Quantitative Cross-Layer Evaluation of Transient-Fault Injection Techniques for Algorithm Comparison

Regular Paper

Horst Schirmeier and Mark Breddemann
Department of Computer Science 12
Technische Universität Dortmund, Germany
e-mail: {horst.schirmeier, mark.breddemann}@tu-dortmund.de

***Abstract*—In the wake of the soft-error problem, fault injection (FI) is a standard methodology to measure fault resilience of programs and to compare algorithm variants. As detailed, e.g. gate-level machine models are often unavailable or too slow to simulate, FI is usually carried out in fast simulators based on abstracted system models, using e.g. ISA-level register injection. However, the literature deems such injection techniques too inaccurate and yielding wrong conclusions about analyzed programs.**

In this paper, we empirically challenge this assumption by applying gate-, flip-flop- and ISA-level FI techniques on an Arm® Cortex®-M0 processor. Analyzing FI results from 18 benchmark programs, we initially confirm related work by reporting SDC-rate discrepancies of up to an order of magnitude between a gate-level baseline and injection techniques on higher machine-model levels, suggesting gate-level injection should be used e.g. to select a specific sorting algorithm. We discuss why these discrepancies are, however, *to be expected*, and show that the *extrapolated absolute failure-count* metric combined with *relative* inter-benchmark measurements yield a significantly better cross-layer alignment of algorithm-resilience rankings. Our results indicate that ISA-level injection techniques suffice for evaluating and selecting program and algorithm variants on low-end processors.

I. Introduction

Computer systems have been known for decades to occasionally malfunction due to transient hardware errors (*soft errors*) [1], and have since been hardened on different hardware and software levels to reduce the likelihood of system failures. Besides explicit hardening, using e.g. software-implemented hardware fault tolerance (SIHFT), also the choice of algorithm variant can influence a program's susceptibility to soft errors, its runtime – and in consequence the probability it ends up with a faulty output (a *silent data corruption*, SDC) [2]. To choose e.g. a specific sorting algorithm for use in a soft-error prone environment, a whole zoo of existing sorting algorithms needs to be measured, compared and ranked. Similarly, from a set of SIHFT implementations, one variant is usually the most effective for a particular use-case scenario.

Fault injection (FI) has grown into the standard methodology to measure fault resilience of programs and effectiveness of SIHFT. Hardware-based FI – based on experiments with radiation sources [3], with pin-level injections, or electromagnetic interference [4] – is expensive and suffers from low controllability and repeatability. Consequently, simulation-based FI has become an important tool for analyzing application-level effects of faults, for quantifying the effectiveness of SIHFT mechanisms, and for selecting the most resilient algorithm variant for a specific application.

Note that FI is also being used to *test* SIHFT implementations for detecting all faults they are specified to cover, or to gain insights on the complete *failure-mode spectrum* a program exhibits under faults – both FI uses being out of scope of this paper. In these use cases, the goal of FI is to maximally challenge the SIHFT mechanism, or to elicit also rare failure modes by using a particularly realistic FI technique. Metrics are less momentous in these cases – e.g. it is not essential whether a SIHFT mechanism results in an SDC rate of 10% or 15% if it is supposed to provide 0%.

In this paper, we focus on soft errors on the gate level – a fault model known to closely match real-world soft errors [5], [6], [7]. FI techniques injecting in machine models on this abstraction level generally rely on slow gate-level simulations, sometimes with hardware acceleration or emulation. Besides their low speed, gate-level models of commercial CPUs are very hard to come by, which often precludes experiments with low-level machine models. In contrast, high-level CPU models are generally available; FI techniques aiming at higher abstraction levels – e.g., injecting in the ISA-level register file – are much less precise but can be very fast [8], and also ease understanding erroneous application behavior [9].

In order to select an appropriate FI technique, developers must choose the lesser of two evils: extremely slow injection speeds – if a detailed machine model can be obtained at all, – or high-level model inaccuracies potentially rendering the results useless. Unfortunately, recent cross-layer FI studies report that high-level techniques are too inaccurate to be useful [8], or document significant discrepancies at least for specific failure modes [10]. From their analysis of FI-result comparison between flip-flop level and higher abstractions, Cho et al. conclude that high-level FI *"can result in high degrees of inaccuracies by more than an order of magnitude"* [8], quoting an error of up to factor 45 with no trend towards over- or underapproximation. However, Cho et al. also conclude that *"accuracy is not necessarily a requirement"* and that *"an inaccurate error injection technique can be very useful as long*

as it is effective in driving the correct design decisions for building robust systems." [8]

In this paper, we explore this opportunity for fast FI experiments with high-level FI techniques, and analyze whether they lead to correct design decisions. We compare gate-level FI results on a low-end microprocessor – an Arm® Cortex®-M0 modeled on the gate/net-list level – to results on the flip-flop and register level. We quantitatively confirm Cho et al. [8] by reporting SDC-rate discrepancies of up to an order of magnitude between the gate-level ground truth and high-level FI techniques, suggesting that the latter is too inaccurate for ranking program variants. Subsequently, we discuss why these discrepancies are, however, to be expected, and show that the *extrapolated absolute failure-count* metric [11] combined with *relative* inter-benchmark measurements yield a significantly better cross-layer alignment of algorithm-resilience rankings. In effect, our results indicate that ISA-level injection techniques suffice for evaluating and selecting program and algorithm variants on low-end processors.

In particular, the contributions of this paper are:

- We perform FI experiments on gate-, flip-flop and register-file levels of an Arm® Cortex®-M0 microprocessor, and quantitatively – in order to be able to relate to the work of Cho et al. [8] – confirm high levels of SDC-rate inaccuracies between low- and high-level FI techniques (Section IV).
- We discuss these cross-layer fault-coverage discrepancies and provide an explanation why they are to be expected *already on the metric level*. We demonstrate that – despite up to an order of magnitude difference between results from different abstraction levels – for a subset of the 18 benchmarks we used, the right design decisions would have been made as the *relative inter-benchmark rankings* stay roughly similar across FI techniques (Section IV).
- We show that when using the *extrapolated absolute failure count* (EAFC) metric [11] instead of the fault-coverage factor, *virtually all* design decisions based on high-level FI results are correct when compared to the gate-level ground truth (Section V).

