Fault-Tolerant Computing with Heterogeneous Hardening Modes

Florian Kriebel, Faiq Khalid, Bharath Srinivas Prabakaran, Semeen Rehman, and Muhammad Shafique

1 Introduction

Recent technological advancements in the field of transistor fabrication, such as FinFETs and GAAFETs, have led to significant improvements in the performance of next-generation multi-core processors but at the expense of an increased susceptibility to reliability threats such as soft errors [4, 37], aging [14], and process variations [14]. These threats generate permanent and/or temporary faults that can lead to unexpected system failures and can be disastrous to several safety-critical applications such as automotive, healthcare, aerospace, etc., as well as high-performance computing systems. Therefore, several techniques have been proposed to detect, prevent, and mitigate these reliability threats across the computing stack ranging from the transistor and circuit layer [27, 43] to the software/application layer [2, 42, 44]. Oftentimes, (full-scale) redundancy is employed at the hardware and the software layers, for example, at the *software layer*, by executing multiple redundant thread versions of an application, either spatially or temporally, and at the *hardware layer*, by duplicating or triplicating the pipeline, i.e., Double/Triple Modular Redundancy (DMR/TMR) [28, 29, 32, 46]. However, these reliability techniques exhibit several key limitations, as discussed below:

1. Ensuring temporal redundancy at the software layer, by executing multiple redundant threads of a given application on the same core, would incur a significant performance overhead.

F. Kriebel (✉) · F. Khalid · B. S. Prabakaran · S. Rehman · M. Shafique
Technische Universität Wien (TU Wien), Vienna, Austria
e-mail: florian.kriebel@tuwien.ac.at; faiq.khalid@tuwien.ac.at; bharath.prabakaran@tuwien.ac.at; semeen.rehman@tuwien.ac.at; muhammad.shafique@tuwien.ac.at

J. Henkel, N. Dutt (eds.), *Dependable Embedded Systems*, Embedded Systems,
https://doi.org/10.1007/978-3-030-52017-5_7

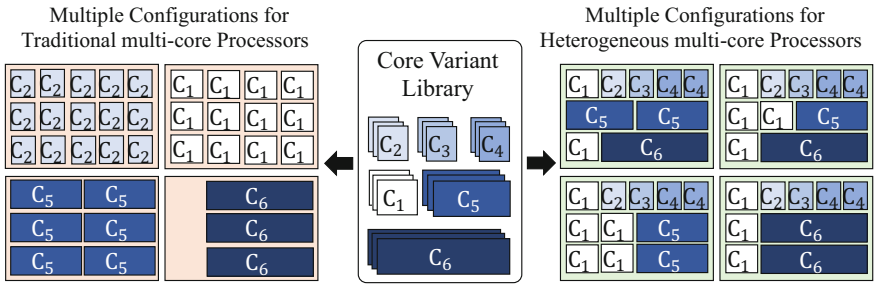

Fig. 1 Different configurations for mitigating dependability threats for traditional (homogeneous) and heterogeneous multi-core systems, respectively

2. Executing multiple redundant threads in multiple cores concurrently, instead of a single core, provides spatial redundancy and nullifies the performance overhead caused by the temporal redundancy. However, due to the activation of multiple cores, this technique incurs a significant power overhead.
3. Similarly, fabricating redundant hardware components to provide full-scale TMR across the pipeline incurs additional area, power, and energy overheads including additional on-chip resources for the data correction and control units.
4. Moreover, these techniques are not adaptive with respect to the dependability requirements of the applications, as well as their inherent error tolerance, during their execution.

To address these limitations, we proposed the **reliability-heterogeneous architectures** in [20, 21, 33, 34]. They offer different types of reliability modes in different cores (i.e., the so-called *reliability-heterogeneous cores*), realized through hardening of different pipeline components using different reliability mechanisms. Hence, such processors provide a foundation for design- and run-time trade-offs in terms of reliability, power/energy, and area. Their motivation arises from the fact that different applications exhibit varying degrees of error tolerance and inherent masking to soft errors due to data and control flow masking. Hence, depending upon the executing applications, their tasks can be mapped to a set of *reliability-heterogeneous cores* to mitigate soft errors, as shown in Fig. 1.

Although this solution significantly reduces the power/energy and performance overheads, it requires a sophisticated *run-time management system* that performs an appropriate code-to-core mapping of the applications, based on their requirements and given power/performance constraints. This requires enabling certain features across the hardware and software layers such as additional control logic, core monitoring units at the hardware layer, and a run-time manager at the software layer. Embedding this chapter's content in the scope of this book and the overall projects [11, 13], the focus of this chapter is limited to the design of such

