Evaluating the Effectiveness of Microarchitectural Hardware Fault Detection for Application-Specific Requirements

Konstantinos-Nikolaos Papadopoulos[1]   Christina Giannoula[2]   Nikolaos-Charalampos Papadopoulos[1]

Nektarios Koziris[1]   José M.G. Merayo[3]   Dionisios N. Pnevmatikatos[1]

[1]*National Technical University of Athens*
[2]*University of Toronto*
[3]*Technical University of Denmark*

Abstract

Reliability is necessary in safety-critical applications spanning numerous domains. Conventional hardware-based fault tolerance techniques, such as component redundancy, ensure reliability, typically at the expense of significantly increased power consumption, and almost double (or more) hardware area. To mitigate these costs, microarchitectural fault tolerance methods try to lower overheads by leveraging microarchitectural insights, but prior evaluations focus primarily on only application performance. As different safety-critical applications prioritize different requirements beyond reliability, evaluating only limited metrics cannot guarantee that microarchitectural methods are practical and usable for all different application scenarios. To this end, in this work, we extensively characterize and compare three fault detection methods, each representing a different major fault detection category, considering real requirements from diverse application settings and employing various important metrics such as design area, power, performance overheads and latency in detection. Through this analysis, we provide important insights which may guide designers in applying the most effective fault tolerance method tailored to specific needs, advancing the overall understanding and development of robust computing systems. For this, we study three methods for hardware error detection within a processor, i.e., (i) Dual Modular Redundancy (DMR) as a conventional method, and (ii) Redundant Multithreading (R-SMT) and (iii) Parallel Error Detection (ParDet) as microarchitecture-level methods. We demonstrate that microarchitectural fault tolerance, i.e., R-SMT and ParDet, is comparably robust compared to conventional approaches (DMR), however, still exhibits unappealing trade-offs for specific real-world use cases, thus precluding their usage in certain application scenarios.

1. Introduction

Modern computing systems demand strong robustness, safety, and/or security properties, thus requiring high tolerance against transient and permanent hardware faults, such that to prevent failures and guarantee uninterrupted and reliable operation [20, 37, 38, 62, 91]. Hardware failures can jeopardize human lives, or disrupt the successful completion of critical and costly tasks [9, 40, 60, 75]. For microelectronic circuits operating in high radiation environments, increased reliability is paramount due to radiation interference, which causes highly increased error rates [19]. However, errors are present even in lower radiation environments. This is because technology scaling in modern Integrated Circuits (ICs), aimed at enhancing performance and energy efficiency through the miniaturization of transistor sizes, also increases susceptibility to radiation-particle induced faults [67, 81]. Moreover, this susceptibility is expected to increase further with continued scaling. Likewise, chip manufacturers increase reliability by introducing conservative margins in operating voltage and frequency, and future microprocessors will need robust tolerance to transient errors in order to exploit lower margins and harness back power and performance gains [1, 96].

All such systems require fault tolerance mechanisms, like fault detection, which however, in order to increase reliability come with the cost of either negatively impacting performance, or increasing power consumption, or IC area, depending on the method design and trade-offs. As a result, different methods exhibit different overheads in a number of metrics and represent distinct points in the fault detection method design space. The selection of the most suitable method, depends on the constraints of the design and the application domain, since apart from reliability, critical systems from various application domains have different requirements depending on their particular characteristics. For example, High Performance Computing (HPC) applications, which have growing needs for reliability due to scaling and tight margins, but they also primarily need high performance to solve complex problems and process massive amounts of data [26, 61]. Instead, deep space missions prioritize the need for low area and power costs due to energy constraints in spacecraft probes with limited battery capacities [36, 80]. Therefore, in both cases hardware fault tolerance mechanisms have to be deployed, which however need to meet different constraints: the first mechanism must exhibit low performance overheads, while the second one low area and power overheads. In like manner, other application scenarios have other distinct requirements for performance, area or power and demand
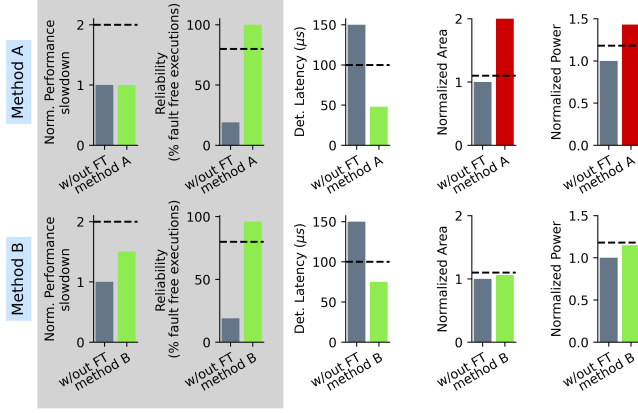
Figure 1: Different fault detection mechanisms can exhibit similar performance and reliability but differ significantly in other unevaluated metrics (detection latency and area). Thus, evaluations must consider all relevant metrics. Here, Method A (DMR) and Method B (R-SMT) are assessed against design constraints from a nano-satellite application. Green bars meet the constraints, while red bars violate them.

fault tolerance mechanisms which comply with their unique constraints. As a result, engineers need to have an accurate estimation on the strengths and weaknesses of each method, covering various metrics apart from performance, when choosing one for a particular application.

Prior works propose effective error detection methods which leverage microarchitectural insights [2, 8, 31, 50, 58, 59, 66, 71, 74, 89, 92], aiming to achieve high performance and low area footprint of the secured systems, and reduce by-design some of the overheads associated with fault tolerance. However, such works mainly focus on the performance aspect of their methods, and do not comprehensively evaluate overheads in other metrics, such as area or power (i.e. [8, 31, 50, 58, 59, 66, 74, 82, 89]), which are of crucial importance for some application domains. In Fig. 1, we demonstrate that by assessing only performance and reliability, evaluations of microarchitectural fault detection methods are not complete nor adequate: as we find after reviewing a wide range of safety-critical applications (Section 3), many such systems demand low overheads in other metrics, like area, power or detection latency, rather than performance. Thus, prior works mainly cover a subset of the use case scenarios that need high performance guarantees (e.g., HPC applications) along with reliability, and overlook other use cases, such as space missions or automotive, that have different priorities.

Fig. 1 evaluates various important metrics and shows the overheads of two different fault detection methods (Method A and Method B) compared to without any fault detection ("w/out FT", in grey bars). Fig. 1 also shows the constraints of a nano-satellite system [22, 84] that requires fault detection with the dashed horizontal lines. When choosing between the two candidate fault detection mechanisms (Method A and B), if engineers only consider existing evaluations focused on performance and reliability

(grey background plots), both methods appear to meet these application constraints, and Method A might be selected. However, when additional metrics critical for orbital systems, such as detection latency, IC area and power, are assessed, it becomes evident that Method A is unsuitable due to its high area and power overheads, which cause the total area cost of the system to exceed the area and power constraints (illustrated with a red bar). Instead, Method B meets these constraints of the application, thereby being the most suitable method to be selected. This example underscores the need for a comprehensive evaluation of fault detection methods across a variety of important metrics tailored to assess the requirements of target applications.

To this end, the *goal* of our work is to extensively characterize a diverse set of different fault detection methods in various key requirements. To our knowledge, there is no prior work to extensively compare and examine microarchitectural fault detection methods across a wide range of use case application scenarios that have different key requirements. Therefore, we conduct the following:

Firstly, we review various application domains requiring robustness properties -both safety-critical and emerging ones (like HPC)- and identify that they have distinct yet important requirements beyond reliability, which however, are overlooked in most evaluations of prior hardware fault tolerance mechanisms. Based on these requirements, we group application domains on real-world use-cases to three types: (a) performance-critical, as the applications which need high performance guarantees, (b) area/power-critical, as the applications that need low area and power overheads, and (c) latency-critical, as the applications which need low error detection latency.

Secondly, based on these use-case categories, we propose a better evaluation methodology that more appropriately compares fault detection methods and their trade-offs. We provide the key metrics on which hardware error detection needs to be evaluated in order to understand in which real-world applications is best fit and in which ones might result in important application degradation. These metrics consist of (i) detection efficiency -namely the effectiveness in error detection- (ii) detection latency -the elapsed time between fault manifestation and error detection- (iii) performance, (iv) design area and (v) power.

