paper, we propose gem5-MARVEL, the first consolidated microarchitecture-level fault injection framework for evaluating transient and permanent fault resilience in heterogeneous systems early in the design cycle. The framework encompasses multiple CPUs with different ISAs and a variety of DSAs. By injecting faults, gem5-MARVEL simulates real-world scenarios, offering significant insights into the entire heterogeneous system's response to faults. Leveraging and combining the state-of-the-art gem5 simulator for CPUs [35] and gem5-SALAM for DSAs [36], gem5-MARVEL provides flexibility, fast design-space exploration, and various scenarios for comprehensive reliability evaluation. The diverse operation modes for CPUs and accelerators enable covering nearly every aspect of reliability evaluation.

Motivation & Importance of the work: Reliability evaluation of a modern heterogeneous computing system at the early design stages, is more important than ever before, because:

1) DSAs are specialized and integrated into larger SoCs with general-purpose CPUs, adding complexity and potential fault sources that impact overall system reliability.
2) Hardware failures and downtime costs are substantial, particularly in data centers with extensive CPU and accelerator deployments. Early reliability evaluation can identify potential weak hardware structures and mitigate the impact of faults in them on the system (either at the software or at the hardware layer).
3) Transient and permanent faults can cause system failures and data corruptions, leading to financial losses and reputation damages. A consolidated fault injection framework for CPUs of different ISAs and DSAs is critical to ensure system reliability and performance.
4) Unlike existing tools (Table I), gem5-MARVEL focuses on reliability evaluation at the microarchitecture level, providing fast and accurate results for all major ISAs and a large variety of DSAs in the entire SoC. A recent study [17] confirms the significance of microarchitecture-level fault injection and the flaws of other types of injections at the software or the ISA layers (e.g., [27]–[29]). Therefore, the value that gem5-MARVEL brings is high and the breadth of studies across different ISAs, microarchitectures and DSA designs is very large.

In summary, the main contributions of this paper are:

1) We extend the full-system support of gem5-SALAM to support the RISC-V ISA along with the Arm ISA, which was the main ISA that gem5-SALAM supported. Thus, the accelerator designs can be simulated along with any of the two prevailing RISC ISAs, enabling broader design space exploration studies.
2) We propose the gem5-MARVEL fault injection framework, that operates at the microarchitecture-level and supports transient and permanent fault injections to all hardware structures of the CPU and for the three prevailing ISAs (Arm, x86, RISC-V).
3) We present, for the first time, a consolidated fault injection framework at the microarchitecture-level that targets heterogeneous SoCs with diverse CPUs and DSAs. We also present numerous insights and observations.
4) gem5-MARVEL evaluates both the end-to-end AVF and HVF (Hardware Vulnerability Factor) [37] through fault injection. Recent studies (e.g., [17], [20]) have shown that both AVF and HVF are essential metrics for any reliability evaluation study.

II. BACKGROUND & RELATED WORK

A. Reliability Concepts & Background

AVF calculates the probability that a fault in a hardware structure of a microprocessor will produce a program visible error. AVF takes into account both the microarchitecture and the program structure and its input data [19]. AVF is technology-independent and captures the full-system vulnerability, including all phases of fault activation and propagation through the microarchitecture, the architecture, and the software layers [38]. For example, if a fault occurs in the physical register file of an out-of-order microprocessor, it may not affect the program's execution if that fault is discarded due to a pipeline flush. However, even if the fault is eventually propagated to the software layer, the algorithm may discard or overwrite the corrupted data and still produce the correct result. Therefore, faults can be masked at either the hardware or the software layer.

To further aid error protection strategies, the full system stack can be divided into layers, and the AVF can be calculated based on individual vulnerability factors for each layer. The System Vulnerability Stack, proposed in [37], illustrates this approach by assigning a vulnerability value to a bit at each layer to determine its visibility. Another important reliability metric is the HVF, which calculates the hardware portion of the AVF. While AVF provides the full-stack vulnerability evaluation, including the microarchitecture, the architecture, and the software layers, HVF provides hardware designers with additional insights beyond what AVF analysis alone offers, particularly regarding microarchitecture-dependent vulnerability.

Computer architects often use AVF as a reference to determine where and when to introduce redundancy in the system design, especially for assessing runtime reliability techniques (e.g., [39], [40]) and for the cross-layer evaluation of fault-tolerant methods [17]. However, relying solely on the AVF for the evaluation of microarchitectural decisions made during the design stage, such as the sizing and the organization of structures, is not an ideal approach. AVF does not consider program-level operations that may be masked, which could obscure changes in hardware performance. For instance, AVF does not account for the placement of dead reads and cannot provide insights into this behavioral change [37]. Therefore, HVF is a useful metric to analyze alongside AVF, as it more diligently supports design decisions for fault protection. gem5-MARVEL is a tool that facilitates both AVF and HVF measurements.

B. Early Reliability Assessment

Early reliability assessment typically occurs using models that are developed prior to the creation of silicon prototypes. These models can be categorized into three main levels: architecture, microarchitecture, and RTL (Register-Transfer Level). These levels vary in terms of detail, with the most abstract available early in the design process and the most detailed (RTL) emerging later on [41], [42]. Architecture-level models often lack most, if not all, hardware-specific details, providing a functional emulation at the software level [17]. In contrast, microarchitecture-level models (e.g., based on gem5) encompass functional and timing-accurate representations of the microarchitecture and offer accuracy at the clock cycle level. Additionally, microarchitecture-level models accurately depict various memory elements within the system, such as SRAMs, registers, and non-logic-related flops and latches, such as state machines. RTL models provide a comprehensive description of the implemented hardware, encompassing logic components and SRAM elements. Simulation time required for each abstraction layer correlates with the level of detail, with each increase in detail adding approximately two orders of magnitude to the simulation time.

C. Microarchitecture-Level Fault Injection

Typically, designers use either Statistical Fault Injection (SFI) [18] or analytical methods like the Architecturally Correct Execution (ACE) analysis [19] to gain insights into the resilience of programs against transient faults[1]. For transient faults, both methods aim to measure the AVF of hardware structures, but each has its advantages and disadvantages. SFI is very accurate, but slow since it requires multiple runs to reach high confidence, whereas ACE analysis is fast, but has a high development effort and may overestimate AVF (up to 3x overestimation [20]). Statistical fault injection is a reliability estimation technique that can provide full-system AVF estimation by directly accessing the program output produced with the injected faults. Statistical fault-injection is a widely adopted method for reliability assessment. It offers flexibility in terms of accuracy, depending on the size of the statistical sample, while providing failure samples generated through simulation. However, it comes with the drawback of requiring multiple simulations, which can be time-consuming and, depending on the level of model detail, may be considered impractical.

[1]ACE applies to transient faults only, while fault injection works for both transient and permanent faults since the entire execution of the program is simulated in the presence of faults.

D. Related Work

George *et al.* [43] argued that a microarchitecture-level fault injection approach can provide accurate vulnerability estimations and proposed a fault injection framework on top of the PTLSim [44]. However, that framework targeted only the x86 ISA and the register file. Yalcin *et al.* [22] proposed FIMSIM, an SFI framework for microarchitectural simulators. While that framework provided fault injection support for various structures, it mainly targeted the M5 simulator, which has been superseded by gem5 and was evaluated only with simple in-order cores of the Alpha ISA. Parasyris *et al.* [24] proposed GemFI, an ISA-level SFI framework based on gem5. It, thus, only supports fault injection for the architectural register file and targets a rather old version of gem5. Kaliorakis *et al.* [23] proposed GeFIN, an SFI framework built on top of gem5. GeFIN supports injections on different microprocessor structures for x86 and Arm, but not for RISC-V or any kind of accelerator, and targets a rather old version of gem5.

The software level SFI approaches typically operate at the program source code [45], assembly [46]–[50], or at the compiler intermediate level [27], [51]–[55]. Prior research [56] has shown that IR-level and assembly-level injections deliver very similar results. Venkatagir *et al.* [29] introduced gem5-Approxilyzer, which operates at the architectural level and precisely evaluates how an instruction error affects the final output. However, these approaches at the software or ISA layer may not fully capture critical fault manifestation and propagation aspects, potentially leading to misleading reliability findings, as it was recently demonstrated in [17].

Li *et al.* [57] characterized the propagation of errors in DNN applications by using fault injection on a high-level simulator for DNN accelerators. Mahmoud *et al.* [58] proposed GoldenEye, a functional simulator with fault injection capabilities for common and emerging numerical formats that target DNN frameworks and accelerators. However, these simulators support only DNN frameworks and accelerators, and lack modeling support at the microarchitecture level. Several high level software tools have been proposed for performing fault injection studies within particular machine learning frameworks, such as TensorFI [59], MindFI [60], and PyTorchFI [61]. However, all these tools inject faults at framework level, into the ML model parameters and the outputs of operators. Reagen *et al.* [62] proposed Ares, a DNN-specific fault injection framework that operates at application level. Agarwal *et al.* proposed LLTFI [28], a fault injection tool for ML applications that operates at the LLVM IR level. Still, these approaches operate at software-level and do not accurately reflect the AVF [17], [25].

III. Microarchitecture-Level SFI Framework

A. Simulation Platform

Fault injection should ideally be based on a real system with physical injection capabilities in all microarchitectural structures (such a system does not exist, but even if it did, making protection decisions after studying and implementing it would be too late) or a very detailed low-level simulator (e.g., RTL, gate) which allows the same. Although low-level simulators may provide accurate fault effects, their simulation throughput is extremely low to be affordable and cannot model long-running workloads with OS activity (RTL simulation is several orders of magnitude slower than cycle accurate microarchitectural simulation [63]–[65]. To this end, gem5-MARVEL relies on microarchitecture-level simulation using the latest version of the gem5 simulator [35], [66], [67], which allows (i) deterministic, (ii) end-to-end, (iii) cycle level execution of (iv) large workloads (v) on top of an operating system; this combination is impossible at lower levels.

B. gem5-based Accelerator Modeling

Recently, several efforts have been made to integrate domain specific accelerator models with the state-of-the-art gem5 simulation environment. These efforts include, among others, gem5-Aladdin [68] and PARADE [69]. Additionally, SystemC support was added to gem5 [70] enabling the potential of cycle-accurate modeling of hardware structures including accelerator datapaths. Therefore, existing works for modeling domain-specific accelerators rely either on pre-RTL [68], [69] or RTL-based solutions [70] (e.g., C/C++ based models).