The following section discusses related work regarding cross-layer FI-result comparisons. Section III describes our FI setup including system, fault and failure models, and the benchmarks we apply them to. Section VI concludes the paper.

II. Related Work

In the literature, few works can be found that *quantitatively* address FI-result interpretation across different FI techniques or system- and fault-model abstractions.

Rebaudengo, Sonza Reorda and Violante [12] report up to 400 percent failure-rate inaccuracies for the SPARC LEON processor when injecting on the register level, compared to a ground truth with injections in the processor pipeline and internal caches. Arlat et al. [13] compare hardware-based FI – using heavy-ion radiation, pin-level stuck-at faults, and electromagnetic interference – and offline injection into the target program's code and data segments. From their study

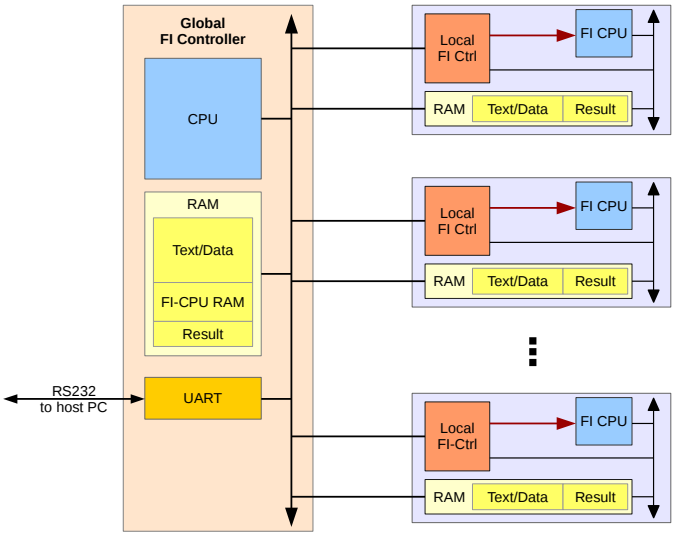


Fig. 1. Architecture of FPGA-based FI system: Multiple Cortex-M0 cores run FI experiments in parallel, while a global FI controller component oversees the campaign, resets FI CPUs and memories, configures local FI controllers with concrete fault coordinates, and collects result data.

on the fault-tolerant MARS real-time system, the authors report similar experiment outcomes for the different injection techniques but stay rather vague regarding differences in failure rates. Similarly, Sanda et al. [14] compare radiation-experiment results to flip-flop level injections and report a close match between both FI techniques.

Cho et al. [8] target an FPGA-implemented SPARC LEON and the IVM processor, and inject on the flip-flop, register-file and program-variable level using different fault distributions. They report that their high-level injection techniques create outcome-rate inaccuracies up to more than an order of magnitude with no single trend towards under- or overapproximation. The authors subsequently analyze the causes for the observed inaccuracies. Wei et al. [10] conduct a similar study but compare LLVM [15] FI with machine-code level injections. They report highly accurate SDC-rates but different crash rates when comparing both injection techniques.

All related work we are aware of quantifies FI-count relative outcome *rates* – or, similarly, fault-coverage rates, – and reports different levels of inaccuracies. In this paper, we follow the same practice, report similar levels of inaccuracies (Section IV), and come up with an explanation and remedy.

III. Fault-Injection Setup

In this section we describe the FI setup including the system models we injected into, the workloads we ran while injecting, and the applied fault and failure models.

A. Target System

We created an FPGA-based FI system with multiple instances of an Arm Cortex-M0 Verilog model on a Xilinx Virtex-7 FPGA. We chose the 32-bit Cortex-M0 with a 3-stage pipeline because it fits multiple times on the FPGA (for parallelization purposes) even with our *saboteur* modifications that allow

TABLE I
FAULT-EFFECT LOOKUP TABLE USED FOR GATE-LEVEL FI INTO ONE OUT OF 10 "VIRTUAL" GATES AN EDGE-TRIGGERED D FLIP-FLOP CONSISTS OF.

Gate	Effect	Gate	Effect
1	timing error	6	Q' undefined
2	timing error	7	$Q' = \neg D$
3	$Q' := 1$	8	$Q' = D$
4	$Q' := 0$	9	Q' undefined
5	Q' undefined	10	Q' undefined

the injection of transient gate-level faults. The Cortex-M0 is used in several commercially available microcontrollers, and is known for its cost- and energy efficiency.

Figure 1 gives an architectural overview of our FI system. On the left, a global FI controller with an administration interface via RS232 to a host PC executes FI campaigns by generating fault coordinates (time instance/fault location pairs), resetting and initializing the FI CPUs and memories, configuring local FI controllers with fault coordinates, and collecting result data. The global FI controller's RAM contains a control program, an initial memory image for the FI CPUs, and the *golden run* result to compare against when each FI experiment finishes.

The right half of Figure 1 shows multiple FI components containing a local FI controller, which is configured by the global controller and injects the actual faults, a local RAM, and a modified Arm Cortex-M0 as a FI CPU. The obfuscated Verilog code of the Cortex-M0 is already very close to a gate-level model, which allowed us with some scripting effort to add *saboteurs* that invert the output of each gate when triggered. In order to stay within FPGA-resource budgets and reasonable operation frequencies, but still be able to run several experiments in parallel, we *partition* the space of all possible fault locations: Not every FI CPU is instrumented with a saboteur for all possible fault locations, but only a subset of them – while still ensuring that each fault location is instrumented in at least one FI CPU. With this partitioning, we fit 16 saboteur-instrumented Arm Cortex-M0 CPUs on the Xilinx Virtex-7, which can be used to capacity when injecting in uniformly selected, random fault locations.