Thirdly, we analyse prior works on hardware error detection and select 3 diverse redundancy approaches across both microarchitectural and conventional methods: (i) hardware redundancy, (ii) multithreading within microprocessors, and (iii) heterogeneous error detection mechanisms. This selection captures different fault detection capabilities, providing valuable insights into their effectiveness and efficiency in maintaining system reliability. We further select one method from each of these three major fault detection method classes: We choose Dual Modular Redundancy, a conventional method of hardware redundancy, Redundant Multithreading via Simultaneous Multithreading (R-SMT), a microarchitectural method which utilizes multithreading within microprocessors and is representative of the design

space encompassing all prior redundant multithreading techniques, and the most recent state-of-the-art heterogeneous fault detection method, Parallel Error Detection with Heterogeneous Cores (ParDet). DMR, is a method of hardware redundancy that duplicates whole hardware components, providing robustness at the expense of significantly increased power consumption and design area. R-SMT [31, 50, 58, 66, 71, 74] duplicates the execution in redundant hardware threads. ParDet [2] uses multiple parallel heterogeneous cores to re-execute parts of the program independently. Please refer to Section 2 for a comprehensive description of all prior fault detection methods.

Fourth, we implement and evaluate all methods using the gem5 simulator [14], in same core configuration for fair comparison. We implement statistical fault injection to induce transient and permanent faults to all methods, covering the whole range of appeared faults in the reviewed application domains. This approach allows to assess reliability in the microarchitectural level, offering both accurate and early in the design stage reliability results. We evaluate all representative methods using our proposed evaluation methodology (of 5 metrics) and a variety of workloads.

We provide new insights and important key observations that can be leveraged to find which method is the most appropriate for a particular application-specific computing system. By comparing DMR, R-SMT, and ParDet, researchers can gain a holistic view of the trade-offs involved in different microarchitectural redundancy strategies, their impact on system robustness, and their suitability for various applications. This comparative analysis helps in identifying the most effective fault tolerance method tailored to specific needs and advances the overall understanding and development of robust computing systems.

Our most significant key insights are:

- Microarchitectural methods i.e., R-SMT and ParDet, can have comparably high capability on identifying errors compared to conventional methods, i.e., DMR.
- Low detection latency is crucial in identifying permanent errors to guarantee high detection efficiency.
- R-SMT is suitable for area-critical applications.
- ParDet is suitable for performance-critical applications thanks to providing best performance across all evaluated workloads.

Overall, we make the following key contributions:

- We find that fault detection methods have different trade-offs in reliability, latency, performance, area and power, and that different applications have different priorities (apart from the reliability which is always the first) among these requirements. We group applications in three categories based on their secondary priority and this categorization can assist both when designing new detection methods to navigate the requirements of the prospective target applications and in selecting suitable existing methods for a specific domain.
- We propose a more appropriate evaluation methodology for fault detection methods, which consists of five key metrics. We believe that our evaluation methodology will be adopted by our research community and industry, when proposing new fault detection methods, in order for new proposed mechanisms to be proven practical and usable in the targeted application types.
- We implement and comprehensively evaluate the most representative hardware fault detection methods from a wide range of fault detection strategies and across all crucial metrics, using the methodology we establish. The new insights we provide will be useful to industry engineers for selecting the optimal methods in different application domains when designing application-specific computing systems and enhancing them with the corresponding fault detection methods.

2. Hardware Error Detection Methods

We choose to evaluate one method from each different category of hardware fault detection, namely Hardware Redundancy, Redundant Multithreading and Heterogeneous Systems, to provide insights for a variety of different fault detection strategies.

2.1. Hardware Redundancy

Spacial hardware redundancy, where the whole core or individual components are duplicated and the outputs of both copies compared for fault detection, has been used in many commercial systems like the IBM G5 [79], Tandem (now HP) NonStop [11, 13], and others [23], most recently in ARM Cortex-R CPUs [6, 41]. Component redundancy, is the primary method of choice in high-risk space missions, due to the high reliability it provides [78, 93]. Additionally, multi-core architectures like Chip Multi-Processors (CMPs) have also been proposed as a platform for DMR [32, 33, 46, 58]. In this study we evaluate a DMR scheme with one redundant core, since it provides increased robustness compared to individual component redundancy and to also provide a baseline of conventional fault detection.

2.2. Redundant Multithreading

Redundant Multithreading employs different hardware threads to run the same program, and compares their results for fault detection. Rotenberg [74] exploits two different instruction streams (A-stream and R-stream) which redundantly run on the same processor using Simultaneous Multithreading for detecting faults. Mukherjee and Reinhardt [71] expand this, evaluating performance improvements like branch and operand prediction by the leading thread and propose the idea of sphere-of-replication.

Other approaches [31, 66, 82] aim at reducing the execution overhead of the redundant thread, introducing partial redundant multithreading: SlicK [66] re-executes only store instructions along with their backward slices, eliminating completely slices which are predicted to be fault-free from the redundant thread. Opportunistic Transient Fault Detection [31] executes a redundant thread during low ILP phases and cache misses and for high ILP phases exploits instruction reuse to reduce the re-execution of the redundant thread. Slipstream processors [82] implement a

pair of streams, assisting each other in both performance and fault tolerance, with the A-stream running ahead of R-stream and eliminating unnecessary instructions from the R-stream. All these works target to improve performance over the initial design [71, 74], while maintaining reliability, degrading however other metrics, which as we identify in Section 3 are of equal or even greater importance, depending on the application setting.

Additionally, prior works only evaluate their microarchitectural methods in terms of detection efficiency and performance, thus not covering all key metrics needed to comprehensively assess the methods in real-world application scenarios. In this study, we aim to compare methods between the different categories to provide general insights for a variety of different hardware strategies, therefore as a representative method from the Redundant Multithreading category, we evaluate the original design. Other variations (i.e. [31, 66, 82] are variations of either full or partial Redundant Multithreading and therefore represent close points in the design space.

However, several prior works [31, 50, 58, 66] propose performance optimizations upon the original R-SMT scheme. In our work, we choose not to include additional performance optimizations in the R-SMT scheme. Such optimizations like perfect branch prediction and operands prediction [74] provide slight performance improvements over R-SMT (without affecting reliability), but negatively affect other metrics such as area overheads. These inherit trade-offs of the mechanism, do not drastically alter its position in the design space; all R-SMT variations represent close points, improving some metrics but deteriorating others. Since we want to provide insights for a variety of different redundancy strategies, we evaluate the lower-bound for R-SMT scheme, without any performance optimizations.

2.3. Heterogeneous Systems

Austin et al. [8, 92] combines the main core with a single checker core to re-execute instructions upon commit enabling both error detection and correction. However, this mechanism leaves the pipeline frontend unprotected, thus incurring significantly lower detection coverage. Necromancer [5] mitigates permanent manufacturing faults in CMPs. In this scheme, special, lighter, cores are added in the CMP and defective cores continue executing, providing performance enhancing hints to the lighter ones, which substitute the defective ones. Lastly, Parallel Error Detection with Heterogeneous Cores (ParDet) [2] is the state-of-the-art method in Heterogeneous Fault Detection methods, which utilizes small cores for error detection and exhibits low overheads in performance, area and power. In our analysis, we evaluate ParDet, as the latest and mostly performant efficient heterogeneous fault detection method.

3. Key Requirements of Safety-Critical Applications

Tolerating as many hardware errors as possible to provide continuous safe and reliable operations, is a pre-requisite in safety-critical systems including space [93], automotive [51], healthcare [54], and nuclear safety systems [39]. However, in such systems, robustness is not *the sole* requirement. In addition to robustness -which is always the first priority for all critical systems- different application scenarios might need to prioritize different requirements as a second priority. We analyze several critical application scenarios, and find that they can be classified in three different categories, depending on their second-level priority requirement:

(1) Performance-Critical: Safety-critical applications that need to provide high-performance capabilities along reliability can be grouped as *performance-critical*. For instance, HPC workloads target high performance efficiency, while they need to tolerate increased error rates due to transistor scaling. Certain automotive use cases also demand high performance [70], fueled by the heavy computational requirements of autonomous driving. Similarly, emerging applications in spaceborne systems need to deliver even more high performance [90] while retaining high fault tolerance, driven by the integration of artificial intelligence [29, 77] and edge computing [15, 24], which require more compute capabilities.