However, all these options have significant disadvantages. On one hand, both gem5-Aladdin and PARADE, although they are both pre-RTL frameworks, and thus, comprehensive, they provide limited support for design space exploration due to their restrictive simulation semantics. In addition, they suffer from low simulation fidelity when data availability, parallelism, and timing are not decoupled from the input dataset. On the other hand, while SystemC support offers the potential of highly accurate modeling, being an RTL-based alternative, it requires considerably higher design effort and eventually provides lower throughput (see Section III-A) than the other two pre-RTL frameworks. To this end, in this work we are based on a new pre-RTL framework, which overcomes these limitations and provides flexibility, and excellent tradeoffs among performance, simulation fidelity, and ease-of-use.

1) gem5-SALAM: gem5-MARVEL is based on gem5-SALAM [36] that uses an advanced dynamic graph execution engine based on LLVM [71]. gem5-SALAM instruments the LLVM IR (Intermediate Representation) to model DSAs using C descriptions of their functionality. Its main advantages are:

1) It provides accurate representation of the accelerator datapath based on analysis of the LLVM intermediate representation of the accelerated algorithm.
2) It provides cycle level modeling through the dynamic LLVM-based runtime execution engine.
3) It decouples the datapath and memory components to aid design space exploration and system optimization.
4) gem5's tight integration enables seamless and intricate interaction between the accelerator and other system modules, including the CPU and the memory subsystem.

For the above reasons, we believe that gem5-SALAM is the ideal candidate for integration into our fault-injection framework to complement the CPU side of the system with the accelerators side. This allows us to evaluate the performance and reliability of a plethora of different accelerator architectures, ranging from loosely-coupled multi-accelerator configurations to tightly-coupled coprocessors.

2) gem5-SALAM Architecture Overview: gem5-SALAM consists of two core components: the Compute Unit and the Communications Interface. The Compute Unit serves as the datapath for the custom accelerator, while the Communications Interface enables memory access, control, and synchronization through memory access ports, Memory-Mapped Registers (MMRs), and interrupt lines. Memory access ports allow parallel access to various memory types like scratchpad memories (SPMs) and register banks. MMRs consist of configurable status, control, and data registers, facilitating low-level device configuration and communication between the accelerator and the host, as well as between multiple accelerators in a cluster. The accelerator is treated as a memory-mapped device, allowing the host to use interrupt signals for synchronization without constant polling. gem5-SALAM also includes Direct Memory Access (DMA) devices and custom memories, which can be seamlessly integrated into accelerator designs, enhancing its versatility. The SoC architecture, including gem5-SALAM's features, is shown in Figure 1. The accelerator designs are loosely coupled and communicate with the host CPU through MMRs and DMA transactions. The CPU writes input and output memory addresses to the accelerator MMRs and directs the accelerator to start computation. DMA transfers data between the accelerator and its SPMs or Register Banks, where calculations are performed, and then the results are transferred back to the system memory. After task completion, the accelerator notifies the host via a pre-defined interrupt.

C. Adding RISC-V ISA support in gem5-SALAM

Currently, gem5-SALAM only supports the Arm ISA when it comes to the simulated system processor cores. However, the tremendous growth of the RISC-V ecosystem in the past few years, and its rapid adoption in both academia and industry,

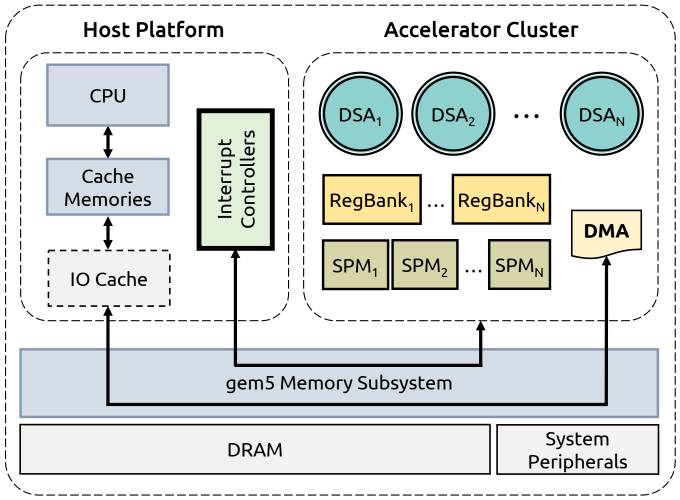

Fig. 1. gem5-based SoC architecture and interconnection.

motivated us to port gem5-SALAM to also support the RISC-V ISA and system configuration. The recent introduction of RISC-V full-system execution support into gem5 [35], [66] was highly beneficial to this endeavor. Nonetheless, the main challenge of extending the framework to support also the RISC-V ISA is to identify the Arm specific components (i.e., the ISA dependencies) and translate them into the corresponding RISC-V ones. We summarize below the major components that had strong dependency on the Arm platform. Specifically:

1) The interrupt system used by gem5-SALAM hardware components that employed the Arm General Interrupt Controller (GIC) for posting interrupts to the host CPU,
2) The automatic gem5 configuration script generator that used an Arm gem5 configuration script as a template.

Next, we briefly discuss these two ISA-dependent features, and how we convert them to enable RISC-V support into gem5-SALAM.

1) From (Arm) GIC to (RISC-V) PLIC: gem5-SALAM hardware components, use the Arm GIC to send and receive interrupts to and from the CPU, aiding the synchronization between accelerator and host and removing the overheads of constant polling. We translated these functions at both hardware and software levels to the Platform Level Interrupt Controller (PLIC) that is present in the current gem5 RISC-V model.

2) Automatic Configuration Script Generator: gem5-SALAM simplifies accelerator-rich SoC development using an automatic configuration script generator. The generator parses a single YAML file containing the system description to produce complex configuration scripts. To enable RISC-V ISA support in the latest gem5 version, we replaced the Arm-specific script template with an existing RISC-V full-system configuration script. Additionally, we made necessary modifications to initialize gem5-SALAM components and included accelerator memory-mapped addresses within the RISC-V platform's address ranges.

D. Configurations, Benchmark Suite & Accelerator Designs

gem5-MARVEL supports all dominant 64-bit ISAs, i.e., Arm, x86, and RISC-V, and for the CPU side evaluations we present in the following sections, we model the same out-of-order microarchitecture for every ISA (microarchitectural modifications can of course be arbitrarily implemented in gem5), as shown in Table II. gem5-MARVEL targets tens of hardware structures for both CPUs and DSAs (see Section IV-E). In this paper, we showcase the framework for 5 of the most important structures of a modern OoO microprocessor: the L1 data and instruction caches, the Physical Register File, and the load and store queues. We employ a comprehensive and diverse set of 15 workloads from the MiBench benchmarks suite [72] for all 3 different CPU ISAs. This suite is commonly used in reliability studies [17], [43], [63], [64], [73]–[81] *as it facilitates complete end-to-end executions*.

For the DSAs evaluation, we present results for fault injections in the two largest types of accelerator memory structures: the scratchpad memories and the register banks of the designs (see section IV-E for details). For the needs of our study, we also employ 8 MachSuite accelerator designs [82] (see Table IV in Section V-E) and we measure their AVF when running full-system simulations based on the RISC-V ISA for the CPU side. For each structure, 1,000 single-bit faults are randomly generated following the uniform distribution as defined in [18], resulting in nearly 250,000 fault injection runs in total for all benchmarks and accelerator designs, 3 different 64-bit ISAs, and all hardware components for CPU and DSAs. We follow the widely adopted formulation of [18] for the statistical fault sampling calculations; our 1,000 faults correspond to 3% error margin with 95% confidence level.

IV. Features & Implementation

A. Fault Models & Effects Classification

1) Fault Models: gem5-MARVEL models the following types of faults on microarchitectural structures: transient and permanent faults, as well as their combinations (see Table III). These types of fault models allow a wide analysis of the effect of different factors that affect reliability: latent manufacturing defects, environmental conditions, early-life failures, chip aging and wear-out, and voltage scaling. Additionally, gem5-MARVEL provides support for fault injection experiments that can simulate multiple faults occurring in various combinations to mimic the spatial and temporal behavior of faults in hardware structures. These combinations involve injecting multiple faults of any type in a single structure, or multiple faults occurring

TABLE II
Major Simulator Configurations for each ISA.

Parameter	Value
ISA	RISC-V / Arm / x86
Pipeline	64-bit OoO (8-issue)
L1 Instruction Cache	32KB, 64B line, 128 sets, 4-way
L1 Data Cache	32KB, 64B line, 128 sets, 4-way
L2 Cache	1MB, 64B line, 2048 sets, 8-way
Physical Register File	128 Int; 128 FP
LQ/SQ/IQ/ROB entries	32/32/64/128

TABLE III
FAULT MODELS DESCRIPTION.

Fault Model	Description
Transient	A storage element's bit value is flipped in a clock cycle of the program execution; the bit position and the cycle can be set arbitrarily (randomly or directed)
Permanent	A storage element's bit value is permanently set '0' or to '1'; the bit position can be set arbitrarily

in different structures. Although, due to space limitations, we show results only for single bit flips, gem5-MARVEL also supports multibit fault injection.

2) Fault Effects Classification: For the AVF evaluations, gem5-MARVEL classifies the effect on the program output into the following classes (typically used in all SFI studies):

Masked: Simulation finished with no deviations from a fault-free execution. Thus, the fault did not affect the system or the application in any observable way.

Silent Data Corruption (SDC): Simulation finished normally, but the program output was different from the fault-free simulation, without any observable indication.

Crash: A simulation was interrupted by a catastrophic event, preventing it from completing or reaching the program's end. Consequently, no program output was generated.

On the other hand, HVF campaigns have two separate classes of fault effect classification:

Masked: The fault did not reach the commit stage (i.e., did not reach the software layer) until the end of simulation.

Corruption: A mismatch was detected in the commit stage compared to the fault-free trace (i.e., the fault became visible to the software layer). The mismatch may pertain to the instruction, operands, data transactions, or program order.

B. Implementation of gem5-MARVEL

The objectives of gem5-MARVEL are enumerated below:

Ensuring Accuracy: gem5-MARVEL aims to deliver accurate vulnerability reports (AVF and HVF). It achieves this by executing applications or accelerator models until completion, unless a fault is detected earlier, which is masked (e.g., invalid cache line or overwritten before being read). This approach captures the effect of faults on the program outcome.

Increasing Speed of Fault Injection Campaigns: gem5-MARVEL optimizes SFI campaigns, particularly for transient faults, by utilizing multiple workstations. It can promptly terminate an SFI run when a fault is inserted into an invalid or unused hardware structure entry or when a faulty entry is overwritten before being read. These optimizations significantly reduce the time needed for individual fault injection runs.