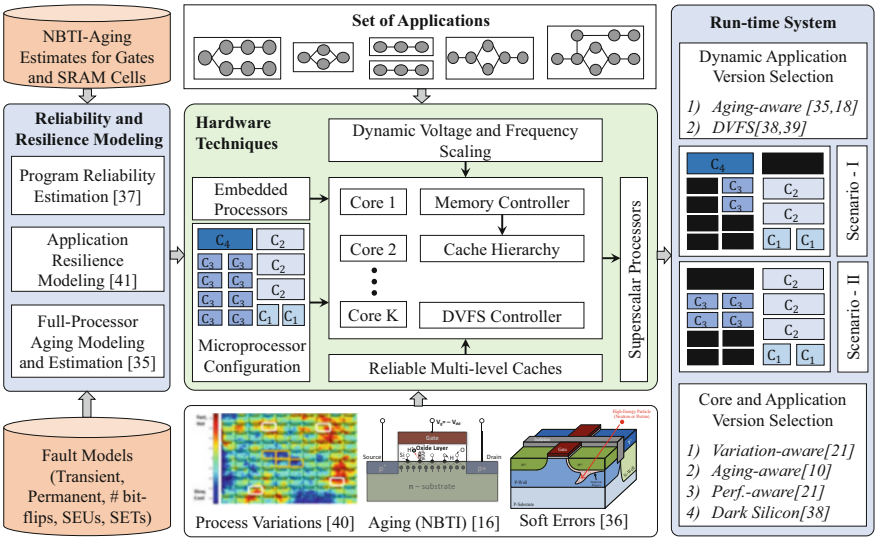

Fig. 2 An overview of dependable computing with heterogeneous hardening modes (adapted from [34]). Image sources: [16, 37]

hardware/software techniques that can enable heterogeneous dependable computing (see Fig. 3).

Typically, the hardware solutions for dependable heterogeneous architectures consist of the following three phases (see Fig. 2):

1. **Reliability and Resilience Modeling:** First, the effects of different reliability threats (i.e., soft errors, aging, and process variations) on different components of a given multi-core system and different applications are modeled and analyzed based on mathematical analysis, simulation, and/or emulation.
2. **Hardware Techniques:** Based on the vulnerability analysis of the previous step, multiple reliability-heterogeneous core variants are developed by hardening a combination of the pipeline and/or memory components. Similarly, an analysis of multi-level cache hierarchies has led to the design of multiple heterogeneous reliability cache variants and reliability-aware reconfigurable caches.
3. **Run-time System:** Afterwards, appropriate task-to-core mapping as well as reliable code version selection are performed, while satisfying the application's reliability requirement and minimizing the power/area overheads. These problems can also be formulated as constrained optimization problems.

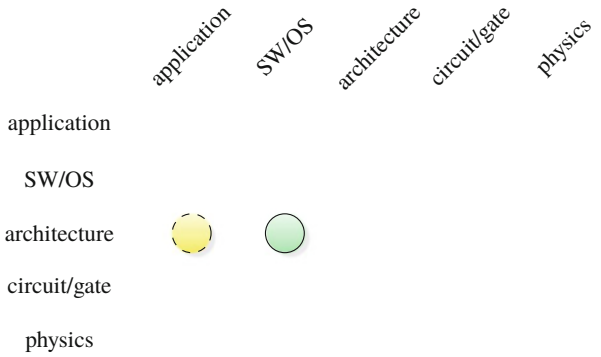

Fig. 3 Main abstraction layers of embedded systems and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions

2 Fault-Tolerant Heterogeneous Processors

Reliability threats not only affect the computing cores in the microprocessors, but can also significantly affect the on-chip memory sub-systems, like multi-level caches. This section provides an overview of our techniques for developing reliability-heterogeneous in-order processors and multi-level cache hierarchies. Unlike the traditional homogeneous dependable processors, the development of reliability-heterogeneous processors not only requires design-time efforts to develop multiple variable-reliability processor variants but also requires a run-time management system that can efficiently cater the applications' requirements (see Fig. 2). Therefore, as illustrated in Fig. 4, developing these hardware techniques can be divided into two phases, namely, design-time and run-time:

1. **Design-Time:** At design-time, first the overall vulnerability of a processor is analyzed. Based on this analysis, we develop hardware techniques that can be used to design reliability-heterogeneous processor cores (see Sect. 2.1). In the next step, these hardened cores are integrated into an architectural-level simulator to evaluate their effectiveness. Similarly, we evaluate the vulnerability of caches, based on which hardware techniques are designed to mitigate the effects of reliability threats in caches (see Sect. 2.2). These reliability-aware caches and multiple reliability-heterogeneous cores are used to design a reliability-heterogeneous processor, as depicted by *Design-Time* in Fig. 4.
2. **Run-time:** To effectively use the reliability-heterogeneous processor, an *adaptive run-time manager for soft error resilience (ASER)* is used to estimate the reliability requirements of the applications (as well as their resilience properties), and to efficiently map their threads to a set of hardened cores while adhering to the user and performance constraints, as depicted by *Run-Time* in Fig. 4.