(2) Area/Power-Critical: In many important applications, the first priority -additionally to fault tolerance- is the minimization of silicon area, and hence such applications can be considered as *area/power-critical*. These include energy-constrained systems, such as on-board computers in deep-space missions [16, 47, 87]: larger IC area corresponds to increased energy consumption and increased probability of radiation particle collisions, which thus significantly increase the probability of errors.

(3) Latency-Critical: Latency in error detection, i.e., the time period from the time that the error arises to the time that the error is detected by the method, is another key requirement that needs to be optimized [44] in many different application domains (space [18], automotive [52], nuclear [55]), forming the class of *latency-critical* applications. Increased latency not only prolongs the response time of error-hardened systems, but also increases the performance overheads in error corrections (late detection of an error requires replaying more instructions to restore a safe state) and compromises reliability (accumulated faults that have not been yet detected significantly increase the failure risk, which is particularly important in systems with limited error tolerance).

Overall, different critical application scenarios have different second-level needs, i.e., they need to prioritize either performance, area, and latency, second to reliability. Fault detection mechanisms impose different performance and area overheads, and also have different degrees of detection efficiency and latency, thus influencing all the above requirements. Prior works [8, 31, 50, 58, 59, 66, 74, 82, 89] mainly focus their evaluations of existing fault detection methods only on performance metrics, thus targeting only a subset of the applications (e.g., performance-critical such as the HPC focused only). To this end, we conduct a comprehensive evaluation study of the a diverse set of

4

microarchitecture-level fault detection methods to present, for the first time, meaningful insights and trade-offs that cover all the aforementioned application requirements. To facilitate such a complete comparative evaluation, we consider the following metrics:

1) **detection efficiency:** the effectiveness in accurately detecting the errors occurred.
2) **detection latency:** the time difference between error manifestation and its detection.
3) **performance overhead:** the system performance degradation imposed by the error detection method.
4) **area overhead:** the additional surface area required for the error detection resources.
5) **power overhead:** the additional power consumption due to the implementation and function of the error detection mechanism.

4. Description of Evaluated Methods

In this section, we present the main functionality and characteristics of each method that is considered in this paper. All three methods enhance the microprocessor's reliability in the presence of transient and permanent faults.

4.1. Spatial Dual Modular Redundancy (DMR)

DMR is a spatial redundancy technique, in which each hardware component is replicated and computation is repeated in both component copies of the system, to provide reliable operation through redundancy. We evaluate a DMR scheme that consists of two single core processors which share the same memory subsystem. Both cores are executing the same instructions concurrently. As shown in Fig. 2a, each instruction of the main core (green segments) is executed concurrently with the corresponding same instruction of the redundant core (yellow segments). In this scheme, both transient and permanent errors are detected by comparing the instruction results from the two processors.

4.2. Redundant Simultaneous Multithreading (R-SMT)

R-SMT [31, 50, 58, 66, 71, 74], is a class of time redundancy fault-tolerant technique, in which redundant execution is taking place into two different SMT threads running on the same processor core, and redundantly executing the same instructions. Fig. 2b shows the main and redundant instructions from the two threads interleaving execution within the core. Then, the two SMT threads validate the instruction results for error detection. To do so, instruction results of the primary thread are stored to a hardware buffer (named *comparison buffer*) and consumed by the redundant thread which subsequently utilizes them to perform result comparisons. When the comparison buffer is full, the primary thread stalls (*full comparison buffer stalls*), until redundant thread consumes some entries. Similarly, when the buffer is empty, the redundant thread stalls, until it gets filled again.
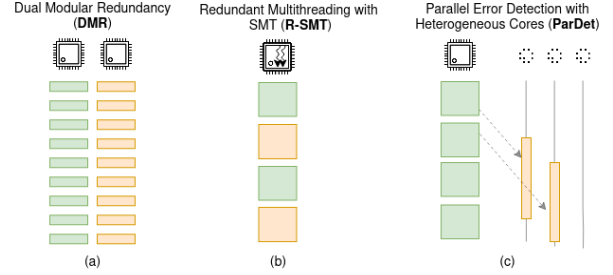

Figure 2: High-level overview of state-of-the-art hardware detection methods. Green segments represent the main execution, while yellow ones the redundant execution for error detection.

4.3. Parallel Error Detection with Heterogenous Cores (ParDet)

ParDet [2] is a microarchitectural method to retain low area, power, and performance overheads. ParDet parallelizes fault detection by combining the main processor with auxiliary low power processor cores, that redundantly repeat the same instructions with the main core.

Specifically, the execution on the main core is segmented into parts, each consisting of consecutive instructions. Each segmented part after being executed on the main core is offloaded to one auxiliary (checker) core for re-execution (as shown in Fig. 2c), along with the architectural state before (starting state) and after (ending state) that part's execution. Each checker core is initialized with the starting architectural state and executes the current segment. If no errors have occurred after the execution of the segment's instructions, its architectural state is expected match the provided ending state, so errors are detected by comparing these two architectural states. This way, program segments can be verified independently and in parallel across multiple low-power auxiliary cores (as illustrated in Fig. 2c with the overlap of yellow segments in the y-axis, which represents time). The process of copying the starting and ending architectural states is called *checkpointing*. Finally, all values that are read and written by the main core are duplicated in a hardware buffer (*load-store log*), which checker cores can access. The load-store log is split in segments, the number of which is equal to the number of checker cores. Offloading to a checker core is performed, when the segment of the load-store log that corresponds to that checker core is completely filled, or after a certain number of instructions.

5. Methodology

We evaluate the three methods in the gem5 simulator [14], implementing DMR and R-SMT and using the original artifact [4] for ParDet. We choose a representative set of 9 benchmarks from MiBench suite [34]. MiBench has been widely used by prior fault tolerance and reliability assessment studies [21, 25, 63–65], as it consists of realistic workloads with reasonable execution time, therefore enabling the thousands of executions required for such a comprehensive

fault injection experiment, to complete within reasonable simulation time. Additionally, the workloads we choose span a wide variety of different application domains, as presented in Table 1, suitable for our application-specific analysis.

TABLE 1: BENCHMARKS APPLICATION DOMAINS AND CLASSES.

Benchmark	Application domain	Second-level priority class
dijkstra	robotics [83]	area/power-critical
djpeg	medical imaging [86], re-mote sensing	latency-critical, area/power-critical
fft, ifft	satellite communications, medical [17], HPC [12]	performance-critical
patricia	satellite communications, avionics	latency-critical, area/power-critical
qsort	automotive, industrial control [34]	latency-critical, performance-critical
sha	IoT [10]	area/power-critical
edges, smooth	medical imaging [34], autonomous driving [76]	latency-critical, performance-critical

Table 2 summarizes the system configuration we use in our evaluation experiments. We model a medium- to high-end processor with the ARMv8 ISA. The configuration we use resembles closely real-world CPU configurations of modern critical systems from various domains, like i.e. the ARM Cortex A72 [7], used in automotive [85]. Equally performant cores (also featuring other ISAs) are also used space-grade processors [42], like i.e. Gaisler NOEL-V [28].

We conduct a statistical fault injection (SFI) experiment. For each benchmark, 1000 single-bit transient (which appear only for a finite period of time) and permanent faults (stuck-at faults) are injected into the register files, randomly generated following a uniform distribution. According to the widely adopted methodology from [48], this is equal to approximate 4% error margin with 95% confidence level. Modelling single-bit transient and permanent faults covers sufficiently the whole range of the studied application domains, since both error types appear due to scaling [67] and under heavy radiation [19], hence occur in all the reviewed use-cases.

In our experiments, we assume that every large SRAM array (e.g., cache memories) is protected by parity or ECC schemes [27, 43, 73, 95] and we inject faults only in the register file. The register file is a critical component in

TABLE 2: SYSTEM CONFIGURATION.