High Configurability: gem5-MARVEL is highly configurable, allowing users to specify various fault models, fault injection targets, and fault characteristics. It uses fault masks that specify the injection of faults, containing information about targeted components and timing.

Support for Different Hardware Configurations: gem5-MARVEL incorporates configuration presets that define attributes like ISA, memory configuration, CPU core type (in-order, out-of-order), multicore setup, system setup, disk images, kernel versions, etc. It also supports a list of hardware structures for fault injections, and new presets can be easily added to cover various requirements.

Flexibility and Ease of Expansion: gem5-MARVEL provides high flexibility and ease of expansion, making it suitable for reliability evaluation studies involving different microprocessors or accelerator designs. Its modular design utilizes gem5's checkpointing features to ensure that faults only affect the program being studied. Additional functionality has been incorporated into gem5's checkpointing mechanism to preserve both microarchitectural and architectural states, allowing the study of long-running workloads without long warm-up periods and enabling analysis from any desired time point while maintaining the accurate microarchitectural state, including cache data.

Overall, gem5-MARVEL is a powerful and flexible fault injector built upon the gem5 simulator, designed to accurately evaluate and analyze the impact of faults in modern systems. Its optimizations aim to speed up fault injection campaigns and improve the reliability evaluation process.

C. Fault Injection Campaign

As shown in Figure 2, a fault injection campaign in gem5-MARVEL consists of injecting a series of faults (statistical sample) into simulations. These simulations produce output results, which are parsed to estimate the vulnerability or other desired metrics. Depending on system parameters, a gem5-MARVEL component generates fault mask files ❶. Running scripts serve as a campaign controller and manage fault injection campaigns, composing the gem5-MARVEL library. Each fault injection simulation requires a fault mask input file ❷. The simulations are run ❸, with the controller supplying necessary inputs and storing the results. Multiple systems/CPU cores can be used for faster assessment. gem5-MARVEL can configure multiple workers to achieve 100% hardware resource utilization. Each gem5-MARVEL instance corresponds to one injected fault and produces output files and logs ❹. Simulation outputs are stored for post-processing, determining the effect of the injected fault. Results are parsed ❺, and finally, the AVF or HVF estimation is exposed ❻.

1) Hardware Configurations: To perform a vulnerability assessment campaign using gem5-MARVEL, a hardware configuration is required. This configuration includes gem5 system scripts, instructions for running fault-free and fault injection simulations, and hardware structure information for generating appropriate faults within benchmark execution bounds.

2) Workloads & Accelerator Designs: gem5-MARVEL also maintains a pool of pre-configured workloads, similar to the hardware presets, using the functionality of gem5. There are checkpoints at the beginning of a workload which are used by gem5-MARVEL exactly like gem5 does. The requirements for the workload presets are to have a checkpoint at the beginning of the benchmark, switch to emulation at the end of the benchmark, export the output to the host during emulation, and terminate the simulation. These actions are performed

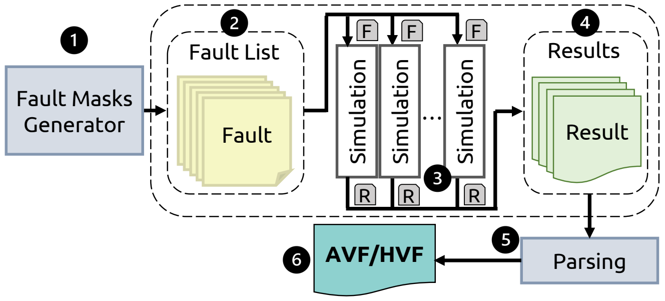

Fig. 2. Fault injection campaign high-level layout with parallel workers.

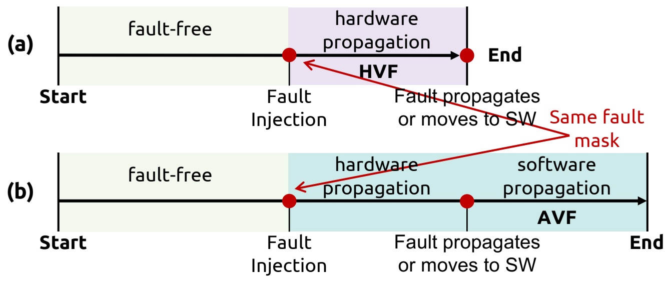

Fig. 3. (a) HVF evaluation process; (b) AVF evaluation process, which may also consider the HVF evaluation.

using directives (i.e., magic instructions of gem5) given either from the host or the simulated workload.

D. Measuring AVF and HVF

gem5-MARVEL uses two vulnerability evaluation methodologies: HVF assessment [37] and AVF assessment [76]. As shown in Figure 3(a), HVF assesses the effects of hardware faults until they first impact the software layer and stops there. It categorizes faults as Benign (masked by microarchitectural operations) or Corruptions (architecturally visible at the commit stage). AVF (see Figure 3(b)), on the other hand, provides a cross-layer vulnerability assessment, considering the entire program's execution. Another essential contribution of gem5-MARVEL is that it can perform HVF and AVF analysis either on the same or separate runs. Additionally, since the fault masks should be the same between different evaluation methodologies (i.e., HVF or AVF), gem5-MARVEL can also provide in-depth information about the propagation path of a specific fault (see Figure 3(b)). *To the best of our knowledge, there is no other framework that can provide such a fine-grained correlation of faults, enabling numerous explorations*. For accelerator designs targeting scratchpad memories, or register banks, HVF and AVF analyses are identical, since any fault is visible unless it hits an invalid or unused cell, which is then considered masked.

E. Target Hardware Structures

gem5-MARVEL supports various CPU microarchitectural components as fault-injection targets, including integer and floating-point physical register files, memory cache levels, load and store queues, reorder buffer, TLBs, register renaming unit, etc. Due to space constraints, this paper focuses on five major CPU structures for fault injection: (1) Integer Physical Register File, (2) L1 Instruction Cache, (3) L1 Data Cache, (4) Load Queue, and (5) Store Queue. For DSA designs, fault injection results are reported for scratchpad memories and register banks, since these are the available DSA hardware structures. Scratchpad memories are high-speed internal memories located close to accelerator functional units (see Figure 1; DSAs consist of functional units). They serve as temporary storage for ongoing calculations and data manipulation, tailored to the specific needs of the accelerator design. In our accelerator configurations, scratchpad memories play a crucial role in handling input, output, and intermediate data for accelerated algorithms. Input data are transferred from the CPU to accelerators via DMA, and the results are DMA'd back to the CPU after processing. Register banks fulfill a comparable role to SPMs but function as slower and less complex components, showing a delta delay between the moment a register is written to and when the data becomes available for read operations.

F. Sanity Checking of the Proposed Fault Injector

To validate gem5-MARVEL fault injection, specific test programs were developed for each individual microarchitectural component that gem5-MARVEL supports. We present the program used to validate the fault injection on the L1 Data cache; similar approaches were followed for all other hardware structures (details omitted due to limited space). As shown in Listing 1, the validation program incorporates inline assembly (i.e., keep variables and iterators in registers instead of spilling to the memory) and compiler directives to prevent interference from compiler optimizations and memory alignment (lines 1-8). All test programs are compiled using -O0 GCC flag, which instructs the compiler to avoid any code optimizations. The validation test program starts by creating an array aligned to the cache-line size, equal to the L1 data cache size (lines 7-8). Through 10 iterations [2], the array is filled with zeros, ensuring the data cache is entirely zero-filled (lines 13-15). A checkpoint using the gem5's pseudo-instruction `m5_checkpoint()` [86] is taken to capture the system's architectural and microarchitectural state, including cache contents (see Section IV-B).

This pseudo-instruction indicates the beginning of the fault injection process (line 17). During the injection window, a `nop` loop runs to avoid disturbing cache contents (lines 18-19), while gem5-MARVEL injects faults at the L1 Data Cache, as described in Section IV-C. The program switches to emulation using `m5_switch_cpu()` pseudo-instruction, indicating the end of the fault injection process (line 21). In a fault-injection free window (lines 23-25), cache contents are checked by summing all words in the array, where *a non-zero sum indicates a correctly injected fault*. Faults are uniformly distributed across the entire data array during a 10,000 fault injection campaign to target all cache lines.

[2]Before running any cache test, we need to warm up the cache by iteratively accessing the desired data multiple times. This process ensures that all cache ways are fully filled due to pseudo-LRU replacement policy [83]–[85].

```
#define CSIZE 4096
register uint32_t  i     asm ("r28");
register uint32_t  j     asm ("r27");
register uint64_t  *arr  asm ("r26");
register uint64_t  sum   asm ("r25");

uint64_t  __attribute__((section (".myArrSec")))
   array[CSIZE] __attribute__ ((aligned (64)));

int main(void) {
    arr = array;
    sum = 0;
    for (j = 0; j < 10; j++)
        for (i = 0; i < CSIZE; i++)
            arr[i] = (uint64_t)0x00;

    m5_checkpoint();  // Start injection here
    for (i = 0; i < 10000; i++)
        asm volatile ("nop");
    m5_switch_cpu();  // End injection here

    for (i = 0; i < CSIZE; i++)
        sum += arr[i];
    printf("%X", sum);
    m5_exit();    // Simulation ends here
}
```

Listing 1. L1 data cache validation test program for Arm ISA.

The measured AVF is 100%, indicating that gem5-MARVEL correctly measures the vulnerability of the entire L1 data cache contents (100% coverage).

V. Experiments — Case Studies — Insights

In this section, we present results from multiple case studies conducted on gem5-MARVEL to demonstrate its effectiveness and key insights. The objective is to showcase various features and capabilities of the framework rather than exhaustively analyzing all reliability aspects for each CPU, workload, ISA, or DSA. gem5-MARVEL allows fault injection in either a single component or multiple components, and the aggregated results can provide a comprehensive SoC AVF report (see Section IV-A1). However, for clarity, we present the results separately to offer distinct insights for each case.

A. Weighted AVF

To provide a comprehensive summary of detailed data and results for each hardware structure, and to enable meaningful comparisons among different CPU and accelerator designs, we adopt an aggregated AVF-based metric. Considering the different execution times of the benchmarks, we use a weighted approach for calculating the AVFs, similar to [73], [77]. This weighting naturally leads to a smaller impact on to aggregate AVF of hardware structures coming from benchmarks with shorter execution times compared to those with longer execution times. The weighted AVF is obtained by summing the AVFs of all benchmarks, with each AVF multiplied by the corresponding benchmark's execution time (the weight), and then dividing by the sum of execution times of all benchmarks, as shown below:

$$wAVF(c) = \sum_{k=1}^{N}(AVF_k(c) \times t_k) \; / \sum_{k=1}^{N} t_k$$

where, *wAVF(c)* is the weighted AVF of a component *c*, $AVF_k(c)$ is the AVF of component *c* for benchmark *k*, t_k is the execution time of each benchmark *k*, and *N* is the total number of benchmarks. The weighted AVF (wAVF) is shown for each hardware structure at the rightmost bars of each figure.