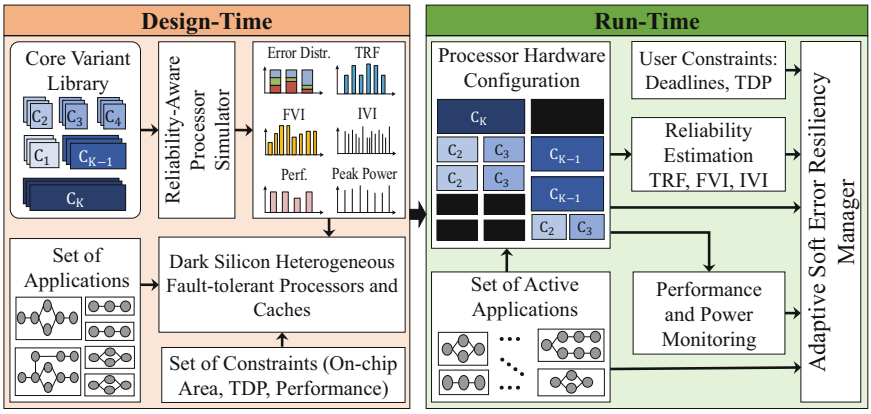

Fig. 4 The design- and run-time methodology to develop dependable heterogeneous processors (adapted from [20, 34])

2.1 Hardening Embedded Processors

To design the hardened cores for reliability-heterogeneous architectures, first, we analyze the vulnerability of these cores to different reliability threats. Based on this vulnerability analysis, instead of enabling full-scale DMR/TMR, we design the micro-architecture of different hardened (in-order) cores. These cores have distinct reliability mechanisms in different pipeline components (ranging from unprotected to fully-protected), but implement the same instruction set architecture (ISA); see the core variant library in Fig. 4. Hence, these cores provide a trade-off between reliability, area, and power/energy consumption. Since not all transistors on a chip can be powered-on at the same time (i.e., the dark silicon problem [7, 31, 41]), we leverage this fact to integrate many different hardened cores to develop a reliability-heterogeneous ISO-ISA processor [20, 21], while adhering to hardware and user-defined constraints (e.g., area, power) considering a target domain (i.e., given a particular set of target applications).

To cater for the application-specific requirements at run-time, an *adaptive run-time manager for soft error resilience (ASER)* determines an efficient application-to-core mapping considering the application's vulnerability and deadline requirements, system performance, thermal design power (TDP), and other user-defined constraints. For example, Fig. 5 depicts the varying reliability improvements of the ASER run-time system approach in comparison with multiple state-of-the-art reliability techniques such as *TRO* (timing dependability optimization aiming at minimizing the deadline misses), *RTO* (optimizing functional as well as timing dependability), *Full-TMR* (activating full TMR), and *AdTMR* (deactivating TMR when the vulnerability lies below a pre-defined threshold). The reliability is measured using the Reliability Profit Function (RPF) which is defined as follows:

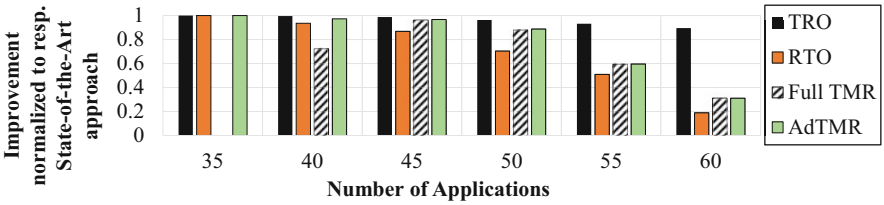

Fig. 5 Reliability Profit Function improvement over different state-of-the-art reliability techniques, i.e., timing reliability optimization (TRO), functional and timing reliability (RTO), TMR, adaptive TMR (adTMR) (adapted from [20])

$$RPF = 1 - \left(\frac{RTP(t)_{ASER}}{RTP(t)_{Z}} \right) \tag{1}$$

where $\forall t \in T$, T is a set of run-time concurrently executing application tasks ($T = \{T_1, T_2, \ldots, T_M)\}$), $Z \in \{TRO, RTO, TMR, adTMR\}$, and RTP is the Reliability-Timing Penalty [38]. Note, a higher value of RPF translates to a better reliability. The ASER approach achieves 58–96% overall system reliability improvements when compared to these four state-of-the-art techniques.

2.2 Reliability Techniques for Multi-Level Caches

In any microprocessor, on-chip memories play a significant role to improve the throughput and performance of an application. Moreover, memory elements (such as caches) are even more susceptible to soft errors compared to the computing elements (i.e., logic) as they occupy a significant portion of the total on-chip area [9]. Therefore, for designing dependable multi/many-core processors, different (individual) cache levels as well as the complete cache hierarchy (considering inter-dependency between different cache levels) have to be analyzed and optimized for mitigating reliability threats.