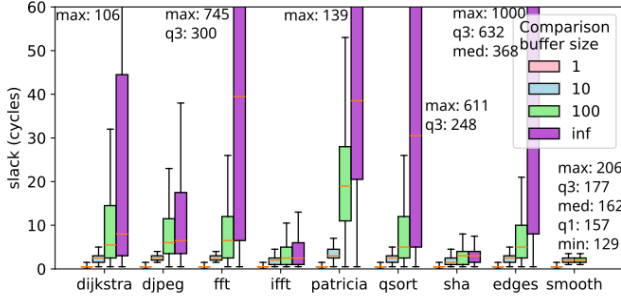| Common options for all methods	
CPU[a]	3-way out-of-order, 2GHz, 192-entry ROB, 64-entry IQ, 128 int + 128 float registers
L1 Dcache	32kB, 8-way, 2-cycle hit latency
L2 cache	2MB shared, 16-way, 20-cycle hit latency
R-SMT specific options	
Comparison buffer size	10 entries
ParDet specific options	
Checker cores	12 cores @ 1GHz, 4 stage, in-order
Load-store log size	36KiB

[a]for ParDet this refers to the main core.

modern processors, responsible for storing and manipulating data during instruction execution. Faults in the register file can significantly affect the correctness and reliability of program execution, potentially resulting in erroneous outcomes, and impacting the entire pipeline stages. While focusing on a specific component, our evaluation indirectly assesses the overall resilience of the fault detection methods we consider, in the context of the entire pipeline. The reason is that the physical register file is not protected by any scheme due to strict performance requirements, as it is located in the critical path of the processor pipeline. This is not the case for other performance-critical hardware components, such as the load/store queue, TLBs, L1 cache memories, etc., which are usually protected, typically by a parity scheme.

Fault Injection. For assessing detection efficiency, we classify the outcome of each fault injection as (i) *detection*: if the fault mitigation technique successfully detects and reports the error, (ii) *mask*: if the execution is completed successfully and the program results were correct, (iii) *silent data corruption (SDC)*: if the execution is completed successfully but produced erroneous results, (iv) *crash*: if the benchmark crashed and (v) *hang*: if the simulation does not terminate within three times the normal execution time.

Detection latency is measured as the elapsed time -in clock cycles- between the fault injection and the detection, while the overhead in the processor performance is evaluated with the Instructions Per Cycle (IPC) metric, in fault-free execution.

The area overheads are calculated based on relevant literature works. Area modelling simulators, like i.e. McPAT [49], provide only coarse-grained area estimations. RTL simulators on the other, can both perform fault injection and calculate more accurate area overheads, but have significantly slower simulation times, and therefore do not allow for the multiple simulation runs necessary for SFI to reach high confidence. Instead, by assessing reliability in the microarchitectural level using SFI on gem5 and basing our area analysis on previous literature, we are able to both accurately access reliability and obtain precise area measurements from real implementations. However, we do utilize McPAT to obtain power consumption estimations, since power depends highly on the workload and this would not be captured through power estimations from literature.

6. Evaluation

6.1. Analysis of Detection Latency

6.1.1. Re-execution slack in R-SMT. For R-SMT, we define as re-execution slack (commit slack) the delay between the retirement of an instruction by the primary thread and the retirement of the same re-executed instruction by the redundant thread. To thoroughly analyze the detection latency of R-SMT, it is necessary to quantify first the re-execution slack, given it significantly impacts detection latency. In Fig. 3, we measure the re-execution slack for various configurations, when varying the mechanism's comparison buffer size. We find that smaller buffer sizes

Figure 3: Distribution of re-execution slack when varying the comparison buffer size in R-SMT.

Figure 4: Distribution of detection latency across multiple injection experiments for all evaluated methods.

Figure 5: Detection efficiency in transient errors.

result in smaller divergence between the primary and redundant thread during execution. This is because when the primary thread has placed more instruction results in the buffer than the redundant is able to consume, and the buffer becomes fully occupied, instruction commit of the main thread halts (named as *full comparison buffer stalls*), since there are no available entries to store the subsequent instruction results. Consequently, the redundant thread is allowed to commit instead, thus converging with the primary one. Given that with smaller buffer sizes, the buffer is more frequently filled, and in turn the primary thread is more frequently denied committing, smaller comparison buffer sizes eventually lead to smaller re-execution slack.

6.1.2. Error detection latency. In Fig. 4, we measure the error detection latency, which yields the same trends across all benchmarks: DMR demonstrates the lowest minimum (in all benchmarks) and mean (in 8 out of 9 benchmarks) latency, because the primary and redundant instructions are concurrently executed in each core. R-SMT exhibits higher latency, because the re-execution slack introduces an *additional* source of delay in the error detection, given that both the main execution and the redundant execution of an instruction must first commit before error detection can occur. Nonetheless, R-SMT demonstrates higher mean latency compared to DMR, but with close values in all benchmarks. Specifically, in 4 out of 9 (*fft, ifft, qsort, smooth*) benchmarks the deviation from DMR is less than 20 cycles and for the rest, never exceeds 240 cycles (*djpeg*). As shown in the previous experiment, with the given comparison buffer size (10 entries), slack is consistently lower than ten instructions, justifying the marginal latency increase observed in this experiment. ParDet consistently exhibits higher min and mean latency in all benchmarks, with up to two orders of magnitude larger compared to DMR (42x higher average for *patricia*): for an error to be detected in ParDet, the whole segment needs to be first re-executed in the checker cores, which have considerably much lower compute capabilities than the main core. Moreover, check-pointing before offloading segments in a checker core, incurs additional latency overheads.
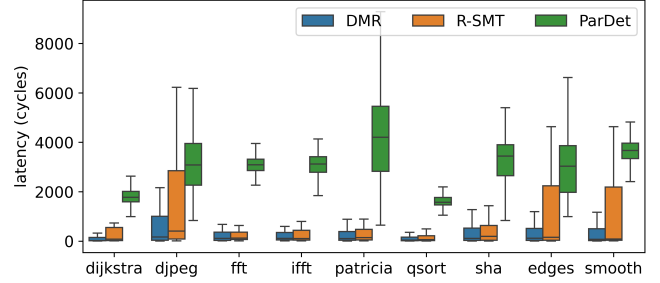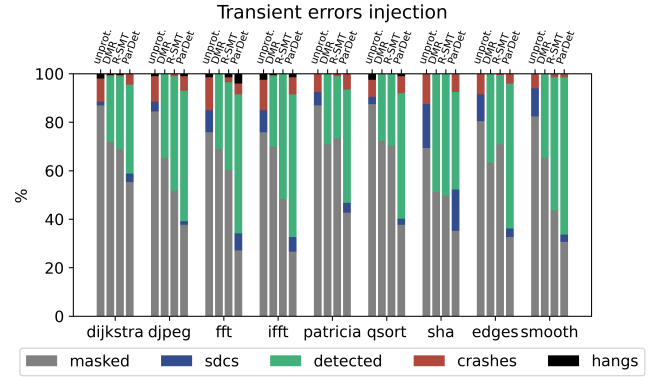
Key insight 1: For R-SMT, slack between threads is the key factor that causes the high detection latency.

Key insight 2: Although ParDet might have the potential to detect errors early, since it performs architectural state comparisons in re-executions, these comparisons happen infrequently due to the low compute capabilities of the checker cores, thus detection latency is consistently larger than DMR and R-SMT.

6.2. Analysis of Detection Efficiency

To assess the effectiveness of error detection in each method, we also inject errors to an unprotected system to use as a baseline.

Fig. 5 compares the error detection efficiency in the presence of transient faults. We observe that ParDet detects the largest number of errors across all benchmarks (52% on average, compared to 39% for R-SMT and 34% for DMR). This is because ParDet's implementation of architectural state comparison for error detection, which detects cases where the injected erroneous register will not affect any instruction/the execution, i.e. when the erroneous register is overwritten. In contrast, DMR and R-SMT correctly classify such cases as masked, since no erroneous result is produced. Masked injections occur both in the unprotected and protected designs, because the erroneous register is

overwritten, given that the fault injection can occur in un-mapped registers (it depends on the fault masks generation, which follows the uniform distribution according to [48]), so the value of that register will never be used.