B. CPUs Vulnerability Evaluation — the 3 ISAs

In this subsection, we present AVF fault injection campaigns, targeting the major microarchitectural components of an OoO microprocessor model and for the three different 64-bit ISAs. Note that the insights we discuss below about the relative vulnerabilities of the ISAs are only applicable to the specific workloads and microarchitectural configuration we employed (which is the same across the three ISAs; as discussed in Section III-D and Table II of the paper). Our results do not imply that one ISA is generally more robust (i.e., less vulnerable) than another. Naturally, there is no guarantee that the rank ordering of the ISAs would not change under a different microarchitectural configuration or workload suite. Such an investigation would require an extensive study involving numerous microarchitectural configurations and workloads and is not the purpose of this paper. Instead, we present a baseline microarchitecture to demonstrate the potential of gem5-MARVEL, along with initial observations and implications specific to the case studies outlined in this paper.

1) Physical Register File: Figure 4 presents the AVF results for the Integer Physical Register File (RF) across fifteen benchmarks of the MiBench [72] suite for the three prevailing ISAs (Arm, x86, RISC-V). At the right of the graph, we show the weighted AVF (wAVF) for each ISA that aggregates the individual AVF results of different benchmarks. As shown in Figure 4, the AVF varies from 6% to 14% for Arm, 4.7% to 13.2% for x86 and 5.1% to 20.8% for RISC-V. The AVF strongly depends on the program being executed, indicating a high sensitivity to the access pattern and the utilization of the register file. For most benchmarks, the RISC-V ISA has a consistently higher AVF compared to both Arm and x86, with notable outliers being the *dijkstra*, *edges* and *corners*.

Observation #1: Transient fault vulnerability of the physical register file is significantly higher in RISC-V than in Arm and x86 for the considered microarchitectural configuration and workloads.

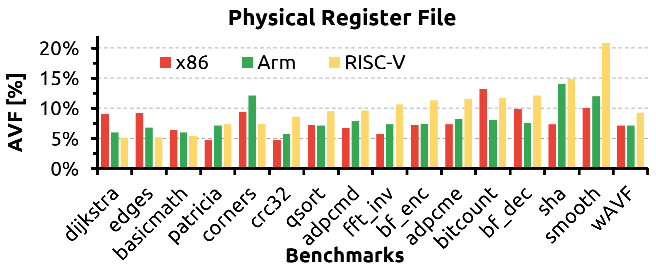

Fig. 4. AVF for the physical register file for Arm, x86 and RISC-V ISAs.

Architectural Implication #1: Although the source code workloads are the same among the three ISAs, the different binaries may influence how transient faults in the register file contents affect program execution. Some workloads are more likely to trigger specific access patterns of the register file that could exacerbate vulnerabilities in a particular ISA (RISC-V in this case) rather than in other ISAs.

2) L1 Instruction Cache: Figure 5 presents the AVF results of the L1 Instruction Cache across the fifteen benchmarks for all three ISAs. Unlike the register file (Figure 4), the L1 instruction cache presents more uniform distribution across the benchmarks, ranging from 20.5% to 37.9% for Arm, 25.2% to 38.2% for x86 and 16.4% to 34.9% for RISC-V. As shown in Figure 4, the Arm ISA exhibits the highest vulnerability (i.e., high AVF) across most benchmarks, while RISC-V has the lowest AVF compared to Arm and x86. However, the relatively lower variance of the AVF in the L1 instruction cache compared to the RF indicates that the access pattern of the instruction cache for all ISAs is less dependent on the specific benchmark being tested. The smaller AVF of the RISC-V ISA might be related to the simpler encoding logic of RISC-V instructions, resulting in higher masking effects than in other ISAs.

Observation #2: The transient fault vulnerability of the L1 Instruction Cache for the RISC-V ISA is the lowest among the three ISAs for the considered microarchitectural configuration and workloads.

Architectural Implication #2: The RISC-V ISA's simplicity and modularity, along with its efficient decoding logic, may result in a reduced probability of faults during instruction fetch and decode stages. Additionally, compiler optimizations may influence the vulnerability of the instruction cache (even if all binaries are compiled with the same GCC compiler version for all ISAs), considering different ISAs may have varied compiler strategies and code generation quality.

3) L1 Data Cache: Figure 6 presents the AVF results of the L1 Data Cache across fifteen benchmarks for all three ISAs. The AVF for this hardware structure ranges between 4.3% to 44.9% for Arm, 3.4% to 35.1% for x86 and 5.9% to 40.9% for RISC-V. The L1 Data Cache AVF has the largest variance of all three components. This can be attributed to the fact that the L1 data cache access pattern and utilization are highly dependent on the benchmark being executed. x86 has the lowest wAVF, with Arm and RISC-V being quite close.

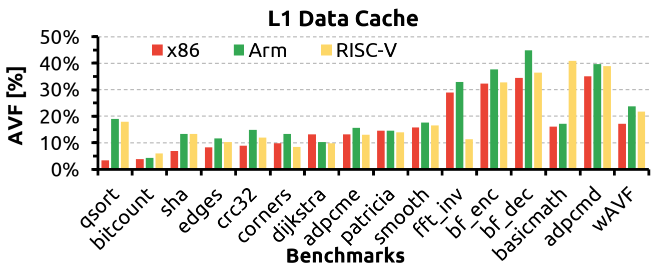

Fig. 6. AVF for the L1 data cache for Arm, x86 and RISC-V ISAs.

Observation #3: The Arm and RISC-V ISAs exhibit the highest vulnerability to transient faults compared to the x86 ISA for the L1 data cache for the considered microarchitectural configuration and workloads.

Architectural Implication #3: Data cache requests may be influenced by various factors such as memory operations within the program's execution path, load instructions that were predicted incorrectly, prefetching, and load/store dependencies that are yet to be resolved. The x86 ISA typically has more complex instructions that perform multiple simpler operations in a single instruction. This complexity may lead to more intricate cache access patterns compared to Arm and RISC-V ISAs, increasing the hardware masking probability, and thus, making them less vulnerable to transient faults.

4) Load Queue: Figure 7 presents the AVF results of the Load Queue across fifteen benchmarks for all three ISAs. The AVF for this hardware structure ranges between 2.4% to 8.6% for Arm, 2.7% to 11.1% for x86 and 3.5% to 12.9% for RISC-V. The benchmark with the highest AVF averaged across all three ISAs is *smooth* and the benchmark with the lowest AVF is *adpcme*. We observe significant variability among benchmarks and ISAs; however, benchmark-dependent trends remain consistent across all three ISAs.

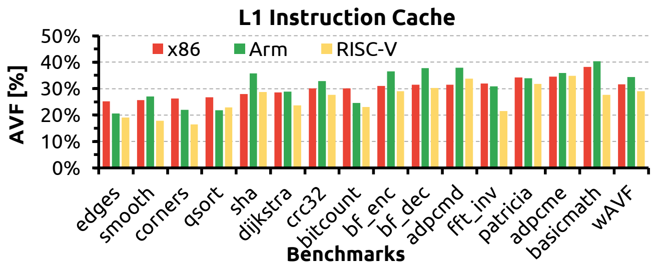

Fig. 5. AVF for the L1 instruction cache for Arm, x86 and RISC-V ISAs.

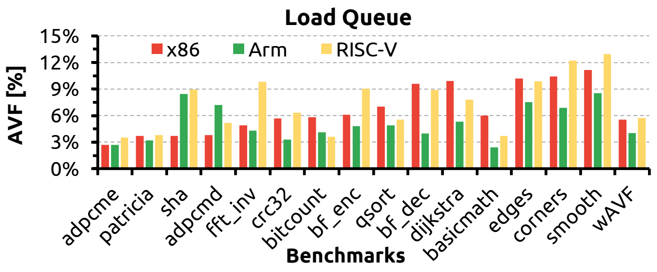

Fig. 7. AVF for the Load Queue for Arm, x86 and RISC-V ISAs.

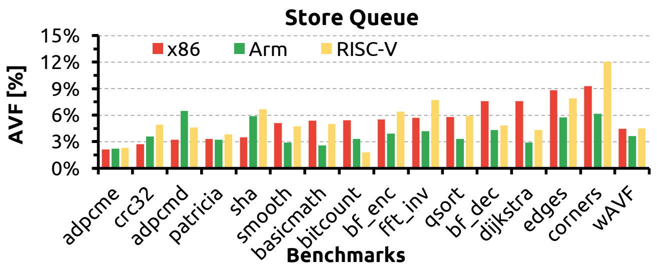

Fig. 8. AVF for the Store Queue for Arm, x86 and RISC-V ISAs.

5) Store Queue: Figure 8 presents the AVF results of the Store Queue across fifteen benchmarks for all three ISAs. The AVF for this hardware structure ranges between 2.2% to 6.2% for Arm, 2.1% to 9.3% for x86 and 1.8% to 12.0% for RISC-V. The benchmark with the highest AVF averaged across all three ISAs is *corners* and the benchmark with the lowest AVF is *adpcme*. We observe significant variability between benchmarks and ISAs; however, benchmark-dependent trends remain consistent across all three ISAs.

Observation #4: Transient fault vulnerability of the load and store queues is lower in Arm compared to RISC-V and x86 for the considered microarchitectural configuration and workloads.

Architectural Implication #4: The ISA memory ordering model defines load and store interactions with memory and each other. Variations in memory models may affect the number of in-flight or queued load and store instructions. Therefore, Arm's memory ordering model may positively impact the load and store queue's resilience to transient faults.

Considering the observations mentioned above, there are instances about specific microarchitectural components where one ISA appears less vulnerable than others. In contrast, in other scenarios, the same ISA may appear more vulnerable. This outcome can be explained in two ways. Firstly, each ISA can be affected differently by the specific microarchitectural configuration. Some ISAs might exhibit superior optimization or design that minimizes the impact of transient faults under certain configurations, yet render them more susceptible under different setups. Secondly, diverse workloads impose varying stress (usage) levels on different parts of the microarchitecture. Certain ISAs may excel under particular workloads owing to their architecture's handling of specific operations or data accesses, thus resulting in different vulnerabilities across scenarios.