2.2.1 Improving the Reliability of Last-Level Caches

Dynamic reconfiguration of the caches with respect to the running applications has a significant impact on the vulnerability of the on-chip last-level caches, as shown in Fig. 6. It can be observed from the vulnerability analysis of a given cache configuration (see Fig. 6) that due to different access patterns and occupancy of last-level caches for the application, the vulnerability also varies depending on the executing applications. This dynamic change in vulnerability at run-time can be exploited to improve the reliability of the last-level cache. Therefore, dynamic reconfiguration of the last-level cache is exploited to develop a *reliability-aware*

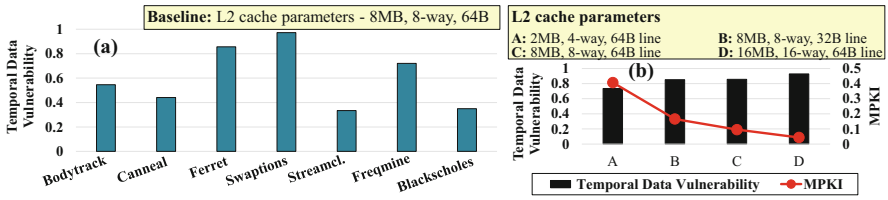

Fig. 6 (**a**) Vulnerability analysis of different applications from the *PARSEC* benchmark for the baseline case (L2 cache parameters—8 MB, 8-way, 64 B). (**b**) Vulnerabilities and cache misses (MKPI) for the *Ferret* application for different cache configurations (adapted from [19])

reconfigurable cache architecture [19, 22]. Towards this, we aim at reducing the vulnerability of concurrently executing applications by employing the following features:

1. A methodology to *quantify the cache vulnerability* with respect to concurrently executing applications.
2. A method for *lightweight online prediction of the application vulnerability online* based on the cache utilization and performance data.
3. A methodology to *dynamically reconfigure the last-level cache* at run-time that targets at minimizing the application vulnerability w.r.t. cache while keeping the performance overhead low, or within a tolerable bound.

This reliability-aware cache reconfiguration [22] can also be applied in conjunction with the error correcting codes (ECCs). For example, Single Error Correcting-Double Error Detecting (SEC-DED) [6] can be combined with the reliability-aware cache reconfiguration [22] to improve reliability in multi-bit error scenarios, or in cases where only some of the cache partitions are ECC-protected due to the area constraints.

2.2.2 Improving the Reliability of the Complete Cache Hierarchy

The application vulnerability towards soft errors is not only dependent on the individual utilization or dynamic reconfiguration of the different individual cache levels (e.g., L1 or L2). Rather, the *vulnerability interdependencies across different cache* levels also have significant impact on the reliability of the system. Therefore, the vulnerability of the concurrently executing applications with respect to the corresponding cache configuration can further be improved by considering these interdependencies across different cache levels. To achieve an efficient design, we first performed an *architectural design space exploration* (DSE), while considering multi-core processors with multiple cache levels executing different multi-threaded applications. Our cache DSE methodology identifies the pareto-optimal configurations with respect to constraints, performance overhead, and targeted vulnerabilities [45]. Afterwards, these configurations are used at run time to

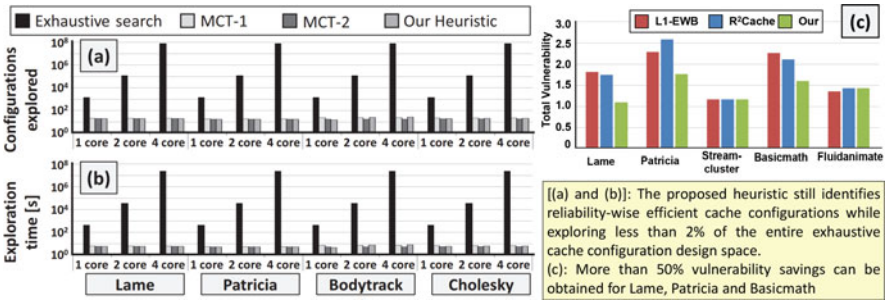

Fig. 7 (**a**) and (**b**) Exploration time saving achieved by the proposed approach with respect to exhaustive exploration, multi-level tuning approach (MCT-1) [47] and heuristic (MCT-2) [47]. (**c**) Vulnerability saving comparison of the proposed approach compared to non-reconfigurable baseline cache with L1 Early WriteBack (EWB) [15] and reliability-aware last-level cache partitioning (R^2Cache) [22] schemes (adapted from [45])

perform *reliability-aware cache reconfiguration for the complete cache hierarchy*. Figure 7 shows that more than 50% vulnerability saving is achieved by the proposed solution as compared to non-reconfigurable baseline cache with L1 Early WriteBack (EWB) [15] and Reliability-Aware Last-Level Cache Partitioning (R^2Cache) while exploring less than 2% of the entire exhaustive cache configuration design space.