As CPU-internal flip-flops are not modeled on the gate level but mapped to real flip-flops in FPGA slices, we dealt with them specifically to mimic gate-level faults in flip-flops. We model them as edge-triggered D flip-flops consisting of 8 NAND- and 2 NOT gates, and use a pre-computed fault-effect lookup table (see Table I) when injecting into one of those "virtual" gates. Depending on the affected gate, Q' (the flip-flop's internal state) can be unaffected, be forced to 0, 1, be inverted, or undefined (which we implement by forcing a random value). We ignore timing errors in this gate-level flip-flop model.

B. Workloads

In total, we use a set of 18 benchmarks with low memory footprint in order to fit into the FPGA's on-chip memory:

- From the *MiBench Embedded Benchmark Suite* – consult the original *MiBench* paper [16] for benchmark details – we ported the *automotive* (benchmarks: *basicmath*, *bitcount*, *susan*) and *security* (*blowfish*, *rijndael*, *sha*) suites to our FI platform. The primary modification we carried out was the removal of filesystem access for reading input data, and replacing it by directly linking the data into the benchmark binaries. In several cases we also had to reduce the input-data size to stay within our memory limits.[1]
- As an example for choosing the most resilient variant out of a set of multiple algorithm variants, we used different *sorting algorithm* implementations as workloads: *bubblesort*, *gnomesort*, *heapsort*, *insertionsort*, *mergesort*, *quicksort*, *selectionsort*, *shellsort*.
- Also, we used two sorting benchmarks in a baseline variant and a SIHFT variant protected with Δ-encoding [17]: Δ-*bubblesort-base/-prot* and Δ-*quicksort-base/-prot*.

TABLE II
BENCHMARKS WITH RUNTIMES IN CPU CYCLES.

Benchmark	Runtime in CPU cyles
Δ-bubblesort-base	579,048
Δ-bubblesort-prot	24,588,989
Δ-quicksort-base	789,990
Δ-quicksort-prot	15,751,833
mibench-auto-basicmath	118,033,276
mibench-auto-bitcount	52,623,529
mibench-auto-susan	38,605,276
mibench-security-blowfish	1,727,563
mibench-security-rijndael	2,218,896
mibench-security-sha	332,638
bubblesort	3,718,167
gnomesort	1,724,176
heapsort	144,061
insertionsort	933,441
mergesort	241,393
quicksort	108,060
selectionsort	2,759,685
shellsort	148,528

Table II lists all 18 benchmarks with their runtime as measured on the Arm Cortex-M0.

C. Fault Model and Fault-Injection Techniques

We assume single-event upsets in the CPU as our fault model, implemented by gate-output inversions for the lowest-level FI technique, and transient single- and burst bit-flips for the higher-level FI techniques. Concretely, we use the following FI techniques:

- **Gate-level:** During a benchmark run, we uniformly choose a random point in time and a random gate (from a total of 20,956 gates). When reaching the specified cycle, the FI system inverts the gate's output for one cycle before returning to normal operation. We assume gate-level results to be the most realistic and choose them as our *ground truth*.

[1] Benchmarks available at: https://github.com/danceos/edcc2019-mibench

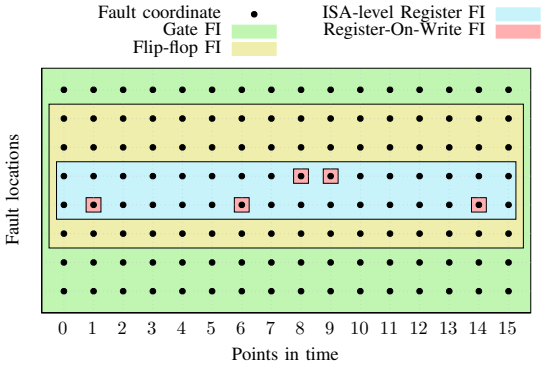


Fig. 2. Overlapping fault spaces for different FI techniques and fault models: Gate-level FI injects into the set of all available CPU gates – i.e. all possible injection (time/space) coordinates, – while the other techniques inject in decreasing subsets thereof.

- **Flip-flop level:** Similarly to the gate level, we uniformly choose a random point in time and a random flip-flop (from a total of 840 flip-flops in the design). At the specified time, we invert the flip-flop's state.
- **Register-file level:** Only injects in flip-flops that are part of an ISA-visible register. We implement different fault models on this level:
  - **Register:** Random injection time chosen uniformly, single-bit flip (in one out of 508 register bits).
  - **Register Byte:** Similar to *Register*, but flip 8 bits at once (in one out of 60 byte positions), also known as *burst bit-flip*. We chose this fault model, because the results of Cho et al. [8] indicate that multi-bit faults on higher abstraction levels are a good model for faults on the lower levels; lacking realistic multi-bit patterns to inject, we approximate the model by simply injecting in 8 adjacent register bits.
  - **Register-on-Write:** Injection time chosen randomly from all times when the register is being written into, also known as *Inject-on-Write*. We chose this fault model as it is generally known to well model CPU-internal faults by injection on the ISA-register level [18] and should, hence, correlate well with lower-level FI techniques.

Figure 2 schematically summarizes the different FI techniques and fault models we apply on the FPGA-based FI platform: Higher-level fault coordinates (fault location/injection time) are a subset of those from lower-level models, with the gate-level model encompassing all possible FI coordinates of higher-level models.

D. Failure Model

In the initial analysis steps in the next section we distinguish the following experiment-result types:

- **No Impact:** The workload does not exhibit any observable deviation from a fault-less *golden run*.
- **Timeout:** The workload was aborted when not terminating normally within $2\times$ the runtime of the golden run.
- **Lockup:** The Arm Cortex-M0 reported a *lockup* condition.
- **Silent Data Corruption (SDC):** The workload terminated normally, but its output deviated from the golden run.
- **Detected:** The workload itself detected an error; the two benchmarks protected with the Δ-encoding scheme are explicitly capable of error detection.