C. SDC Contribution to AVF

In this subsection, we present the contribution of Silent Data Corruptions (SDCs) to the overall AVF, considering three major microarchitectural components of an OoO core. Note that the right-most columns of each graph present the SDC weighted AVF (wAVF) for each ISA that aggregates the individual SDC AVF results of all benchmarks.

1) Physical Register File: Figure 9 presents the SDC AVF results for the Integer Physical Register File (RF) across fifteen benchmarks of the MiBench [72] suite for the three prevailing ISAs (Arm, x86, RISC-V). As shown in Figure 9, the AVF varies from 0% to 6.9% for Arm, 0% to 3.7% for x86 and 0.1% to 9.9% for RISC-V. Compared to the overall weighted AVF, the SDC wAVF is 4.6 times lower for Arm, 5 times lower for x86, and 4 times lower for RISC-V. Therefore, the SDC probability for the RF is significantly less than the Crash probability, which is the remaining fault effect that composes the overall AVF.

2) L1 Instruction Cache: Figure 10 presents the SDC AVF results for the L1 Instruction Cache across fifteen benchmarks of the MiBench [72] suite for the three ISAs (Arm, x86, RISC-V). As shown in Figure 10, the AVF varies from 0.3% to 9.9% for Arm, 0.3% to 4.6% for x86 and 0.2% to 5.7% for RISC-V. Compared to the overall weighted AVF, the SDC weighted AVF is 9 times lower for Arm, 11.2 times lower for x86, and 17 times lower for RISC-V. Therefore, the SDC probability for the L1 Instruction Cache is extremely low compared to the Crash probability.

3) L1 Data Cache: Figure 11 presents the SDC AVF results for the L1 Data Cache across fifteen benchmarks of the MiBench [72] suite for the three ISAs (Arm, x86, RISC-V). As shown in Figure 11, the AVF varies from 1.2% to 43% for Arm, 0.7% to 32.6% for x86 and 0.8% to 40.2% for RISC-V. Compared to the overall weighted AVF, the SDC weighted AVF is 20.6% vs 23.8% for Arm, 13.7% vs 17.1% for x86, and 17.8% vs 21.7% for RISC-V. Therefore, in contrast to

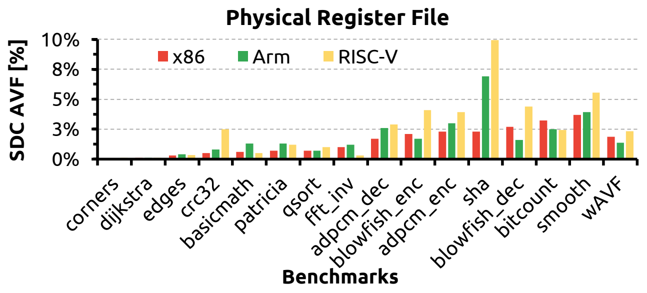

Fig. 9. SDC AVF for the physical register file.

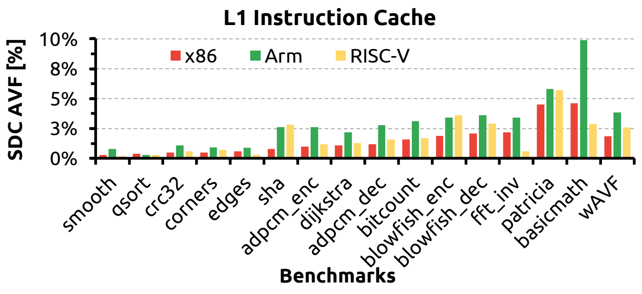

Fig. 10. SDC AVF for the L1 instruction cache.

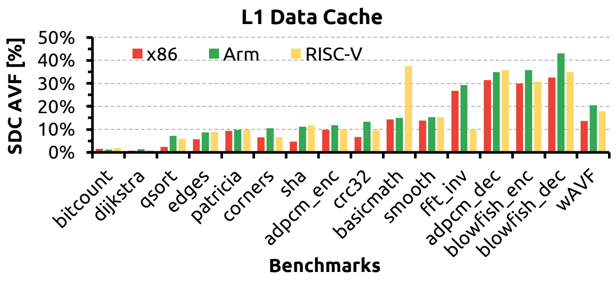

Fig. 11. SDC AVF for the L1 data cache.

other hardware structures, the SDC probability for the L1 Data Cache is extremely high compared to the Crash probability.

Observation #5: SDCs are much rarer in the Physical Integer Register File and L1 Instruction Cache than in the L1 Data Cache, where they are the dominant fault effect.

Architectural Implication #5: Wrong values in registers are very likely to result in illegal memory accesses, and corrupted blocks in the L1 Instruction cache will most likely result in an illegal instruction being executed. Data cache corruptions, on the other hand, are less likely to cause a crash and can easily propagate to the program output resulting in an SDC.

D. SDCs due to Permanent Faults

Permanent faults, also known as hard errors, are long-lasting or permanent defects in the CPU's hardware structures, such as SRAM arrays. These faults typically result from manufacturing defects, physical damage (e.g., from overheating or electrical overloads), or wear and tear over time. Permanent faults are persistent and usually require hardware repair or replacement to resolve. Permanent faults may have serious consequences for CPU operation, potentially resulting in system instability, data corruption, or even severe system failures, depending on their severity and frequency. In this subsection, we present the probability of permanent faults generating SDCs for the three ISAs of this study, for two major microarchitectural components of an OoO core (i.e., the L1 Instruction Cache, and the L1 Data Cache).

1) L1 Instruction Cache: Figure 12 presents the SDC probability results for permanent faults in the L1 Instruction Cache across fifteen benchmarks of the MiBench [72] suite for the three ISAs (Arm, x86, RISC-V). As shown in Figure 12, the SDC probability varies from 0.1% to 2.3% for Arm, 0.1% to 1.3% for x86 and 0.3% to 2.7% for RISC-V. On average for all benchmarks, the x86 ISA shows the lowest SDC probability among the ISAs studied in this paper (i.e., most benchmarks have the lowest SDC probability), while in most of the benchmarks we can see that the RISC-V ISA shows the highest SDC probability.

2) L1 Data Cache: Figure 13 shows the SDC probability results for permanent faults in the L1 Data Cache across fifteen benchmarks of the MiBench [72] suite for the three ISAs (Arm, x86, RISC-V). As shown in Figure 13, the probability varies from 5.1% to 53.3% for Arm, 4.4% to 64.7% for x86 and 4.4% to 70.8% for RISC-V. On average for all benchmarks, the RISC-V ISA has the highest SDC probability among all ISAs studied in this paper (i.e., most benchmarks have the highest SDC probability).

Overall, for the microarchitecture and workloads we analyzed, the RISC-V ISA exhibits a significantly higher probability of SDCs due to permanent faults compared to the other ISAs, namely Arm and x86.

E. Domain-Specific Accelerators Vulnerability Evaluation

In this subsection, we report the AVF results from fault injection on eight different DSA designs, targeting their large on-chip SRAMs: scratchpad memories (SPMs) and register banks (RegBanks). These components store input, output data, and intermediate results of accelerated algorithms. For each DSA, we select representative SPMs and RegBanks for independent fault injection campaigns to assess their AVF, as shown in Table IV. Figure 14 presents the breakdown of SDC and Crash fault effects, which together constitute the complete AVF, for all designs.

The *BFS* DSA design uses two distinct RegBanks for accessing the EDGES and the NODES of the input graph, and does not use any SPMs. The AVF of the EDGES RegBank is 35%, while the AVF of the NODES RegBank is 20%, and as shown in Figure 14, nearly all fault effects of *BFS* are Crashes. This is due to data from both RegBanks being used as indices for graph traversals by the accelerator hardware. As a result, faults in any RegBank lead to either excessively long execution times or out-of-bounds memory accesses that surpass the size of the system's physical memory. The *FFT* design utilizes two SPMs to store the imaginary (IMG) and REAL components of the algorithm's output. The IMG SPM has an AVF of 44.5%, while the REAL SPM has an AVF of 45.1%. These AVFs are quite similar because a fault in either the imaginary or real part of the *FFT* result has an equal probability of corrupting the accelerator output. Interestingly, as shown in Figure 14, all faulty runs result in SDCs, since the SPM data is not utilized by any accelerator control logic or used as indices for memory accesses. The same pattern is also observed in the *GEMM* and *MERGESORT* designs. GEMM holds the input data of one of the matrices to be multiplied in one SPM and the result of the matrix multiplication in another SPM, while MERGESORT uses two SPMs to store the main array data and temporary

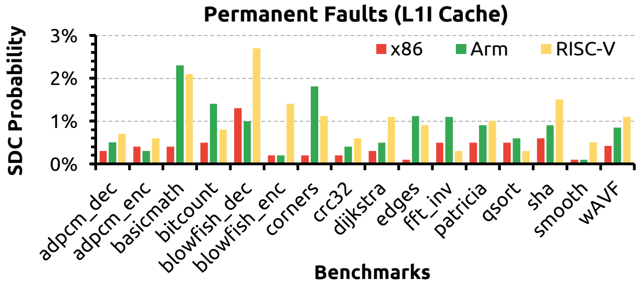

Fig. 12. SDC AVF for permanent faults of the L1 instruction cache.

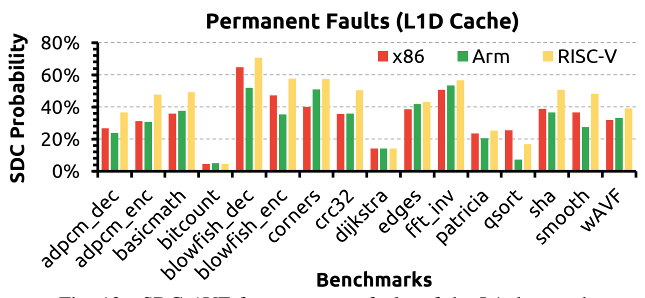

Fig. 13. SDC AVF for permanent faults of the L1 data cache.

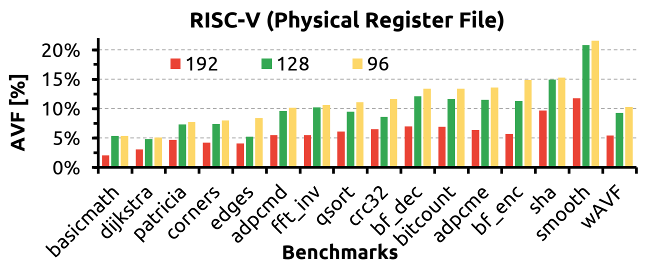

Fig. 15. AVF for the physical register file of the RISC-V CPU, with different numbers of physical registers (i.e., 96, 128, and 192).