3 Heterogeneous Reliability Modes of Out-of-Order Superscalar Cores

Embedded processors, although important in a wide range of applications and scenarios, cannot cater the high throughput and performance requirements of personal computers or high-performance computing platforms such as cloud servers or data-centers, which are also constrained in the amount of power that can be consumed. Such high-throughput systems deploy multi-core out-of-order (O3) superscalar processors, such as Intel Core i7 processors in PCs, and Intel Xeon or AMD Opteron processors in servers and data-centers worldwide. An O3 processor executes the instructions of a program *out-of-order*, instead of in-order as is the case in embedded processors (e.g., LEON3), to utilize the instruction cycles that would otherwise be wasted in pipeline stalls. A *superscalar* processor, on the other hand, implements instruction-level parallelism to execute more than one instruction in parallel by dispatching instructions to multiple different execution units embedded in the processor core. Therefore, an O3 superscalar processor offers a significantly higher throughput by combining the advantages of these two individual techniques. However, enabling such high throughput comes at the cost of implementing additional hardware units such as the Re-order Buffer (ROB), which keeps track of the instructions executing out-of-order.

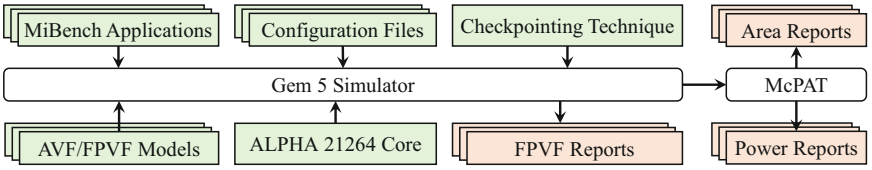

Fig. 8 Experimental tool-flow for vulnerability analysis of out-of-order superscalar *ALPHA* cores

In this section, we analyze the vulnerability of the ALPHA 21264 [17] O3 superscalar processor and design multiple reliability-heterogeneous processor cores from which an optimal configuration can be chosen at run-time based on the applications' reliability requirements.

3.1 Experimental Setup

Figure 8 presents an overview of the tool-flow used to obtain the results. We utilize a modified version of the *gem5 simulator* [5] extended to support the following functionality:

1. Determine the *vulnerable time* of all pipeline components, which in turn is used to compute their *Architectural Vulnerability Factors (AVFs)* [30],
2. *Full support for simulating reliability-heterogeneous cores* obtained by triplicating key pipeline components (instead of implementing full-scale TMR), and
3. *Checkpoint processor state compression* using techniques like DMTCP [3], HBICT [1], and GNU zip [8].

We evaluate our reliability-heterogeneous *ALPHA* 21264 four-issue superscalar processor cores using the *MiBench* application benchmark suite [12].

3.2 Vulnerability Analysis of Out-of-Order Superscalar Processors

The AVF of a component C over a period of N clock-cycles is defined as the probability of a fault that is generated in C to propagate to the final output resulting in an erroneous application output or intermittent termination of the program [30]. It is computed using the following equation:

$$AVF_C = \frac{\sum_{n=0}^{n=N} VulnerableBits}{TotalBits \times N} \tag{2}$$

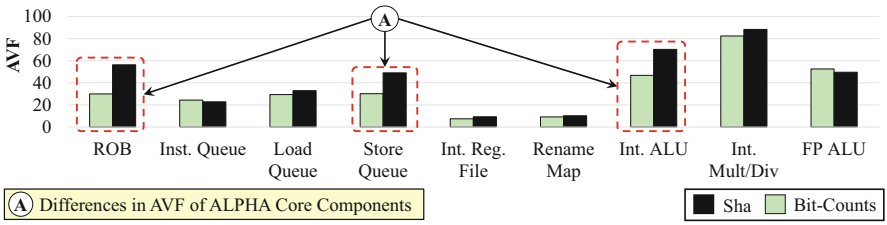

Fig. 9 Differences in AVF of *ALPHA* core components during application execution (*SHA* and *Bit-counts*) (adapted from [33])

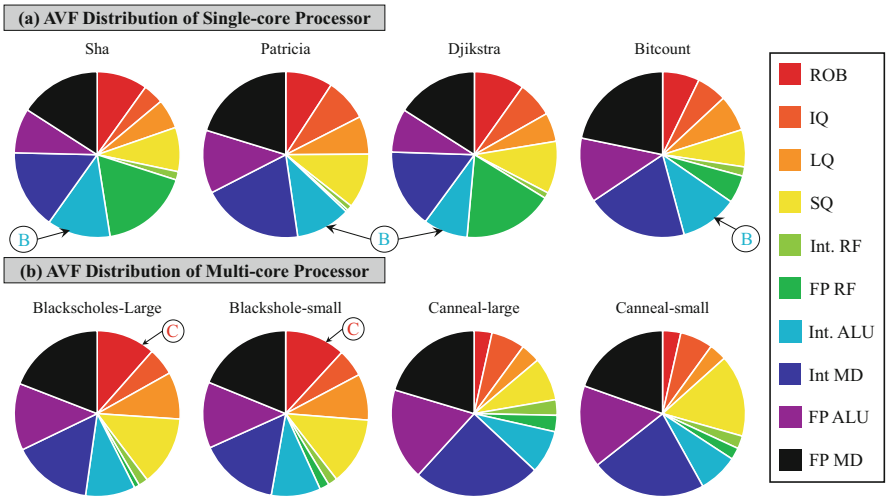

Fig. 10 Vulnerability analysis of *ALPHA* cores in single- and multi-core processors (adapted from [33])