In the later part of the analysis, we focus on SDC, as we assume all other result types to be either benign or detectable by hardware mechanisms (e.g. by a watchdog chip).

IV. Results: Classic Interpretation

In this section, we present raw FI results and discuss their interpretation with the classic fault-coverage factor metric used in related work. Subsequently we investigate whether – in spite of high cross-layer fault-coverage discrepancies in our result data – still the right design decisions would have been made using only results from a high-level FI technique. The section concludes with an analysis of the suitability of the fault-coverage factor for cross-layer comparison.

A. Result-Distribution Comparison

Figure 3 presents the raw FI results, broken down by workload, FI technique, and experiment outcome. For each fault model and workload combination, we ran 20,000 FI experiments to achieve a 99 percent confidence interval smaller than ± 1.6 percent, using the conservative occurrence probability $p = 0.5$ assumption Leveugle et al. [19] propose.

Already at first glance the results show that *for no single workload* any of the high-level result distributions resembles the distribution of the gate-level ground truth. A closer look at the numbers reveals that result-type percentages are in some cases off by up to an order of magnitude, confirming similar reports by Cho et al. [8] when comparing a flip-flop level ground truth to FI results on higher abstraction levels:

- **No Impact** is off by down to $0.335\times$ for *insertionsort* when comparing the Register-file FI technique (fault model: Register-on-Write) results with the gate-level ground truth.
- **Timeout** is off by up to $3.16\times$ (*mibench-auto-bitcount*, Register Byte) and down to $0.127\times$ (*selectionsort*, Register-on-Write).
- **Lockup** is off by up to $19.38\times$ (*insertionsort*, Reg. Byte).
- **SDC** deviates by up to $13.7\times$ (Δ-*quicksort-prot*, Register-on-Write) and down to $0.544\times$ (Δ-*bubblesort-prot*, Register-on-Write).
- **Detected** is off by up to $4.38\times$ (Δ-*bubblesort-prot*, Register-on-Write).

When focusing on the first three FI techniques (gate-, flip-flop-, register-level), a clear trend can be made out: The result percentages for all result types besides *No Impact* tend to *increase* from lower- to higher-level FI techniques (e.g. SDC increases for *all* workloads besides Δ-*quicksort-prot*), while *No Impact* decreases. We discuss this phenomenon in Section IV-C. Cho et al. [8] analyze in detail where cross-layer deviations come from, and why some effects from low-level FI cannot be reproduced with higher-level FI techniques. However, in the

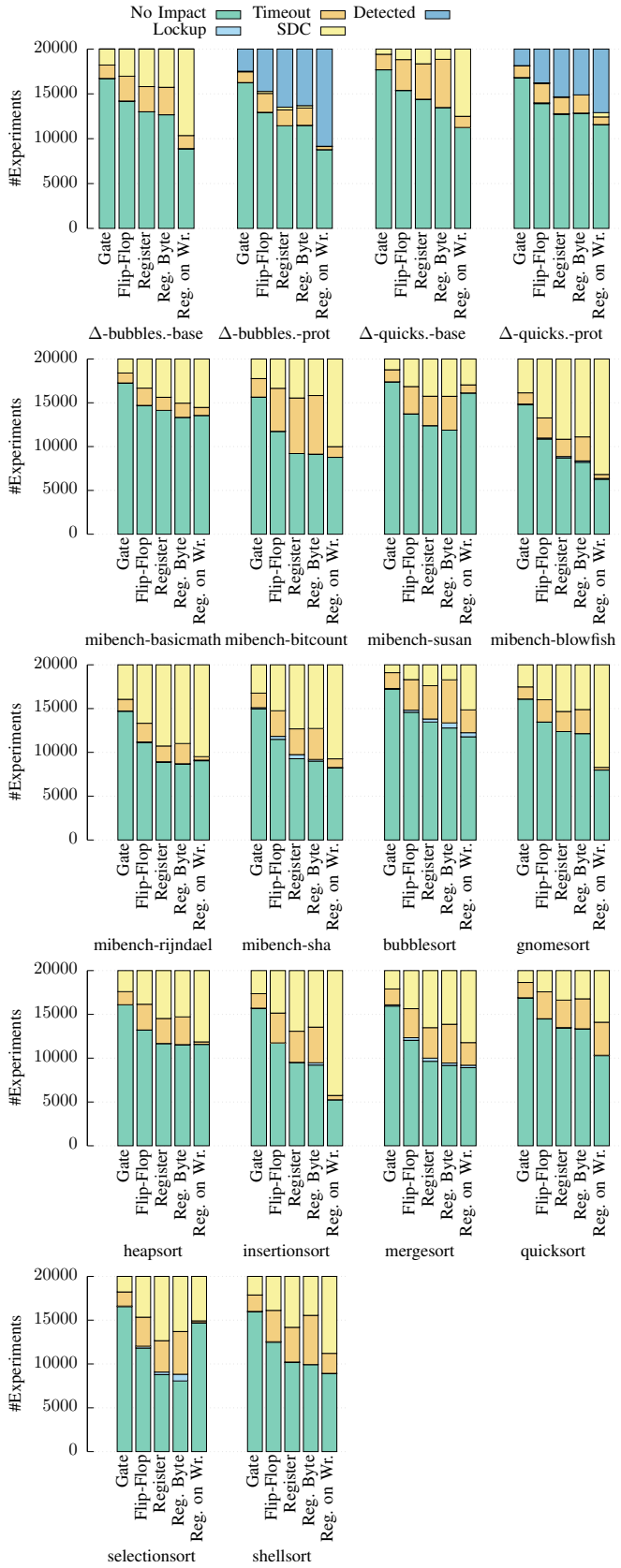
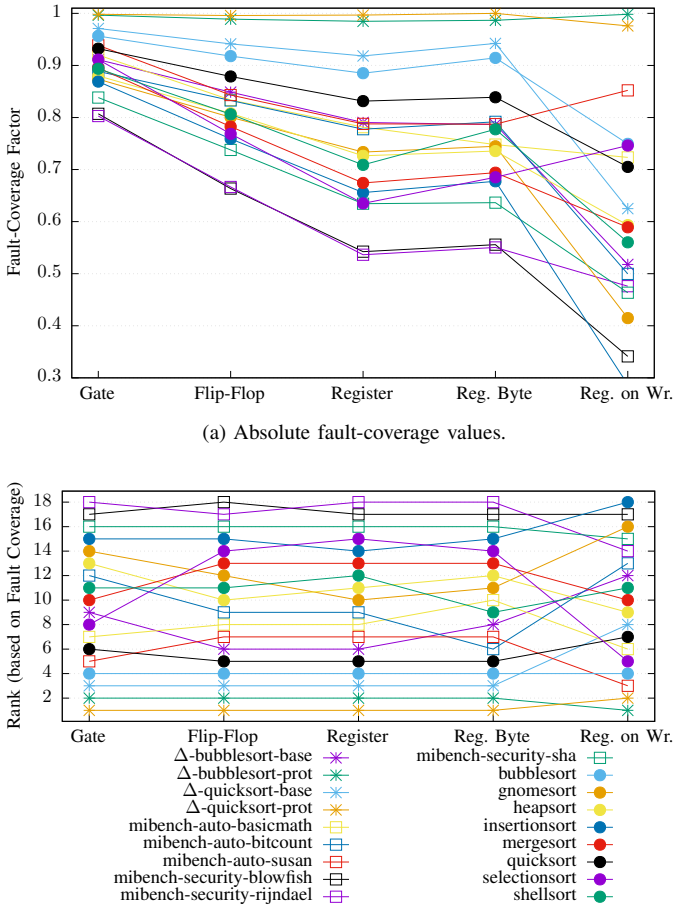