TABLE IV
TARGET INJECTION COMPONENTS FOR EACH DSA DESIGN.

Accelerator	Component	Memory Size (Bytes)	Memory Type
BFS	EDGES	16,384	RegBank
	NODES	2,048	RegBank
FFT	IMG	8,192	SPM
	REAL	8,192	SPM
GEMM	MATRIX1	32,768	SPM
	MATRIX3	32,768	SPM
MD_KNN	NLADDR	16,384	SPM
	FORCEX	2,048	SPM
MERGESORT	MAIN	8,192	SPM
	TEMP	8,192	SPM
SPMV	VAL	13,328	SPM
	COLS	6,664	SPM
STENCIL2D	ORIG	32,768	SPM
	SOL	32,768	SPM
	FILTER	360	RegBank
STENCIL3D	ORIG	65,536	SPM
	SOL	65,536	SPM
	C_VAR	8	RegBank

intermediate values. As shown in Figure 14, the output SPM (MATRIX3) of *GEMM* has significantly lower AVF than the input SPM. This can be attributed to the injected faults in the output SPM being overwritten much more often because the input SPM gets written to only once by the DMA device when the accelerator is initialized, whereas the output SPM gets written to for the entire accelerator runtime. For *MERGESORT*, the TEMP SPM has significantly lower AVF than in MAIN SPM, which can be attributed to the overwriting of numerous faults due to the continuous stream of memory writes to the SPM. Similar observations hold also for the remaining DSA designs, which are omitted due to space limitations.

Observation #6: Most accelerator designs result in very high SDC rates in the presence of faults.

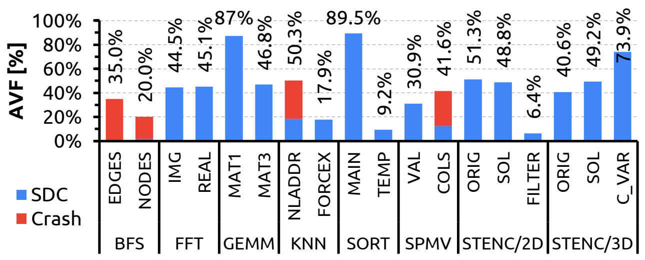

Fig. 14. AVF for different accelerator designs with different injection targets.

Architectural Implication #6: Most accelerators are datapath-heavy, with few control dependencies on their input data. This characteristic enables accelerators to efficiently execute data-intensive tasks by prioritizing parallel processing of large data volumes. However, it also amplifies the probability of SDCs in the presence of transient faults. Therefore, fault mitigation strategies for accelerators should primarily focus on data corruptions and not the control flow.

F. Sensitivity Analysis: Registers Number and Vulnerability

Figure 15 presents the AVF for all fifteen benchmarks used in this study, running for the RISC-V ISA with different Physical Register File sizes (i.e, 96, 128 and 192 physical registers) as a sensitivity case study. As shown in Figure 15, the vulnerability of the physical register file increases as the number of physical registers is reduced due to the increased utilization of each register. With fewer physical registers, each register is accessed and updated more frequently, increasing the chances of a fault occurring during register access. Additionally, with fewer physical registers available, the microprocessor will reuse a register for multiple purposes, which can increase the probability of a fault occurring due to an unexpected value in the register. Therefore, reducing the number of physical registers in a microprocessor can increase the vulnerability of the physical register file to transient faults. This is a consistent observation across the three ISAs.

G. Performance-Aware Comparison

We explore how different computing systems can be fairly compared using gem5-MARVEL regarding reliability, and show how vulnerability measurements can be combined with system performance. We showcase our methodology with a test case scenario by comparing the reliability of two different computing systems: a standalone RISC-V CPU and a standalone DSA. For a fair comparison, 4 particular algorithms are properly implemented to run and are modeled in both computing systems. These are a Matrix Multiplication Algorithm (i.e., GEMM), BFS, FFT, and KNN algorithms (as described in Table IV). AVF is a pure reliability metric that *does not provide any information about the system performance*. AVF alone cannot provide any insights on the tradeoff between performance and reliability of a chip. To this end, gem5-MARVEL is also able to compute a new simple reliability metric named *Operations per Failure (OPF)*. OPF is the number of times a workload

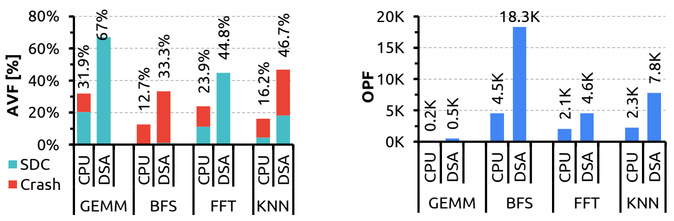

Fig. 16. Breakdown of SDC and Crash AVF of 4 algorithms for both CPU and accelerator (left graph), and the OPF for CPU and accelerator (right graph).

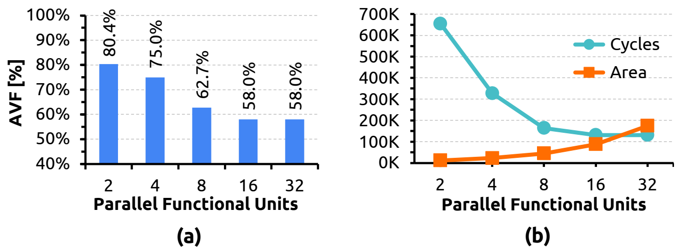

Fig. 17. Gemm accelerator design for five different configurations of available functional units, showing (a) the AVF for different functional units, and (b) the performance and area statistics.

is executed before a system failure happens, and is computed using the following formula: $OPF = OPS/AVF$, where OPS (Operations per Second) is the number of operations (i.e., tasks) that the compute unit can perform during 1 second. Assume, for example, the Matrix Multiplication algorithm, which performs $2 \times N^3$ operations, where N is the size of the matrices. Thus, $OPS = 2 \times N^3/Exec\_Time$. The OPF metric enables a combined analysis of performance and reliability into a single metric. For the same workload that runs in different platforms (a CPU or an accelerator in our example), *larger OPF values indicate better tradeoff between reliability and performance (larger number of correct executions over time).*

Figure 16 showcases the pure reliability evaluations against the new proposed metric for the 4 algorithms, which takes into account the performance of the platform and presents the tradeoff between performance and reliability in a single metric. As Figure 16 demonstrates, while the AVF (left graph) shows that all 4 algorithms running on the accelerator (the DSA labels in the x-axis) are significantly more vulnerable than running on a RISC-V CPU, the combined vulnerability and performance metric OPF (right graph) shows that the same algorithms can be executed in the accelerator significantly more times than in the CPU before observing a system failure (i.e., better tradeoff between performance and reliability for the accelerator design).

Observation #7: Although the accelerator design is more vulnerable, it demonstrates better tradeoff between performance and reliability.

Architectural Implication #7: The higher OPF value suggests that the accelerator design offers increased resilience and can maintain stable operation for a more extended period, making it more suitable for executing the algorithm in real-world scenarios where reliability is crucial. Although the AVF highlights its higher vulnerability, the advantages in OPF emphasize the benefits of using the accelerator design to achieve improved performance while maintaining an acceptable level of reliability.

H. Accelerator Design Space Exploration

gem5-MARVEL models data-dependent control accelerator execution by independently evaluating static and dynamic elements, providing more configuration options for design space exploration. Users can specify constraints on hardware resources to enforce functional unit reuse. The hardware resource model is generated dynamically from YAML configuration files (an inherent feature of the gem5-SALAM), enabling the creation of hardware profiles at the granularity of individual accelerators within the cluster. This allows users to redefine parameters, including customized instructions and functional units, without recompiling or rebuilding the system [36]. The proposed gem5-MARVEL injection framework leverages the features from both gem5 and gem5-SALAM to explore the accelerator design space from the reliability perspective. As a case study, we show the reliability evaluation of the MachSuite [82] GEMM accelerator for different degrees of parallel processing, i.e., amount of parallel functional units. Figure 17(a) shows the AVF of MATRIX1 SPM (holds the data of one of the input matrices) for the different accelerator configurations. We can see that as the parallel functional units are reduced, the AVF increases significantly. This can be relatively safely attributed to the slower SPM access rate that allows more faults to propagate to the output without being masked. Note that the data occupy the entire SPM from the beginning of processing. Along the same lines, in Figure 17(b) we can see the differences of performance and area for the GEMM accelerator design. It is clear from these graphs that based on the outcomes of gem5-MARVEL we can find the optimal tradeoff between reliability, area, and performance.

Observation #8: The AVF increases significantly when the number of parallel functional units in DSAs is reduced.

Architectural Implication #8: With fewer parallel functional units, faults have a higher chance of affecting the output data due to the data occupying the entire SPM from the beginning of processing. This implies that designs with fewer parallel functional units may be more susceptible to transient faults, potentially leading to incorrect results during computation. To enhance fault resilience, future designs could consider optimizing the access rates to the SPM and implementing suitable fault-tolerance mechanisms.

I. HVF Results

Figure 18 shows the HVF and AVF results for six benchmarks targeting the physical register file and L1 data cache. As shown in Figure 18, the corruptions measured by HVF analysis (the yellow bars in Figure 18) are constantly higher than the AVF measurements (i.e., the blue bars). The reason is that HVF calculates any corruption that could not get masked at the hardware layer and eventually reaches the software layer. After that point, the corruption may or may not get masked at the software layer. In case that it gets masked at the software layer, it counts and "masked" in the AVF measurement, otherwise, it counts as an SDC or a Crash, depending on its effect. Thus, by definition, the HVF measurements will always be greater than the AVF measurements.