The AVF of each pipeline component is estimated using applications from the *MiBench* and *PARSEC* application benchmark suites for single- and multi-core *ALPHA* 21264 superscalar processors for key pipeline components such as: (1) Re-order Buffer (ROB), (2) Instruction (IQ), (3) Load (LQ), (4) Store Queues (SQ), (5) Integer Register Files (Int. RF), (6) Floating Point Register Files (FP RF), (7) Rename Map (RM), (8) Integer ALUs (Int. ALU), (9) Floating Point ALUs (FP ALU), (10) Integer Multiply/Divide (Int. MD), and (11) Floating Point Multiply/Divide (FP MD). Figures 9 and 10 illustrate the results of the vulnerability analysis experiments for both the single-core and multi-core processors.

We analyze the results obtained from the vulnerability analysis to make the following **key observations**:

1. We have identified three key pipeline components (Integer ALU, Store Queue, and Re-order Buffer) that are more vulnerable during the execution of *SHA*, when compared to *Bit-counts*, as depicted by *A* in Fig. 9.

2. The AVFs of the individual pipeline components vary for different application workloads. For example, as shown in Fig. 10a the vulnerability of the Integer ALU widely varies for the four application workloads evaluated (labeled *B*).
3. In case of multi-core processors, the size of the input data does not significantly affect the AVF of the pipeline components, as shown by *C* in Fig. 10b.

The AVF of a component varies based on the type and number of instructions present in the application and its properties such as its compute- or memory-intensiveness, instruction-level parallelism, cache hit/miss rate, etc. For example, components like the ROB and the SQ are more vulnerable in *SHA* because of higher levels of instruction-level parallelism and more store instructions.

Therefore, based on this information, we can select certain key pipeline components that can be hardened/triplicated to increase the reliability of the processor for a given application workload. By hardening multiple key pipeline components in different combinations, we design a wide range of reliability-heterogeneous O3 superscalar *ALPHA* cores from which an optimal design configuration can be selected at run-time based on an application's reliability requirement while minimizing the area and/or power overheads.

3.3 Methodology for Hardening Out-of-Order Superscalar Processors

Our methodology for designing reliability-heterogeneous O3 superscalar processors targets two key approaches: (1) Redundancy, and (2) Checkpointing. Redundancy at the hardware layer is ensured by designing a wide range of reliability-heterogeneous processor cores by hardening a combination of the vulnerable pipeline components, depending on the reliability requirements of the target application. The vulnerable components are selected based on the fault-injection experiments and the AVF values of each component for different application workloads. Second, to further enhance processor reliability, we investigate and analyze various compression mechanisms that can be used to efficiently reduce the size of checkpointing data. An overview of our methodology for hardening O3 superscalar processors is presented in Fig. 11. First, we explain how we evaluate the vulnerability of the full processor for a given application workload.

3.3.1 Full-Processor Vulnerability Factor

For evaluating the vulnerability of the full processor for a given application workload, we propose to extend the AVF to estimate what we refer to as the **Full-Processor Vulnerability Factor (FPVF)**. It is defined as the ratio of the total number of vulnerable bits (*VulnerableBits*) in the processor pipeline for the duration they are vulnerable (*VulnerableTime*) to the total number of bits in the processor

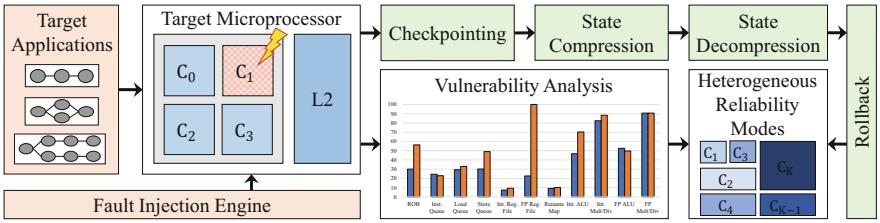

Fig. 11 Methodology for hardening out-of-order superscalar processors (adapted from [33])

pipeline (*TotalBits*) for the total duration of application execution (*TotalTime*). For a given application workload (W), we estimate FPVF of our proposed reliability-heterogeneous processors as:

$$FPVF_W = \frac{\sum_{\forall i \in Components} VulnerableBits_i \times VulnerableTime_i}{\sum_{\forall i \in Components} TotalBits_i \times TotalTime_i} \tag{3}$$

3.3.2 Heterogeneous Reliability Modes for ALPHA Cores

Enabling full-scale TMR for all application workloads leads to 200% (or more) area and power overheads, which might not be a feasible option in many real-world systems. Considering the analysis presented in Sect. 3.2, which illustrates that the AVF of the pipeline components varies based on the application workload, *we propose to enable fine-grained TMR at the component-level*. This involves hardening a combination of highly vulnerable pipeline components, instead of the full-processor pipeline to increase processor reliability while reducing the power and area overheads associated with TMR. Hardening involves instantiating three instances of the component with the same set of inputs and a voter circuit that is used to elect the majority output. We propose and analyze 10 different reliability modes (RM) for heterogeneous processors, including the baseline unprotected (U) core. The list of components hardened in these modes are presented in Table 1.