Fig. 3. Raw FI results: Across different FI techniques, for some benchmarks the proportion of specific result types is off by up to an order of magnitude compared to the gate-level baseline.



(a) Absolute fault-coverage values.

(b) Fault-coverage based workload ranking.

Fig. 4. Fault-coverage factors across FI techniques: Absolute values and extracted workload ranking.

remainder of this paper we instead want to explore how these seemingly useless results can be interpreted to drive design decisions like the selection of a particular algorithm.

B. Fault-Coverage based Workload Ranking

In their paper, Cho et al. realize that *"accuracy is not necessarily a requirement"* [8] when the FI results still lead to the right design decisions – i.e., choosing one algorithm over another, or rate one SIHFT-protected program variant better than a differently configured one that is less fault-resilient. Such design decisions are often made on the basis of the SDC rate, $\frac{\text{SDC count}}{\text{total injection count}}$, or the closely related *fault-coverage factor* [20] metric $c = 1 - \frac{\text{SDC count}}{\text{total injection count}}$ where higher values indicate more robust systems.

Figure 4a plots the fault-coverage factor – calculated from the raw results in Figure 3 – for each workload across different FI techniques. Although, e.g., the Register-on-Write fault coverage deviates by a large factor from the gate-level ground truth, the *relative ranking* within each FI technique's results could lead to the same correct design decisions. Figure 4b distills this relative ranking from the fault-coverage numbers in Figure 4a: Intersecting lines indicate some rank shuffling and potentially incorrect design decisions. Note, however, that some of the

TABLE III
RANK CORRELATION METRICS τ, K AND $\hat{K}$, COMPARING WORKLOAD RANKINGS ATTAINED BY GATE-LEVEL AND HIGHER-LEVEL FI TECHNIQUES. τ, K AND $\hat{K}$ ARE CALCULATED OVER *all* WORKLOADS, AND INDIVIDUALLY FOR THREE WORKLOAD SUBSETS (Δ, *mibench*, *sorting*), BASED ON FAULT-COVERAGE FACTOR VS. EXTRAPOLATED ABSOLUTE FAILURE COUNT (EAFC) MEASUREMENTS.

Rank correlation metrics based on Fault-Coverage Factor:

	Gate vs.	Flip-flop	Register	Reg. Byte	Reg. on Wr.
all	τ	**0.791**	**0.765**	**0.765**	**0.699**
	K ($\hat{K}$)	16 (0.105)	18 (0.118)	18 (0.118)	23 (0.150)
Δ	τ	**1.0**	**1.0**	**1.0**	**0.667**
	K ($\hat{K}$)	0 (0.0)	0 (0.0)	0 (0.0)	1 (0.167)
mib.	τ	**0.867**	**1.0**	**0.733**	**0.733**
	K ($\hat{K}$)	1 (0.067)	0 (0.0)	2 (0.133)	2 (0.133)
sort.	τ	**0.429**	**0.214**	**0.429**	**0.786**
	K ($\hat{K}$)	8 (0.286)	11 (0.393)	8 (0.286)	3 (0.107)

Rank correlation metrics based on EAFC:

	Gate vs.	Flip-flop	Register	Reg. Byte	Reg. on Wr.
all	τ	**0.948**	**0.961**	**0.948**	**0.712**
	K ($\hat{K}$)	4 (0.026)	3 (0.020)	4 (0.026)	22 (0.144)
Δ	τ	**1.0**	**1.0**	**1.0**	**-1.0**
	K ($\hat{K}$)	0 (0.0)	0 (0.0)	0 (0.0)	6 (1.0)
mib.	τ	**1.0**	**1.0**	**1.0**	**0.867**
	K ($\hat{K}$)	0 (0.0)	0 (0.0)	0 (0.0)	1 (0.067)
sort.	τ	**0.929**	**0.929**	**1.0**	**0.714**
	K ($\hat{K}$)	1 (0.036)	1 (0.036)	0 (0.0)	4 (0.143)

absolute fault-coverage values – e.g., the sorting benchmarks in the gate-level injection results – are so similar that ranking them is difficult.