> **Key insight 3:** High detection percentage does not guarantee low crash percentage, as methods such as ParDet might suffer from *overdetection*.

Overdetection, where benign errors that will be subsequently masked are identified as detected, creates false positives, which is generally unappealing. For instance, in systems where errors are also corrected after detection, false positives lead to unnecessary corrective actions, which introduce additional performance overhead and increase response time.

R-SMT detects more faults compared to DMR, owned to the structure of the injection campaign: to fairly compare fault detection capabilities, the same errors are injected in all methods, and to better approximate the response of fault detection in real-world execution scenarios, each injection is *synchronized* to occur for all methods when the *same* time interval has elapsed (sampled from a normal distribution and measured from the start of the program execution). This results in errors manifesting earlier in the program order in R-SMT, thus causing a higher probability that the error will propagate to other registers via data dependencies. Specifically, R-SMT occurs a higher probability to detect errors, as errors potentially corrupt more instructions.

Despite DMR exhibiting the lowest detection percentage, it experiences the fewest failures (0.1% on average, crashes and hangs combined), followed by R-SMT (1.1%) and then ParDet (6.2%). This is because R-SMT and ParDet have higher detection latency than DMR, which might cause the error to propagate deeper, and in turn causing failures (crashes and hangs) more frequently than DMR.

Lastly, neither DMR nor R-SMT produces SDCs, because for an erroneous value to be written in program output, it must first be produced in some instruction result and consequently, would have been detected.

Fig. 6 compares the efficiency in permanent error detection. All methods experience more crashes and hangs, since permanent errors propagate more, due to their persistent nature, i.e., affecting more instructions. This, combined with the increased detection latency of ParDet renders it the method with the lowest permanent error detection efficiency. In contrast, DMR and R-SMT demonstrate higher detection efficiency, because permanent errors through their wider propagation more likely corrupt instruction results, thus making them detectable by these methods.

> **Key insight 4:** R-SMT despite duplicating hardware, can effectively detect permanent errors, by detecting altered instruction results before and after the permanent hardware corruption.
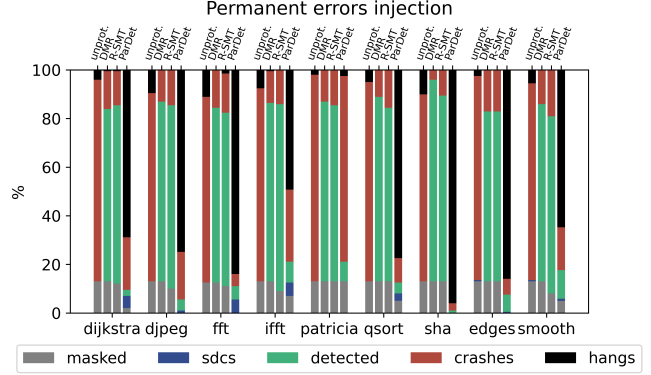

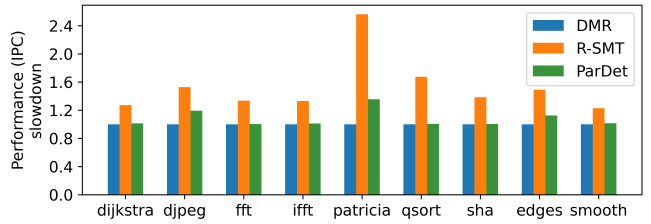**Figure 6: Detection efficiency in permanent errors.**

Figure 7: Comparison of performance in error-free execution for all 3 methods.

> **Key insight 5:** Low detection latency is even more crucial in permanent error detection, since it provides higher detection efficiency.

6.3. Performance Analysis

Fig. 7 presents the performance overhead each method introduces to the execution using the IPC metric. DMR emerges as the most performant across the three, providing on average 1.53x and 1.08x better peformance than R-SMT and ParDet, respectively, as it introduces no performance degradation from normal (unprotected) execution. ParDet experiences a performance slowdown less than 2% in all benchmarks except from *djpeg*, *patricia* and *edges*: checkpointing causes more frequent stalls of the main core in these benchmarks. In all benchmarks, ParDet is more performant than R-SMT by 1.4 times on average. Lastly, the performance overhead of R-SMT is due to two reasons: i) the performance degradation of SMT (2-thread execution) compared to single threaded execution, and ii) the performance loss due to full comparison buffer stalls. Fig. 8 shows the impact of full comparison buffer stalls of R-SMT scheme for various comparison buffer sizes. We find that larger sizes improve performance by causing more infrequent full comparison buffer stalls of the primary thread.

6.4. Area Cost Analysis

In Table 3 we compare the area costs of all three methods over the unprotected baseline that does not integrate fault
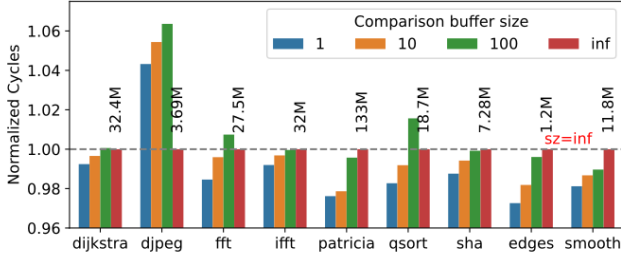
Figure 8: Slowdown of R-SMT in error-free execution, when varying the size of the comparison buffer.

Figure 9: Power consumption for all 3 methods, normalized to the unprotected design.

detection schemes. For DMR, we assume a two-times area overhead, since all hardware components of the core are replicated. Given that area modelling simulators, i.e. McPat [49], do not estimate the area overhead of SMT, we evaluate the area overheads of R-SMT by relevant literature [45, 53, 68], in order to also obtain more accurate results: the layout overhead of SMT in real designs is less than 6% of the core's area, excluding cache memories, and the comparison buffer of 10 entries corresponds to 0.125% of the L1 cache thus contributing to an additional 0.04% area increase [69]. For ParDet, we use the results from the original publication [2] (since we are also modelling the same microarchitecture), i.e., incurring 24% ($0.14mm^2$/core and $0.08mm^2$ for the 36KiB load-store log) overhead over the main core.

TABLE 3: Area Overheads.

Method	Overhead per Component		Total area
	Component	*Overhead*	
Unprotected	-	-	1x
DMR	Redundant core	100%	2x
R-SMT	SMT overhead	6%	1.0604x
	Comparison buffer (10 entries)	0.04%	
ParDet	Checker cores (12)	20.2%	1.24x
	Load-store log (36 KiB)	3.8%	

Key insight 6: R-SMT configurations which maintain small slack will harness multiple benefits: without needing a large comparison buffer, will enjoy both low area overhead and better performance, due to infrequent stalls.

6.5. Power overheads analysis

Fig. 9 shows the total power consumption of all methods across all benchmarks, estimated using McPAT and normalized to the power consumption of the unprotected design, which consists of a single-core processor. DMR exhibits on average power overhead of 43%, given that introduces an additional core and duplicate instructions. In contrast, R-SMT demonstrates the lowest power increase among all methods, with only 15% increase over the unprotected baseline. This minimal increase is due to maintaining the same core configuration for all methods, with the power
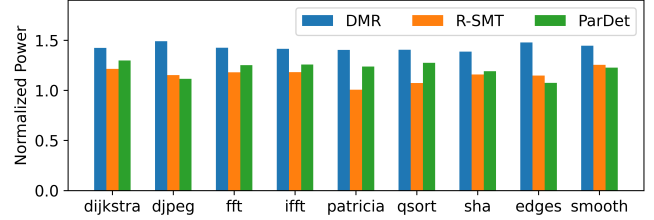
increment attributed only to the execution of the redundant SMT thread.

If the processor microarchitecture were to be more aggressive to support SMT initially (e.g., with larger ROB or register files, etc.), a further increase in power consumption would be anticipated. However, since the redundant thread does not significantly congest the main thread due to re-execution slack, we retained the same microarchitecture for R-SMT as well, not augmenting it with more resources for SMT.

Lastly, ParDet incurs a 21% power overhead, due to the additional power consumption of the checker cores.

Since McPAT provides only coarse estimations for area and power, we expect more accurate estimations of power to yield even lower results, when using more accurate and recent models tailored to current technology.