Next, we execute the four *MiBench* application benchmarks on our 10 proposed RMs to estimate the FPVF of each reliability-heterogeneous processor. We also evaluate the area and power overheads incurred by each reliability mode. The results of the experiments are illustrated in Fig. 12. From these results, we make the following **key observations:**

1. Our initial hypothesis, which stated that hardening different combinations of pipeline components (RMs) can reduce the vulnerability to different extents based on the application workload being executed, was correct. We demonstrate this further by considering the applications *SHA* and *Dijkstra*. Typically, the vulnerability of these two applications is similar to each other, except in the cases

Table 1 Heterogeneous reliability modes and corresponding pareto-optimal reliability modes for *MiBench* applications

Reliability mode	Components hardened	Application	Pareto-optimal reliability modes
U	Unprotected	Bit-counts	U, RM4, RM7
RM1	RF	Dijkstra	U, RM4, RM7, RM8
RM2	IQ, RM	Patricia	U, RM4, RM7
RM3	IQ, LQ, SQ	SHA	U, RM1, RM6, RM7, RM8
RM4	IQ, LQ, SQ, RM, ROB	All	U, RM4, RM7, RM8
RM5	RF, IQ, LQ, SQ		
RM6	RF, RM		
RM7	RF, RM, ROB		
RM8	RM, ROB		
RM9	RF, IQ, LQ, SQ, RM		

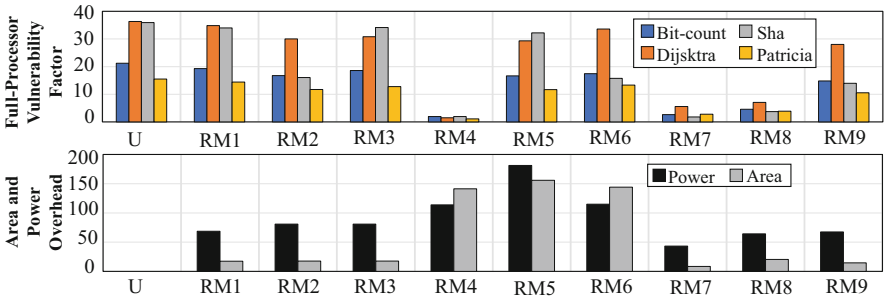

Fig. 12 Full-Processor Vulnerability Factor (FPVF) and power/area trade-off of the proposed heterogeneous reliability modes for different *MiBench* applications (adapted from [33])

of RM2, RM6, and RM9. The full-processor vulnerability of these three RMs has been reduced by more than 50% when executing *SHA* compared to *Dijkstra*.

2. Components such as the Rename Map and Reorder Buffer, when hardened, are highly effective in reducing the FPVF for all four applications. This is illustrated by the reliability modes RM4, RM7, and RM8, which have significantly lower FPVFs compared to their counter-parts. However, these two components occupy a significant percentage of the on-chip resources and hardening them leads to significant area and power overheads as illustrated by Fig. 12. This leads us to infer that hardening specific highly vulnerable pipeline components can significantly reduce the overall processor vulnerability for a wide range of application workloads based on their properties.

Furthermore, based on the data from these experiments, we perform an architectural space exploration that trades-off FPVF, area, and power overheads to extract the pareto-optimal reliability modes. The results of the experiments are illustrated in Fig. 13, where the *x*-, *y*-, and *z*-axes depict the FPVF, area, and power overheads, respectively. From these results, we make the following **key observations:**

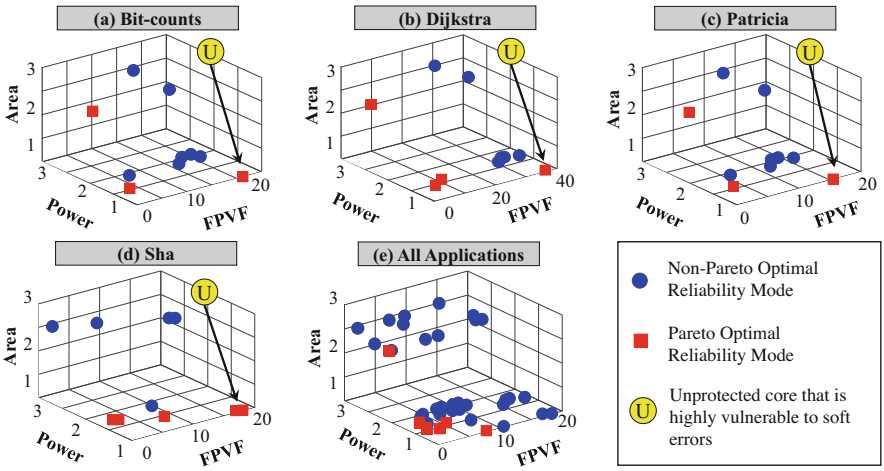

Fig. 13 Architectural space exploration of our heterogeneous reliability modes for *MiBench* applications (adapted from [33])