A closer inspection of the workload rankings reveals that some design decisions based on high-level fault models would be correct, but by far not all. For example, both the gate-level and register-level (register) injection techniques choose *bubblesort* over all other sorting benchmarks, with *quicksort* being in second place in both cases as well. However, *selectionsort* moves from the correct third-best sorting-algorithm position to the last, being replaced by *gnomesort* that should be placed second to last according to the gate-level ground truth.

Similarly, the Δ-encoding protected benchmarks (Δ-*bubblesort-prot*, Δ-*quicksort-prot*) are in a better rank than their baseline (*-base*) pendants in all five FI techniques. However, Register-on-Write chooses Δ-*bubblesort-prot* over Δ-*quicksort-prot*, contradicting all other techniques including our gate-level baseline – although Register-on-Write is supposed to model CPU-internal faults best from the three register-level techniques.

In order to quantify the ranking "quality" of each FI technique in relation to the ranking from the gate-level ground truth, we calculate Kendall's τ [21]. This correlation metric is based on the number of rank inversions: For a set of n workloads, it counts the number K – Kendall's *tau distance* – of pairs (out of $N := \binom{n}{2}$ pairs) whose rank is *discordant*, i.e. they are in a different relative order than in the ground truth; the remaining $L := N - K$ pairs are called *concordant*. The *normalized tau distance* $\hat{K}$ calculates as $\frac{K}{N}$ (i.e., the fraction of all pairs that is discordant), and finally Kendall's $\tau := \frac{L-K}{N}$. Possible τ values lie between -1 (inverted ranking) and $+1$ (identical ranking), with 0 indicating no correlation.

The upper half of Table III shows the application of τ and the related absolute and normalized *tau distances* K and $\hat{K}$ on the ranking derived from the fault-coverage factor measurements (Figure 4b). The τ values over *all* workloads show what the statistics literature calls "strong" correlation [22] between the ground truth and higher-level FI techniques, but in the case of register-level FI 18 (Register) up to 23 (Register-on-Write) out of 153 workload pairs are discordant – meaning wrong design decisions would have been made in these cases. Applying the metrics to the rankings *within* the three workload subsets – the Δ-encoding benchmarks, MiBench, and the sorting algorithms – shows that especially the sorting algorithms show little agreement between register-level FI and the ground truth ($\tau = 0.214$, 11 discordant pairs out of 28).

C. Discussion

Although cross-layer comparisons seem to improve with these relative and rank-based interpretations, our *absolute* fault-coverage factor measurements (Figure 4a) still significantly deviate from the gate-level baseline – similar to the results of Cho et al. [8]. However, we argue that this difference is not surprising: From lower- to higher-level FI techniques, more and more possible fault locations are removed from the fault space (see Figure 2). Higher-level fault models tend to concentrate injection into the "more important" structures of the machine:

- **Flip-flop level** FI targets only longer-lasting state bits instead of – additionally – large amounts of logic gates, where a temporary output inversion is more likely to be masked. Hence, flip-flop level faults should be expected to propagate with higher probability.
- **Register-level** FI again subsets the flip-flop level fault space and injects only into those state bits that are *already visible on the ISA level*, and, hence, have an even shorter path to propagate to the program's output than the remaining flip-flops.

In effect, the trend towards lower fault-coverage factor measurements in higher-level FI in Figure 4a is primarily caused by the fault-location selectivity of these techniques. In other words, *"inaccuracies by more than an order of magnitude"* [8] should not come as a surprise but *are to be expected.*

However, in earlier work [11] we noted that the fault-coverage factor metric is *generally* unsuitable for comparing and ranking programs running on the same system with the same fault model. We attribute this to the fact that this metric does not take different fault-space sizes – e.g., due to different program runtimes – into account. As a remedy, we proposed the *extrapolated absolute failure count* (EAFC) metric, which in essence multiplies the SDC rate (or $1 - c$, see Figure 4b) by the size w of the fault-space the FI samples are taken from:

$$F_{\text{extrapolated}} = w \cdot \frac{\text{SDC count}}{\text{total injection count}}$$

Note that the related *vulnerability* metric [23] multiplies the SDC rate by the program's runtime, essentially taking

into account *one* of the two w dimensions. The EAFC is proportional to the failure probability $P(\text{Failure})$ – with a potentially unknown, hardware- and environment dependent proportionality factor g [11] – and, hence, a good choice for program comparison *within the same system and fault model*.

However, the EAFC metric seems not to be suitable for direct program comparison across different system and fault models, as is the case in this paper: We must assume different proportionality factors g for FI on different abstraction levels. For example, on the gate level we inject into 20,956 gates, while on the flip-flop level we target 840 flip-flops, so we should expect a reduction factor of roughly 25 between EAFC measurements using those FI techniques. This factor will actually be lower than 25 due to the aforementioned higher propagation probability in higher-level FI techniques.

V. Results: EAFC Interpretation

Consequently, in this section we re-evaluate the FI results using the EAFC metric. We explore whether an approximately linear relationship exists between EAFC measurements using gate-level FI and higher-level FI techniques for the Arm Cortex-M0 processor and the 18 workloads. Additionally, we evaluate whether the workload *ranking* stays intact even better across different models when using an EAFC-based ranking instead of the one based on the fault-coverage factor metric.