6.6. Overall Evaluation

In Table 4, we evaluate various configurations of R-SMT and ParDet, each having a different area cost, using all metrics and present averaged numbers across all benchmarks. For R-SMT, we modify the size of the comparison buffer. For ParDet, we adjust the number of checker cores, scaling the load-store log such that the size that corresponds to each core remains constant in every configuration, since otherwise detection latency would be affected primary by that.

ParDet with 3 auxiliary cores and all configurations of R-SMT exhibit similar area overheads. However, due to the low core count in checker cores, ParDet achieves lower performance over R-SMT by nearly a factor of two (2.92x slowdown compared to 1.53x) and higher error detection latency by a factor of 20 (96-149 cycles compared to 3011 cycles). Additionally, ParDet experiences 6 times more crashes, with the increased detection percentage attributed to overdetection (detecting also benign errors which would be eventually masked). Thus, in terms of area cost, R-SMT provides a good trade-off across all metrics.

Key insight 7: R-SMT is more suitable for area-critical applications.

In ParDet configurations, increasing the core count reduces the detection efficiency, because with more cores, the main one progresses further, and errors propagate further without being limited when the main core stalls.

9

TABLE 4: Overall evaluation for various configurations

Method		R-SMT	R-SMT	R-SMT	ParDet	ParDet	ParDet	ParDet	DMR
Configuration		1-entry	10-entry	100-entry	3 cores	6 cores	12 cores	16 cores	-
Area overhead		6.004%	6.04%	6.4%	6%	12%	24%	32%	100%
Detectability	detected %	37.7	38.9	33.7	53.8	52.6	52.2	51.9	34.7
	SDC %	0	0	0	5.2	5.9	5.4	5.6	0
	masked %	62.2	60	63.2	35.1	35.6	36.2	36	65.3
	crashes %	0.1	0.9	1	5.9	5.5	5.4	4.2	0
	hangs %	0	0.2	2.1	0	0.4	0.8	2.3	0
Performance slowdown		1.54	1.53	1.52	2.92	1.56	1.08	1.06	1
Latency (cycles)		96	149	108	3011	3133	3136	3149	96

However, crashes do not increase, because checkpoints continue to occur at the same frequency, as the segment size per core remains constant.

> **Key insight 8:** ParDet presents a tradeoff between performance and detection efficiency versus latency and area.

R-SMT configurations with varying comparison buffer size, perform similarly in detection efficiency and detection latency. This is because error detection continues to occur as early as possible during the redundant thread's commit phase.

> **Key insight 9:** Both R-SMT's detection efficiency and latency are independent of area configuration (comparison buffer size), thus hardware designers can safely select the lowest-area configuration without compromising neither.

> **Key insight 10:** ParDet with increased core counts is suitable for performance-critical applications, since along with adequate detection efficiency has minimal performance overheads.

> **Key insight 11:** DMR or R-SMT with the lowest comparison buffer sizes are equally good options for latency-critical applications, since both provide the minimal latency.

7. Other Related Work

7.1. Error Detection at Other Architecture Levels

A couple of prior works [30, 72] propose hybrid software-hardware and software-only error detection methods. In this work we focus on a comprehensive analysis of only hardware error detection methods, as opposed to i.e. software-level detection. This focus is crucial, because the selection of any hardware method impacts heavily the late stages (where the impact on area and power is more accurately computed) albeit must be made during the early design stages and cannot be altered afterwards. Consequently, the insights we present impact significantly all stages of hardware development. Additionally, error detection is a necessity in modern computing systems, thus manufacturers have widely integrated error detection schemes in hardware. Therefore, our work focuses in effectively evaluating hardware-level methods that have higher detection efficiency and typically better performance than software-based schemes.

7.2. Error Correction

Several prior works [3, 35, 56, 57, 88, 94] propose error correction methods, i.e., schemes to effectively resolve errors after they have been detected. Error correction is orthogonal to error detection: a self correcting system has to first detect errors prior to correcting them, and the overhead of correction typically dominates that of detection, as restoring the system to a safe state requires additional processing [3, 88]. For this reason, a comparison between both correction and detection would not be fair. Instead, we leave the characterization of error correction methods for future work, as they will potentially require different metrics than the ones proposed here, again depending on the requirements of the applications utilizing correction.

7.3. Other Characterization Studies

Prior characterization studies between hardware methods (for detection or also correction) include only limited metrics. One study [57] assesses scrubbing, a hardware correction technique for memory faults, in terms of reliability. [35] compares the latency of hardware redundancy schemes for error correction, [94] compares the reliability of detection techniques for GPUs and [56] discusses the energy efficiency of fault tolerant methods for parallel HPC systems. Reis et al. [72] propose a hybrid software-hardware method evaluated in performance, reliability, and area in comparison to a software-only method. Gizopoulos et al. [30] discuss hardware and software methods qualitatively comparing them in performance, reliability, detection latency and area overhead, but without evaluating them experimentally in neither of these metrics. In this work we focus on multi-metric evaluation of hardware-level methods, covering all the important metrics needed across all application domains.

8. Conclusion

We presented a comprehensive characterization of three diverse alternatives on hardware error detection. We identified that safety-critical applications from different domains can be classified in three categories based on their second requirement beyond reliability. We proposed

five key metrics on which fault detection methods need to be evaluated to effectively cover all various application classes. We extensively compared three methods, i.e., (i) DMR a conventional fault tolerance method, and (ii) R-SMT and (iii) ParDet two microarchitectural approaches, in all the proposed metrics using various workloads. Our work demonstrates that microarchitectural methods achieve comparable detection efficiency with the conventional one, and proposes that (i) R-SMT can be used in area/power-critical applications, since it provides low area and power without compromising detection efficiency in latency, (ii) both R-SMT and DMR are suitable for latency-critical applications and (iii) ParDet fits very well for performance-critical applications.

References

[1] D. Agiakatsikas, G. Papadimitriou, V. Karakostas, D. Gizopoulos, M. Psarakis, and C. Belanger-Champagne, "Impact of voltage scaling on soft errors susceptibility of multicore server cpus," in *2023 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Los Alamitos, CA, USA: IEEE Computer Society, nov 2023, pp. 957–971.

[2] S. Ainsworth and T. M. Jones, "Parallel error detection using heterogeneous cores," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 338–349.

[3] ——, "Paramedic: Heterogeneous parallel error correction," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 201–213.

[4] ——, "Research data supporting "Parallel Error Detection Using Heterogeneous Cores"," https://www.repository.cam.ac.uk/handle/1810/277417, Jun. 2018.

[5] A. Ansari, S. Feng, S. Gupta, and S. Mahlke, "Necromancer: enhancing system throughput by animating dead cores," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, p. 473–484, jun 2010. [Online]. Available: https://doi.org/10.1145/1816038.1816024

[6] ARM, *Cortex-R5 Technical Reference Manual*, 2011, no. r1p2. [Online]. Available: https://developer.arm.com/documentation/ddi0460/d

[7] *Arm Cortex-A Processor Comparison Table*, ARM, 2023, v. 0500. [Online]. Available: https://developer.arm.com/documentation/102826/latest/

[8] T. Austin, "Diva: a reliable substrate for deep submicron microarchitecture design," in *MICRO-32. Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*, 1999, pp. 196–207.

[9] Australian Transport Safety Bureau, *In-flight upset - Airbus A330-303, VH-QPA, 154 km west of Learmonth, WA, 7 October 2008*.

[10] I. L. Azevedo, A. S. Nery, and A. d. C. Sena, "A SHA-3 Co-Processor for IoT Applications," in *2020 Workshop on Communication Networks and Power Systems (WCNPS)*, 2020, pp. 1–5.

[11] W. Bartlett and L. Spainhower, "Commercial fault tolerance: A tale of two systems," *IEEE Transactions on dependable and secure computing*, vol. 1, no. 1, pp. 87–96, 2004.

[12] S. Barua, R. K. Thulasiram, and P. Thulasiraman, "High performance computing for a financial application using fast fourier transform," in *Euro-Par 2005 Parallel Processing*, J. C. Cunha and P. D. Medeiros, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 1246–1253.