VI. Resilience Assessment of the Heterogeneous Architecture

Naturally, if a different microarchitecture is selected for each ISA (to resemble another CPU chip) the results and part of the final observations of the previous section may be different, since it is well known that a CPU's vulnerability to faults depends on all the following: ISA, microarchitecture, and workload. Our results serve as a case study, primarily demonstrating the capabilities of gem5-MARVEL. For a head-to-head comparison of a particular x86-ISA CPU, an Arm-ISA CPU, and a RISC-V ISA CPU, gem5-MARVEL can be configured to reflect each one's microarchitecture. gem5-MARVEL is a state-of-the-art microarchitecture-level fault injection framework that targets heterogeneous SoC architectures, including both CPUs of different ISAs and DSAs. Such a framework can provide numerous invaluable insights into the overall systems' resilience. Since space limitations do not allow us to present exhaustively all the in-depth observations and implications, we group the fundamental insights that gem5-MARVEL can deliver:

1) **CPU ISA comparison:** By injecting faults into CPUs with different ISAs, gem5-MARVEL can compare fully unprotected designs or any error detection or correction mechanisms at the software or hardware layer. This analysis reveals which ISA performs better under fault conditions and which requires stronger protection.
2) **Accelerator impact:** Fault injection in DSAs helps understand how resilient they are to different fault scenarios, which is critical in several application domains.
3) **System-level resilience:** gem5-MARVEL allows analyzing the overall system's resilience. By injecting faults at different locations in the system, it can be observed how the CPUs and accelerators interact and recover from faults. This knowledge is valuable for designing next-generation fault-tolerant systems at scale.
4) **Error recovery mechanisms:** gem5-MARVEL can shed light on the effectiveness of error protection (detection and correction) mechanisms in both the CPUs and accelerators. This insight can guide improvements in the system's error handling and fault recovery strategies.
5) **Power and performance trade-offs:** gem5-MARVEL can help the concurrent assessment of the complex tradeoffs among power consumption, performance, and resilience. It provides insights into how protection mechanisms can impact the performance and power efficiency.

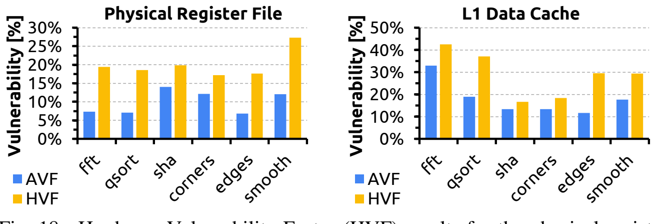

Fig. 18. Hardware Vulnerability Factor (HVF) results for the physical register file and the L1 data cache.

VII. Conclusion

We presented and extensively demonstrated the features of gem5-MARVEL, the first consolidated microarchitecture-level fault injection infrastructure for heterogeneous SoC architectures consisting of CPUs with different ISAs and domain-specific accelerators. Such an infrastructure is an important step forward in measuring the vulnerability of complex computing systems to hardware faults and guiding protection methods. We showcased that gem5-MARVEL offers a flexible and extensible way of simulating a wide range of faults, automating the fault injection process, and analyzed the results. The proposed gem5-MARVEL infrastructure fills a gap in the existing microarchitecture-level fault injection frameworks by offering a complete framework that can be adapted to a wide range of computing systems architectures. We have presented a subset of the plethora of case studies that can be based on gem5-MARVEL; such case studies can deliver deep insights on system properties that can be vulnerability-only or can combine vulnerability and performance.

Acknowledgements

We thank the anonymous HPCA reviewers for their insightful comments and feedback. This work is supported by the European Union's Horizon Europe research and innovation programme under grant agreements No 101070238 (NEUROPULS) and No 101097224 (REBECCA), and by research gifts from AMD and Meta. Views and opinions expressed are those of the authors only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, p. 48–60, jan 2019. [Online]. Available: https://doi.org/10.1145/3282307

[2] J. Cong, V. Sarkar, G. Reinman, and A. Bui, "Customizable domain-specific computing," *IEEE Design & Test of Computers*, vol. 28, no. 2, pp. 6–15, 2011.

[3] Z. Jia, M. Maggioni, J. Smith, and D. P. Scarpazza, "Dissecting the nvidia turing t4 gpu via microbenchmarking," 2019. [Online]. Available: https://arxiv.org/abs/1903.07486

[4] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*, 2016, p. 243–254. [Online]. Available: https://doi.org/10.1109/ISCA.2016.30

[5] Y. Turakhia, G. Bejerano, and W. J. Dally, “Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly,” in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’18)*, 2018, p. 199–213. [Online]. Available: https://doi.org/10.1145/3173162.3173193

[6] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, “Convolution engine: Balancing efficiency & flexibility in specialized computing,” in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA ’13)*, 2013, p. 24–35. [Online]. Available: https://doi.org/10.1145/2485922.2485925

[7] P. Agrawal and W. Dally, “A hardware logic simulation system,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 1, pp. 19–29, 1990.

[8] G. Moore, “Cramming more components onto integrated circuits,” *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[9] C. Constantinescu, “Impact of deep submicron technology on dependability of vlsi circuits,” in *Proceedings International Conference on Dependable Systems and Networks*, 2002, pp. 205–209.

[10] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, “Silent Data Corruptions at Scale,” 2021. [Online]. Available: https://arxiv.org/abs/2102.11245

[11] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, “Cores That Don’t Count,” in *Proceedings of the Workshop on Hot Topics in Operating Systems*, ser. HotOS ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 9–16. [Online]. Available: https://doi.org/10.1145/3458336.3465297

[12] A. Singh, S. Chakravarty, G. Papadimitriou, and D. Gizopoulos, “Silent data errors: Sources, detection, and modeling,” in *2023 IEEE 41st VLSI Test Symposium (VTS)*, 2023.

[13] G. Papadimitriou, D. Gizopoulos, H. Dixit Dattatraya, and S. Sankar, “Silent data corruptions: The stealthy saboteurs of digital integrity,” in *2023 IEEE 29th International Symposium on Online Testing and Robust System Design (IOLTS)*, 2023.

[14] G. Papadimitriou and D. Gizopoulos, “Silent data corruptions: Microarchitectural perspectives,” *IEEE Transactions on Computers*, vol. 72, no. 11, pp. 3072–3085, 2023.

[15] D. Gizopoulos, G. Papadimitriou, and O. Chatzopoulos, “Estimating the failures and silent errors rates of cpus across isas and microarchitectures,” in *2023 IEEE International Test Conference (ITC)*, 2023.

[16] J. Athavale, A. Baldovin, R. Graefe, M. Paulitsch, and R. Rosales, “Ai and reliability trends in safety-critical autonomous systems on ground and air,” in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2020, pp. 74–77.

[17] G. Papadimitriou and D. Gizopoulos, “Demystifying the System Vulnerability Stack: Transient Fault Effects Across the Layers,” in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA ’21)*, 2021, pp. 902–915. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00075

[18] 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.

[19] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin, “A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor,” in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, 2003, pp. 29–40.

[20] G. Papadimitriou and D. Gizopoulos, “Avgi: Microarchitecture-driven, fast and accurate vulnerability assessment,” in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 935–948.

[21] S. 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 the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 29–.

[22] G. Yalcin, O. S. Unsal, A. Cristal, and M. Valero, “Fimsim: A fault injection infrastructure for microarchitectural simulators,” in *2011 IEEE 29th International Conference on Computer Design (ICCD)*, 2011, pp. 431–432.

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

[24] 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*, 2014, pp. 622–629.

[25] A. Tyagi, Y. Gan, S. Liu, B. Yu, P. Whatmough, and Y. Zhu, “Thales: Formulating and estimating architectural vulnerability factors for dnn accelerators,” 2022.

[26] Y. He, P. Balaprakash, and Y. Li, “Fidelity: Efficient resilience analysis framework for deep learning accelerators,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 270–281.

[27] Q. Lu, M. Farahani, J. Wei, A. Thomas, and K. Pattabiraman, “Llfi: An intermediate code-level fault injection tool for hardware faults,” in *2015 IEEE International Conference on Software Quality, Reliability and Security*, 2015, pp. 11–16.

[28] U. K. Agarwal, A. Chan, and K. Pattabiraman, “Lltfi: Framework agnostic fault injection for machine learning applications (tools and artifact track),” in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, 2022, pp. 286–296.

[29] 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.

[30] A. Waterman and K. Asanovic, “The risc-v instruction set manual volume i: Unprivileged isa,” 2019, document Version 20191213, Accessed: 2023-07-25. [Online]. Available: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf

[31] A. Waterman, K. Asanovic, and J. Hauser, “The risc-v instruction set manual volume ii: Privileged architecture,” 2021, document Version 20211203, Accessed: 2023-07-25. [Online]. Available: https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf

[32] N.-J. Wessman, F. Malatesta, J. Andersson, P. Gomez, M. Masmano, V. Nicolau, J. L. Rhun, G. Cabo, F. Bas, R. Lorenzo, O. Sala, D. Trilla, and J. Abella, “De-risc: the first risc-v space-grade platform for safety-critical systems,” in *2021 IEEE Space Computing Conference (SCC)*, 2021, pp. 17–26.

[33] J. Abella, S. Alcaide, J. Anders, F. Bas, S. Becker, E. De Mulder, N. Elhamawy, F. K. Gürkaynak, H. Handschuh, C. Hernandez, M. Hutter, L. Kosmidis, I. Polian, M. Sauer, S. Wagner, and F. Regazzoni, “Security, reliability and test aspects of the risc-v ecosystem,” in *2021 IEEE European Test Symposium (ETS)*, 2021, pp. 1–10.

[34] F. Pavanello, C. Marchand, I. O’Connor, R. Orobtchouk, F. Mandorlo, X. Letartre, S. Cueff, E. I. Vatajelu, G. Di Natale, B. Cluzel, A. Coillet, B. Charbonnier, P. Noé, F. Kavan, M. Zoldak, M. Szaj, P. Bienstman, T. Van Vaerenbergh, U. Ruhrmair, P. Flores, L. G. e Silva, R. Chaves, L.-M. Silveira, M. Ceccato, D. Gizopoulos, G. Papadimitriou, V. Karakostas, A. Brando, F. J. Cazorla, R. Canal, P. Closas, A. Gusi-Amigó, P. Crovetti, A. Carpegna, T. M. Carmona, S. Di Carlo, and A. Savino, “Neuropuls: Neuromorphic energy-efficient secure accelerators based on phase change materials augmented silicon photonics,” in *2023 IEEE European Test Symposium (ETS)*, 2023, pp. 1–6.

[35] 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

[36] S. Rogers, J. Slycord, M. Baharani, and H. Tabkhi, “gem5-salam: A system architecture for llvm-based accelerator modeling,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 471–482.

[37] V. Sridharan and D. R. Kaeli, “Using Hardware Vulnerability Factors to Enhance AVF Analysis,” in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA ’10. New York, NY, USA: ACM, 2010, pp. 461–472. [Online]. Available: http://doi.acm.org/10.1145/1815961.1816023

[38] S. Mukherjee, *Architecture Design for Soft Errors*. Morgan Kaufmann, 2011. [Online]. Available: https://doi.org/10.1016/b978-0-12-369529-1.x5001-0

[39] C. Weaver, J. Emer, S. Mukherjee, and S. Reinhardt, “Techniques to reduce the soft error rate of a high-performance microprocessor,” in *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.*, 2004, pp. 264–275.