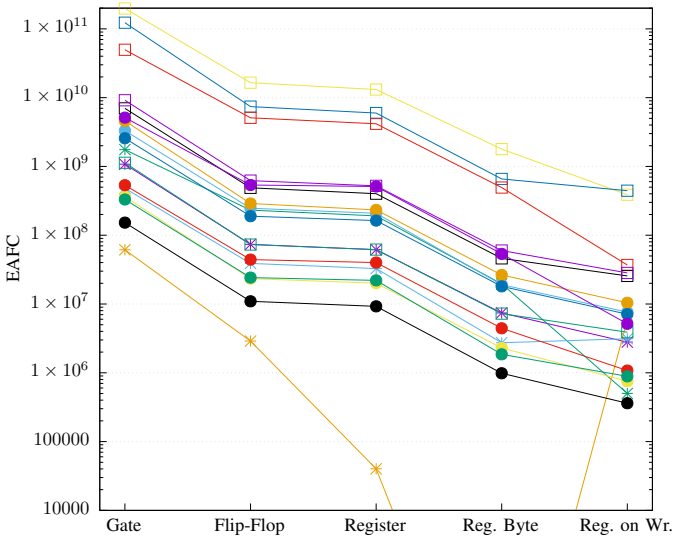
A. Cross-Layer EAFC Correlation

Figure 5a plots the EAFC for each workload across different FI techniques. It is calculated from the raw results in Figure 3 and the fault-space sizes w for each workload, which in turn can be calculated by multiplying their runtime (in CPU cycles) with the number of fault locations that is different for each FI technique, as suggested in Figure 2. As the resulting EAFC can grow quite large for longer-running workloads – for example, mibench-basicmath runs for about $1.18{\times}10^8$ cycles – the Y-axis in Figure 5a has a logarithmic scale. This also means that, unlike the absolute fault-coverage values, the EAFC values are spread out over several orders of magnitude, and much better rankable. The step from gate-level to flip-flop-level EAFC in Figure 5a indeed shows an average reduction factor of $13.7\times$ (min. $7.6\times$, max. $21.2\times$), as anticipated in Section IV-C.

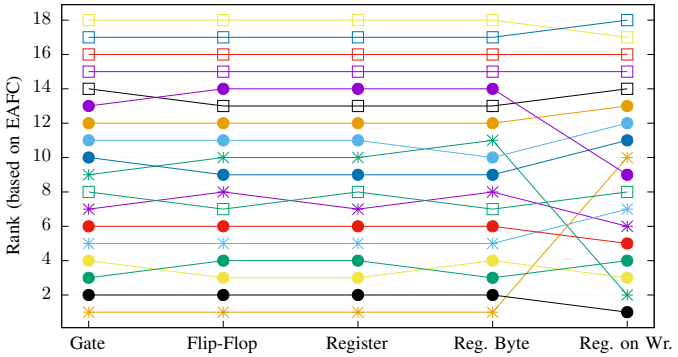
B. EAFC-based Workload Ranking

Figure 5b shows the relative ranking within each FI technique, based on the EAFC measurements in Figure 5a. Qualitatively the ranking seems to stay much more stable across the different techniques than in the fault-coverage based ranking (see Figure 4b). All three workload subsets – Δ-encoding, *MiBench*, and *sorting* – keep their relative order within each subset *across all FI techniques* except Register-on-Write; only the neighboring *shellsort* and *heapsort* swap places, but their absolute EAFC values are almost identical in the gate-level model (see Figure 5a), so no clear "winner" could be declared even in the ground-truth data.

The lower half of Table III quantitatively confirms that the rankings correlate "very strongly" across all FI techniques



(a) Absolute EAFC values (**log. scale**).



(b) EAFC-based benchmark ranking.

Fig. 5. EAFC measurements across FI techniques: Absolute values and extracted benchmark ranking.

– again, except Register-on-Write, – indicating that *design decisions based on EAFC measurements are correct in most cases also with high-level FI*. Register-on-Write is the notable exception: it both qualitatively (Figure 5b) and quantitatively (Table III) seems to introduce a high level of inaccuracy, with the extreme case of the Δ-encoding workloads that receive *a completely inverted ranking* ($\tau = -1$). This is especially surprising as Register-on-Write is the register-level technique that is supposed to model CPU-internal faults best.

Another notable difference to the fault-coverage factor based plots is that Δ-*quicksort-prot* is still chosen over its unprotected baseline Δ-*quicksort-base* – but Δ-*bubblesort-prot* seems to perform *worse* than its unprotected baseline Δ-*bubblesort-base* across all FI techniques (besides Register-on-Write) when evaluated with the EAFC. The reason could either be a real fault-resilience deterioration due to the increased attack surface introduced by the encoding – the authors of the original paper only used the fault-coverage factor metric [17], which does show an improvement also in our measurements – or simply have been introduced by our port from 64-bit AMD64 to 32-bit

Arm Cortex-M0. The issue calls for further investigation, which we postpone to future work.

C. Threats to Validity

The experimental results in this study are limited to specific conditions and scope. First of all, a gate-level CPU model is a coarse simplification of actual CPU structures with, e.g., transistor- and device-level enhancements, which are designed to better withstand radiation. However, consideration of these factors is probably only possible with radiation experiments on real hardware – with the disadvantages mentioned in Section I. Secondly, the Arm Cortex-M0 is a low-end microprocessor without caches, memory protection, or any other complex functionality unnecessary in its application domain; the results from this paper are limited to this low-end CPU class, which nevertheless has a significant market share. Thirdly, the used benchmarks could be seen as too short/simple, too low in number and too narrow in scope, which we intend to remedy in future work.

VI. Conclusions

To summarize, FI is often carried out in fast simulators based on abstracted system models and FI techniques injecting, e.g., only in ISA registers. However, the literature claims that such FI techniques are too inaccurate, in effect falsifying design decisions concerning analyzed programs. In this paper, we empirically challenged this claim by applying different FI techniques on gate-, flip-flop- and ISA levels of an Arm Cortex-M0 processor to a set of 18 benchmark programs.