[13] D. Bernick, B. Bruckert, P. D. Vigna, D. Garcia, R. Jardine, J. Klecka, and J. Smullen, "Nonstop advanced architecture," in *2005 International Conference on Dependable Systems and Networks (DSN'05)*. IEEE, 2005, pp. 12–21.

[14] 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.

[15] N. Bleier, M. H. Mubarik, G. R. Swenson, and R. Kumar, "Space microdatacenters," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 900–915.

[16] M. S. Bonet and L. Kosmidis, "SPARROW: A Low-Cost Hardware/Software Co-designed SIMD Microarchitecture for AI Operations in Space Processors," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 1139–1142.

[17] E. O. Brigham, *The fast Fourier transform and its applications*. Prentice-Hall, Inc., 1988.

[18] R. W. Butler, "A Primer on Architectural Level Fault Tolerance," NASA Langley Research Center, Tech. Rep. L-19403, Feb. 2008.

[19] A. Campbell, P. McDonald, and K. Ray, "Single event upset rates in space," *IEEE Transactions on Nuclear Science*, vol. 39, no. 6, pp. 1828–1835, 1992.

[20] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward exascale resilience," *The International Journal of High Performance Computing Applications*, vol. 23, no. 4, pp. 374–388, 2009.

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

[22] C. Clark and R. Logan, "Power budgets for mission success," in *CubeSat Workshop*, 2011.

[23] W. B. Culbertson, R. Amerson, R. J. Carter, P. Kuekes, and G. Snider, "Defect tolerance on the teramac custom computer," in *Proceedings. The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Cat. No. 97TB100186)*. IEEE, 1997, pp. 116–123.

[24] B. Denby and B. Lucia, "Orbital edge computing: Nanosatellite constellations as a new class of computer system," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 939–954.

[25] B. Döbel, H. Härtig, and M. Engel, "Operating system support for redundant multithreading," in *Proceedings of the Tenth ACM International Conference on Embedded Software*, ser. EMSOFT '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 83–92.

[26] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig *et al.*, "The international exascale software project roadmap," *The international journal of high performance computing applications*, vol. 25, no. 1, pp. 3–60, 2011.

[27] A. Dörflinger, Y. Guan, S. Michalik, S. Michalik, J. Naghmouchi, and H. Michalik, "Ecc memory for fault tolerant risc-v processors," in *Architecture of Computing Systems – ARCS 2020*. Cham: Springer International Publishing, 2020, pp. 44–55.

[28] *GRLIB IP Core User's Manual*, Frontgrade Technologies, Gaisler, 2024, v. 2024.1. [Online]. Available: https://www.gaisler.com/products/grlib/grip.pdf

[29] M. Ghiglione and V. Serra, "Opportunities and challenges of AI on satellite processing units," in *Proceedings of the 19th ACM International Conference on Computing Frontiers*, ser. CF '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 221–224.

[30] D. Gizopoulos, M. Psarakis, S. V. Adve, P. Ramachandran, S. K. S. Hari, D. Sorin, A. Meixner, A. Biswas, and X. Vera, "Architectures for online error detection and recovery in multicore processors," in *2011 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2011, pp. 1–6.

[31] M. Gomaa and T. Vijaykumar, "Opportunistic Transient-Fault Detection," in *32nd International Symposium on Computer Architecture (ISCA'05).* Madison, WI, USA: IEEE, 2005, pp. 172–183.

[32] R. Gong, K. Dai, and Z. Wang, "Transient fault tolerance on chip multiprocessor based on dual and triple core redundancy," in *2008 14th IEEE Pacific Rim International Symposium on Dependable Computing*, 2008, pp. 273–280.

[33] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke, "The stagenet fabric for constructing resilient multicore systems," in *2008 41st IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 141–151.

[34] 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.

[35] Hagbae Kim and K. Shin, "Evaluation of fault tolerance latency from real-time application's perspectives," *IEEE Transactions on Computers*, vol. 49, no. 1, pp. 55–64, Jan. 2000.

[36] C. Han, X. Jia, H. Liao, Y. Miao, Y. LIU, F. Li, and H. Zhang, "Analysis of the characteristics of engineering implementation of NASA's Psyche mission," *Journal of Astronautics*, vol. 44, no. 10, p. 2, 2023.

[37] I. S. Haque and V. S. Pande, "Hard data on soft errors: A large-scale assessment of real-world error rates in gpgpu," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 691–696.

[38] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–10.

[39] K. E. Holbert and L. T. Clark, "Radiation hardened electronics destined for severe nuclear reactor environments," U.S. Department Of Energy, Office of Nuclear Energy, Tech. Rep. DOE-ASU-NE00679, 2 2016.

[40] International Organization for Standardization, *ISO 26262-2:2018, Road vehicles - Functional safety*, 2018, no. 26262–2.

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

[42] A. Keys, J. Adams, R. Ray, M. Johnson, and J. Cressler, "Advanced avionics and processor systems for space and lunar exploration," in *AIAA SPACE 2009 Conference & Exposition.* Pasadena, California: American Institute of Aeronautics and Astronautics, Sep. 2009.

[43] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe, "Multi-bit error tolerant caches using two-dimensional error coding," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007, pp. 197–209.

[44] J. Knight, "Safety critical systems: challenges and directions," in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, 2002, pp. 547–550.

[45] D. Koufaty and D. Marr, "Hyperthreading technology in the netburst microarchitecture," *IEEE Micro*, vol. 23, no. 2, pp. 56–65, Mar. 2003.

[46] C. LaFrieda, E. Ipek, J. F. Martinez, and R. Manohar, "Utilizing dynamically coupled cores to form a resilient chip multiprocessor," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, 2007, pp. 317–326.

[47] B. Q. Le, E. Nhan, R. H. Maurer, R. E. Jenkins, A. L. Lew, H. S. Feldmesser, and J. R. Lander, "Miniaturization of space electronics with chip-on-board technology," *Johns Hopkins Apl Technical Digest*, vol. 20, pp. 50–61, 1999.

[48] 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.

[49] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 469–480.

[50] Y. Ma and H. Zhou, "Efficient transient-fault tolerance for multithreaded processors using dual-thread execution," in *24th International Conference on Computer Design (ICCD 2006), 1-4 October 2006, San Jose, CA, USA.* IEEE, 2006, pp. 120–126. [Online]. Available: https://doi.org/10.1109/ICCD.2006.4380804

[51] A. Manzone, A. Pincetti, and D. De Costantini, "Fault tolerant automotive systems: an overview," in *Proceedings Seventh International On-Line Testing Workshop*, 2001, pp. 117–121.

[52] R. Mariani, B. Vittorelli, and P. Fuhrmann, "Cost-effective approach to error detection for an embedded automotive platform," *SAE Transactions*, pp. 339–350, 2006.

[53] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton, "Hyper-Threading Technology Architecture and Microarchitecture," *Intel Technology Journal*, vol. 6, no. 1, p. 1, Feb. 2002.

[54] G. Maxwell, "Pacemaker reliability: design to explant," in *Annual Reliability and Maintainability Symposium 1995 Proceedings*, 1995, pp. 460–464.

[55] T. McNeil, R. Olmstead, and S. Schafer, "Reliable, fault tolerant control systems for nuclear generating stations," Atomic Energy of Canada Limited, Tech. Rep., 1990.

[56] E. Meneses, O. Sarood, and L. V. Kale, "Assessing energy efficiency of fault tolerance protocols for hpc systems," in *Computer Architecture and High Performance Computing, Symposium on.* Los Alamitos, CA, USA: IEEE Computer Society, oct 2012, pp. 35–42.

[57] S. Mukherjee, J. Emer, T. Fossum, and S. Reinhardt, "Cache scrubbing in microprocessors: myth or necessity?" in *10th IEEE Pacific Rim International Symposium on Dependable Computing, 2004. Proceedings.* Papeete, Tahiti, French Polynesia: IEEE, 2004, pp. 37–42.

[58] S. Mukherjee, M. Kontz, and S. Reinhardt, "Detailed design and evaluation of redundant multi-threading alternatives," in *Proceedings 29th Annual International Symposium on Computer Architecture.* Anchorage, AK, USA: IEEE Comput. Soc, 2002, pp. 99–110.