[40] K. R. Walcott, G. Humphreys, and S. Gurumurthi, "Dynamic prediction of architectural vulnerability from microarchitectural state," in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA '07)*, 2007, p. 516–527. [Online]. Available: https://doi.org/10.1145/1250662.1250726

[41] E. Cheng, S. Mirkhani, L. G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, J. A. Abraham, P. Bose, and S. Mitra, "Tolerating soft errors in processor cores using clear (cross-layer exploration for architecting resilience)," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 9, pp. 1839–1852, 2018.

[42] E. Cheng, Daniel-Mueller-Gritschneder, J. Abraham, P. Bose, A. Buyuktosunoglu, D. Chen, H. Cho, Y. Li, U. Sharif, K. Skadron, M. Stan, U. Schlichtmann, and S. Mitra, "Cross-layer resilience: Challenges, insights, and the road ahead," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3316781.3323474

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

[44] M. T. Yourst, "Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator," in *2007 IEEE International Symposium on Performance Analysis of Systems & Software*, 2007, pp. 23–34.

[45] M. Hiller, A. Jhumka, and N. Suri, "Propane: An environment for examining the propagation of errors in software," *SIGSOFT Softw. Eng. Notes*, vol. 27, no. 4, p. 81–85, jul 2002. [Online]. Available: https://doi.org/10.1145/566171.566184

[46] D. Stott, B. Floering, D. Burke, Z. Kalbarczpk, and R. Iyer, "Nftape: a framework for assessing dependability in distributed systems with lightweight fault injectors," in *Proceedings IEEE International Computer Performance and Dependability Symposium. IPDS 2000*, 2000, pp. 91–100.

[47] D. Skarin, R. Barbosa, and J. Karlsson, "Goofi-2: A tool for experimental dependability assessment," in *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, 2010, pp. 557–562.

[48] R. Maia, L. Henriques, D. Costa, and H. Madeira, "Xception/sup tm/ - enhanced automated fault-injection environment," in *Proceedings International Conference on Dependable Systems and Networks*, 2002, pp. 547–.

[49] G. Georgakoudis, I. Laguna, D. S. Nikolopoulos, and M. Schulz, "Refine: Realistic fault injection via compiler-based instrumentation for accuracy, portability and speed," in *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–14.

[50] J. Wei, A. Thomas, G. Li, and K. Pattabiraman, "Quantifying the accuracy of high-level fault injection techniques for hardware faults," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 375–382.

[51] 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*, 2013, pp. 41–50.

[52] J. Calhoun, L. Olson, and M. Snir, "Flipit: An llvm based fault injector for hpc," in *Euro-Par 2014: Parallel Processing Workshops*, L. Lopes, J. Žilinskas, A. Costan, R. G. Cascella, G. Kecskemeti, E. Jeannot, M. Cannataro, L. Ricci, S. Benkner, S. Petit, V. Scarano, J. Gracia, S. Hunold, S. L. Scott, S. Lankes, C. Lengauer, J. Carretero, J. Breitbart, and M. Alexander, Eds. Cham: Springer International Publishing, 2014, pp. 547–558.

[53] C. Giuffrida, A. Kuijsten, and A. S. Tanenbaum, "Edfi: A dependable fault injection tool for dependability benchmarking experiments," in *2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing*, 2013, pp. 31–40.

[54] M. Raiyat Aliabadi and K. Pattabiraman, "Fidl: A fault injection description language for compiler-based sfi tools," in *Computer Safety, Reliability, and Security*, A. Skavhaug, J. Guiochet, and F. Bitsch, Eds. Cham: Springer International Publishing, 2016, pp. 12–23.

[55] B. Shan, A. Shamji, J. Tian, G. Li, and D. Tao, "Lcfi: A fault injection tool for studying lossy compression error propagation in hpc programs," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 2708–2715.

[56] L. Palazzi, G. Li, B. Fang, and K. Pattabiraman, "A tale of two injectors: End-to-end comparison of ir-level and assembly-level fault injection," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, 2019, pp. 151–162.

[57] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126964

[58] A. Mahmoud, T. Tambe, T. Aloui, D. Brooks, and G.-Y. Wei, "Goldeneye: A platform for evaluating emerging numerical data formats in dnn accelerators," in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022, pp. 206–214.

[59] G. Li, K. Pattabiraman, and N. DeBardeleben, "Tensorfi: A configurable fault injector for tensorflow applications," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2018, pp. 313–320.

[60] Y. Zheng, Z. Feng, Z. Hu, and K. Pei, "Mindfi: A fault injection tool for reliability assessment of mindspore applicacions," in *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2021, pp. 235–238.

[61] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari, "Pytorchfi: A runtime perturbation tool for dnns," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2020, pp. 25–31.

[62] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[63] 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)*, June 2019, pp. 26–38. [Online]. Available: https://doi.org/10.1109/DSN.2019.00018

[64] P. R. Bodmann, G. Papadimitriou, R. L. R. Junior, D. Gizopoulos, and P. Rech, "Soft Error Effects on Arm Microprocessors: Early Estimations versus Chip Measurements," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2358–2369, 2022. [Online]. Available: https://doi.org/10.1109/TC.2021.3128501

[65] H. Cho, S. Mirkhani, C.-Y. Cher, J. A. Abraham, and S. Mitra, "Quantitative evaluation of soft error injection techniques for robust system design," in *Proceedings of the 50th Annual Design Automation Conference (DAC '13)*, 2013. [Online]. Available: https://doi.org/10.1145/2463209.2488859

[66] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj, G. Black, G. Bloom, B. R. Bruce, D. R. Carvalho, J. Castrillon, L. Chen, N. Derumigny, S. Diestelhorst, W. Elsasser, C. Escuin, M. Fariborz, A. Farmahini-Farahani, P. Fotouhi, R. Gambord, J. Gandhi, D. Gope, T. Grass, A. Gutierrez, B. Hanindhito, A. Hansson, S. Haria, A. Harris, T. Hayes, A. Herrera, M. Horsnell, S. A. R. Jafri, R. Jagtap, H. Jang, R. Jeyapaul, T. M. Jones, M. Jung, S. Kannoth, H. Khaleghzadeh, Y. Kodama, T. Krishna, T. Marinelli, C. Menard, A. Mondelli, M. Moreto, T. Mück, O. Naji, K. Nathella, H. Nguyen, N. Nikoleris, L. E. Olson, M. Orr, B. Pham, P. Prieto, T. Reddy, A. Roelke, M. Samani, A. Sandberg, J. Setoain, B. Shingarov, M. D. Sinclair, T. Ta, R. Thakur, G. Travaglini, M. Upton, N. Vaish, I. Vougioukas, W. Wang, Z. Wang, N. Wehn, C. Weis, D. A. Wood, H. Yoon, and Éder F. Zulian, "The gem5 simulator: Version 20.0+," 2020. [Online]. Available: https://arxiv.org/abs/2007.03152

[67] "gem5 GitHub Repository," https://github.com/gem5/gem5, accessed: 2023-07-25.

[68] Y. S. Shao, S. L. Xi, V. Srinivasan, G.-Y. Wei, and D. Brooks, "Co-designing accelerators and soc interfaces using gem5-aladdin," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[69] J. Cong, Z. Fang, M. Gill, and G. Reinman, "Parade: A cycle-accurate full-system simulation platform for accelerator-rich architectural design and exploration," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 380–387.

[70] C. Menard, J. Castrillon, M. Jung, and N. Wehn, "System simulation with gem5 and systemc: The keystone for full interoperability," in *2017*

International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2017, pp. 62–69.

[71] S. Rogers, J. Slycord, R. Raheja, and H. Tabkhi, "Scalable llvm-based accelerator modeling in gem5," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 18–21, 2019.

[72] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, 2001, pp. 3–14. [Online]. Available: https://doi.org/10.1109/WWC.2001.990739

[73] 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.

[74] A. Chatzidimitriou, G. Papadimitriou, D. Gizopoulos, S. Ganapathy, and J. Kalamatianos, "Assessing the Effects of Low Voltage in Branch Prediction Units," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 127–136. [Online]. Available: https://doi.org/10.1109/ISPASS.2019.00020

[75] I. Tsiokanos, G. Papadimitriou, D. Gizopoulos, and G. Karakonstantis, "Boosting Microprocessor Efficiency: Circuit- and Workload-Aware Assessment of Timing Errors," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, 2021, pp. 125–137. [Online]. Available: https://doi.org/10.1109/IISWC53511.2021.00022

[76] G. Papadimitriou and D. Gizopoulos, "Anatomy of On-Chip Memory Hardware Fault Effects Across the Layers," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–12, 2022. [Online]. Available: https://doi.org/10.1109/TETC.2022.3205808

[77] ——, "Characterizing Soft Error Vulnerability of CPUs Across Compiler Optimizations and Microarchitectures," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, 2021, pp. 113–124. [Online]. Available: https://doi.org/10.1109/IISWC53511.2021.00021

[78] 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

[79] A. A. Nair, L. K. John, and L. Eeckhout, "Avf stressmark: Towards an automated methodology for bounding the worst-case vulnerability to soft errors," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010, pp. 125–136.

[80] D. S. Khudia and S. Mahlke, "Harnessing soft computations for low-budget fault tolerance," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 319–330.

[81] A. Chatzidimitriou, G. Papadimitriou, D. Gizopoulos, S. Ganapathy, and J. Kalamatianos, "Analysis and characterization of ultra low power branch predictors," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018, pp. 144–147.

[82] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "Machsuite: Benchmarks for accelerator design and customized architectures," in *2014 IEEE International Symposium on Workload Characterization (IISWC)*, 2014, pp. 110–119.

[83] "gem5 Replacement Policies," https://www.gem5.org/documentation/general_docs/memory_system/replacement_policies/, accessed: 2023-07-25.

[84] G. Papadimitriou, A. Chatzidimitriou, M. Kaliorakis, Y. Vastakis, and D. Gizopoulos, "Micro-Viruses for Fast System-Level Voltage Margins Characterization in Multicore CPUs," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018, pp. 54–63. [Online]. Available: https://doi.org/10.1109/ISPASS.2018.00014

[85] D. A. Jiménez, "Insertion and promotion for tree-based pseudolru last-level caches," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: Association for Computing Machinery, 2013, p. 284–296. [Online]. Available: https://doi.org/10.1145/2540708.2540733

[86] "The gem5's m5 Utility and Magic Instructions," https://gem5.googlesource.com/public/gem5/+/master/util/m5/, accessed: 2023-07-25.