Analyzing FI results, we quantitatively confirmed related work [8] by reporting SDC-rate discrepancies of up to an order of magnitude between a gate-level baseline and higher-level FI techniques. We discussed why these discrepancies are, however, to be expected, and show that the *extrapolated absolute failure-count* metric combined with *relative* inter-benchmark measurements yield a significantly better cross-layer alignment of algorithm-resilience rankings. Our results indicate that high-level FI is useful in comparing the effectiveness of different algorithms or SIHFT mechanisms on low-end processors, and that the inject-on-write FI technique – generally known to model CPU-internal faults well – may not be the best ISA-level technique for this purpose.

Acknowledgments: We thank our anonymous reviewers – also of a previous iteration of this paper – for their very detailed and helpful comments. This work was partly supported by the German Research Foundation (DFG) priority program SPP 1500 under grant no. SP 968/5-3. We also thank Dmitrii Kuvaiskii for his Δ-encoding [17] benchmarks.

References

[1] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *11th IEEE Int. Symp. on High-Performance Computer Architecture (HPCA '05)*. IEEE, 2005.

[2] Q. Guan, N. DeBardeleben, S. Blanchard, and S. Fu, "Empirical studies of the soft error susceptibility of sorting algorithms to statistical fault injection," in *5th W'shop on Fault Tolerance for HPC at eXtreme Scale*, ser. FTXS '15. New York, NY, USA: ACM, 2015, pp. 35–40.

[3] J. Karlsson, P. Liden, P. Dahlgren, R. Johansson, and U. Gunneflo, "Using heavy-ion radiation to validate fault-handling mechanisms," *IEEE Micro*, vol. 14, no. 1, pp. 8–23, Feb. 1994.

[4] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. H. Leber, and J. Reisinger, "Application of three physical fault injection techniques to the experimental assessment of the MARS architecture," in *Conf. on Dep. Comp. for Crit. App. (DCCA '95)*. IEEE, 1995.

[5] N. Karimi, M. Maniatakos, A. Jas, and Y. Makris, "On the correlation between controller faults and instruction-level errors in modern microprocessors," in *2008 Int'l Test Conf. (ITC '08)*. IEEE, Oct. 2008.

[6] M.-L. Li, P. Ramachandran, U. R. Karpuzcu, S. K. Sastry Hari, and S. V. Adve, "Accurate microarchitecture-level fault modeling for studying hardware faults," in *15th IEEE Int. Symp. on High Performance Computer Architecture (HPCA '09)*. IEEE, Feb. 2009, pp. 105–116.

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

[8] 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 *50th Design Autom. Conf. (DAC '13)*. New York, NY, USA: ACM, May 2013, pp. 1–10.

[9] H. Schirmeier, M. Hoffmann, C. Dietrich, M. Lenz, D. Lohmann, and O. Spinczyk, "FAIL*: An open and versatile fault-injection framework for the assessment of software-implemented hardware fault tolerance," in *11th Europ. Depend. Comp. Conf. (EDCC '15)*. Piscataway, NJ, USA: IEEE, Sep. 2015, pp. 245–255.

[10] J. Wei, A. Thomas, G. Li, and K. Pattabiraman, "Quantifying the accuracy of high-level fault injection techniques for hardware faults," in *44th IEEE/IFIP Int. Conf. on Dep. Sys. & Netw. (DSN '14)*. Washington, DC, USA: IEEE, Jun. 2014, pp. 375–382.

[11] H. Schirmeier, C. Borchert, and O. Spinczyk, "Avoiding pitfalls in fault-injection based comparison of program susceptibility to soft errors," in *45th IEEE/IFIP Int. Conf. on Dep. Sys. & Netw. (DSN '15)*. Piscataway, NJ, USA: IEEE, Jun. 2015, pp. 319–330.

[12] M. Rebaudengo, M. S. Reorda, and M. Violante, "Analysis of SEU effects in a pipelined processor," in *8th IEEE International On-Line Testing Workshop (IOLTW 2002)*, Jul. 2002, pp. 112–116.

[13] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, and G. H. Leber, "Comparison of physical and software-implemented fault injection techniques," *IEEE TC*, vol. 52, no. 9, pp. 1115–1133, Sep. 2003.

[14] P. N. Sanda, J. W. Kellington, P. Kudva, R. Kalla, R. B. McBeth, J. Ackaret, R. Lockwood, J. Schumann, and C. R. Jones, "Soft-error resilience of the IBM POWER6 processor," *IBM Journal of Research and Development*, vol. 52, no. 3, pp. 275–284, May 2008.

[15] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *Proceedings of the 2nd International Symposium on Code Generation and Optimization (CGO'04)*. Piscataway, NJ, USA: IEEE, Mar. 2004.

[16] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE Int. W'shop on Workload Characterization (WWC '01)*. Washington, DC, USA: IEEE, 2001, pp. 3–14.

[17] D. Kuvaiskii and C. Fetzer, "Δ-encoding: Practical encoded processing," in *45th IEEE/IFIP Int. Conf. on Dep. Sys. & Netw. (DSN '15)*. IEEE, Jun. 2015, pp. 13–24.

[18] B. Sangchoolie, F. Ayatolahi, R. Johansson, and J. Karlsson, "A comparison of inject-on-read and inject-on-write in ISA-level fault injection," in *11th European Dependable Computing Conference (EDCC)*, Sep. 2015, pp. 178–189.

[19] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Conf. on Design, Autom. & Test in Europe (DATE '09)*. IEEE, Apr. 2009, pp. 502–506.

[20] W. G. Bouricius, W. C. Carter, and P. R. Schneider, "Reliability modeling techniques for self-repairing computer systems," in *24th National Conference*, ser. ACM '69. New York, NY, USA: ACM, 1969, pp. 295–309.

[21] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1-2, pp. 81–93, 1938.

[22] C. P. Dancey and J. Reidy, *Statistics Without Maths for Psychology*. Pearson Education, 2007.

[23] N. Narayanamurthy, K. Pattabiraman, and M. Ripeanu, "Finding resilience-friendly compiler optimizations using meta-heuristic search techniques," in *12th Europ. Depend. Comp. Conf. (EDCC '16)*, 2016.