[59] A. Naithani and L. Eeckhout, "Reliability-aware runahead," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA).* IEEE, Apr. 2022, p. 772–785.

[60] NASA Engineering and Safety Center, *Technical Support to the National Highway Traffic Safety Administration (NHTSA) on the Reported Toyota Motor Corporation (TMC) Unintended Acceleration (UA) Investigation*, Jan. 2018.

[61] M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. F. Cunha, and R. Buyya, "Hpc cloud for scientific and business applications: Taxonomy, vision, and research challenges," *ACM Comput. Surv.*, vol. 51, no. 1, jan 2018.

[62] E. Normand, "Single event upset at ground level," *IEEE Transactions on Nuclear Science*, vol. 43, no. 6, pp. 2742–2750, 1996.

[63] 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)*, 2021, pp. 902–915.

[64] ——, "Avgi: Microarchitecture-driven, fast and accurate vulnerability assessment," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 935–948.

[65] ——, "Silent data corruptions: Microarchitectural perspectives," *IEEE Transactions on Computers*, vol. 72, no. 11, pp. 3072–3085, 2023.

[66] A. Parashar, A. Sivasubramaniam, and S. Gurumurthi, "SlicK: slice-based locality exploitation for efficient redundant multithreading," *ACM SIGPLAN Notices*, vol. 41, no. 11, pp. 95–105, Nov. 2006.

[67] J. C. Pickel, "Effect of cmos miniaturization on cosmic-ray-induced error rate," *IEEE Transactions on Nuclear Science*, vol. 29, no. 6, pp. 2049–2054, 1982.

[68] R. Preston, R. Badeau, D. Bailey, S. Bell, L. Biro, W. Bowhill, D. Dever, S. Felix, R. Gammack, V. Germini, M. Gowan, P. Gronowski, D. Jackson, S. Mehta, S. Morton, J. Pickholtz, M. Reilly, and M. Smith, "Design of an 8-wide superscalar RISC microprocessor with simultaneous multithreading," in *2002 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.02CH37315)*, vol. 1, Feb. 2002, pp. 334–472 vol.1.

[69] P. P. Ramon, "Caching in real-time and embedded systems: Benchmarking the arm cortex-m3 and quark soc x1000 processors," Bsc Thesis, University College Cork, Ireland, Apr. 2015.

[70] F. Rehm, J. Seitter, J.-P. Larsson, S. Saidi, G. Stea, R. Zippo, D. Ziegenbein, M. Andreozzi, and A. Hamann, "The road towards predictable automotive high - performance platforms," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 1915–1924.

[71] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, p. 25–36, May 2000.

[72] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, D. I. August, and S. S. Mukherjee, "Design and Evaluation of Hybrid Fault-Detection Systems," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 2, pp. 148–159, May 2005.

[73] D. Rossi, N. Timoncini, M. Spica, and C. Metra, "Error correcting code analysis for cache memory high reliability and performance," in *2011 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2011, pp. 1–6.

[74] E. Rotenberg, "AR-SMT: a microarchitectural approach to fault tolerance in microprocessors," in *Digest of Papers. Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing (Cat. No.99CB36352)*. Madison, WI, USA: IEEE Comput. Soc, 1999, p. 84–91.

[75] RTCA, Inc. and EUROCAE, *RTCA DO-254 / EUROCAE ED-80, Design Assurance Guidance for Airborne Electronic Hardware*.

[76] Y.-W. Seo, J. Lee, W. Zhang, and D. Wettergreen, "Recognition of highway workzones for reliable autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 708–718, 2015.

[77] Z. Shen, J. Jin, C. Tan, A. Tagami, S. Wang, Q. Li, Q. Zheng, and J. Yuan, "A survey of next-generation computing technologies in space-air-ground integrated networks," *ACM Comput. Surv.*, vol. 56, no. 1, aug 2023.

[78] I. Silva, O. do Espírito Santo, D. do Nascimento, and S. Xavier-de Souza, "Cevero: A soft-error hardened soc for aerospace applications," in *Anais Estendidos do X Simpósio Brasileiro de Engenharia de Sistemas Computacionais*. SBC, 2020, pp. 121–126.

[79] T. Slegel, R. Averill, M. Check, B. Giamei, B. Krumm, C. Krygowski, W. Li, J. Liptay, J. MacDougall, T. McPherson, J. Navarro, E. Schwarz, K. Shum, and C. Webb, "Ibm's s/390 g5 microprocessor design," *IEEE Micro*, vol. 19, no. 2, pp. 12–23, 1999.

[80] R. Smith, S. Bucior, and J. Hahn, "Integration and test challenges of parker solar probe," in *2020 IEEE Aerospace Conference*, 2020, pp. 1–8.

[81] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "The impact of technology scaling on lifetime reliability," in *International Conference on Dependable Systems and Networks, 2004*, 2004, pp. 177–186.

[82] K. Sundaramoorthy, Z. Purser, and E. Rotenberg, "Slipstream processors: improving both performance and fault tolerance," *ACM SIGPLAN Notices*, vol. 35, no. 11, pp. 257–268, Nov. 2000.

[83] S. Sundarraj, R. V. K. Reddy, M. B. Basam, G. H. Lokesh, F. Flammini, and R. Natarajan, "Route planning for an autonomous robotic vehicle employing a weight-controlled particle swarm-optimized dijkstra algorithm," *IEEE Access*, vol. 11, pp. 92 433–92 442, 2023.

[84] A. Tadanki and E. G. Lightsey, "Closing the power budget architecture for a 1u cubesat framework," *Georgia Institute of Technology, Tech. Rep*, 2019.

[85] *TDA4VE TDA4AL TDA4VL Jacinto Processors, Silicon Revision 1.0*, Texas Instruments, 2023, rev. 1.0. [Online]. Available: https://www.ti.com/lit/ds/sprsp62a/sprsp62a.pdf?ts=1717254086175

[86] D. R. Varma, "Managing DICOM images: Tips and tricks for the radiologist," *The Indian Journal of Radiology & Imaging*, vol. 22, no. 1, pp. 4–13, 2012.

[87] O. Vendier, M. Huan, C. Drevofi, J. Cazaux, E. Beyne, R. Van Hoof, A. Marty, S. Pinel, J. Tasselli, S. Marco, and J. Morante, "Ultra thin electronics for space applications," in *2001 Proceedings. 51st Electronic Components and Technology Conference (Cat. No.01CH37220)*, 2001, pp. 767–771.

[88] T. N. Vijaykumar, I. Pomeranz, and K. Cheng, "Transient-fault recovery using simultaneous multithreading," *ACM SIGARCH Computer Architecture News*, vol. 30, no. 2, pp. 87–98, May 2002.

[89] N. Wang and S. Patel, "Restore: Symptom-based soft error detection in microprocessors," *IEEE Trans. on Depend. and Secure Comp.*, 2006.

[90] S. Wang and Q. Li, "Satellite computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 10, no. 24, pp. 22 514–22 529, 2023.

[91] S. Wang, G. Zhang, J. Wei, Y. Wang, J. Wu, and Q. Luo, "Understanding silent data corruptions in a large production cpu population," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 216–230. [Online]. Available: https://doi.org/10.1145/3600006.3613149

[92] C. Weaver and T. Austin, "A fault tolerant approach to microprocessor design," in *Proceedings International Conference on Dependable Systems and Networks*. Goteborg, Sweden: IEEE Comput. Soc, 2001, p. 411–420.

[93] J. R. Wertz and W. J. Larson, *Space Mission Analysis and Design*. Springer Netherlands, Sep. 1999.

[94] H.-J. Wunderlich, C. Braun, and S. Halder, "Efficacy and efficiency of algorithm-based fault-tolerance on GPUs," in *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*. Chania: IEEE, Jul. 2013, pp. 240–243.

[95] D. H. Yoon and M. Erez, "Memory mapped ecc: low-cost error protection for last level caches," in *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*, ser. ISCA '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 116–127.

[96] Y. Zu, C. R. Lefurgy, J. Leng, M. Halpern, M. S. Floyd, and V. J. Reddi, "Adaptive guardband scheduling to improve system-level efficiency of the power7+," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 308–321.