1. The design labeled *U*, i.e., the unprotected core, is pareto-optimal for all application workloads. This is expected as this reliability mode incurs zero area and power overheads and represents the least reliable processor design.
2. Although RM7 and RM8 significantly reduce the FPVF, due to their differences in power and area overheads, RM7 lies on the pareto-front for all individual application workloads, whereas RM8 is pareto-optimal only for *SHA* and *Dijkstra*. Similarly, RM4 is pareto-optimal for three of the four application workloads.
3. RM4, RM7, and RM8, all lie on the pareto-front when all applications are executed on the cores. This behavior is observed because of the varying levels of vulnerability savings achieved by the RMs when compared to their area and power overheads.
4. RM7 is pareto-optimal for four individual application workloads and reduces the FPVF by 87%, on average, while incurring area and power overheads of 10% and 43%, respectively.

3.3.3 State Compression Techniques

Reliability can also be improved at the software layer by inserting checkpoints in the application code. When an application encounters a checkpoint, the complete processor state, including all intermediate register and cache values, is stored in the main memory. These checkpoint states can be used to re-initialize the processor, which is referred to as rollback, in case a failure is detected and the next sequence of instructions are re-executed.

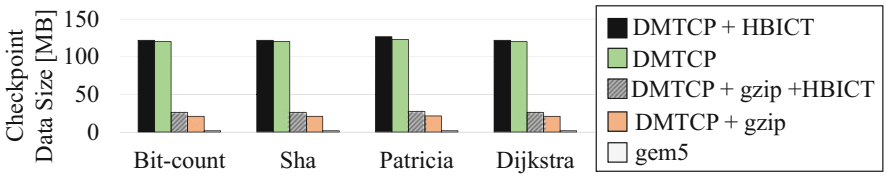

Fig. 14 Effectiveness of state compression techniques in reducing state size (adapted from [33])

The way checkpointing is implemented in *gem5* leads to significant loss in performance in case of frequent checkpoint restoration as the cache and pipeline states are not preserved, which, in turn, leads to a higher number of instructions being executed. *Distributed Multi-Threaded Checkpointing (DMTCP)* is a Linux compatible checkpointing tool that is used to checkpoint Linux processes. The back-end mechanism of DMTCP is accessible to programmers, via Application Programming Interfaces (APIs), to insert checkpoints into their application code. Inside *gem5*, these APIs can be used in combination with its pseudo-instructions to offer the functionality of creating/recovering checkpoint states for the applications being simulated inside *gem5*. Furthermore, the size of data generated by each checkpoint is typically large, especially in the case of O3 superscalar processors with large multi-level cache hierarchies. Therefore, we explore various compression strategies that can be used to efficiently compress and reduce the checkpoint data using techniques like the *Hash-Based Incremental Checkpointing Tool (HBICT)* and *GNU zip (gzip)*. HBICT provides DMTCP support to enable checkpoint compression using an approach called *delta compression*. This kind of compression mechanism preserves only changed fragments of a program's state, thereby considerably reducing the size of checkpoint data. gzip is a file compression technique based on the *DEFLATE algorithm*, which is a combination of lossless data compression techniques such as *LZ77* and *Huffman coding*. gzip can drastically reduce the size of checkpoint data, as illustrated by the results presented in Fig. 14. These techniques and compression algorithms are implemented in *gem5*, in different combinations, to reduce the size of checkpoint data for the four aforementioned *MiBench* applications, by executing them on an unprotected *ALPHA* processor. The effectiveness of different combinations of compression algorithms is illustrated in terms of checkpoint data size in Fig. 14. It can be observed that the combination of DMTCP and gzip is highly successful in reducing the checkpoint size by $\sim 6\times$. On the other hand, a combination of DMTCP, HBICT, and gzip techniques reduces the checkpoint size by $\sim 5.7\times$.

4 Run-Time Systems for Heterogeneous Fault-Tolerance

The techniques discussed in Sects. 2 and 3 also require a run-time manager for incorporating the application vulnerabilities with respect to several reliability threats

such as soft errors, aging, and process variation, as well as considering constraints like dark silicon and required performance (or tolerable performance overhead). Most of the adaptive hardware techniques exploit the application vulnerability to map applications on appropriate cores to reduce their vulnerability. Similarly, this concept can be applied to modify the applications with respect to the available hardened core or caches, which can also be combined with other hardware techniques to further reduce the vulnerabilities of the heterogeneous multi/many-core processors. Therefore, several techniques have been proposed to modify the execution patterns of the application or partitioning the application to develop a run-time system for reliability-heterogeneous multi/many-core processors.

1. **Aging- and Process Variation-Aware Redundant Multithreading [18, 36]:** *dTune* leverages multiple reliable versions of an application and redundant multithreading (RMT) simultaneously for achieving high soft error resilience under aging and process variability [36]. Based on the reliability requirements of the executing applications, *dTune* performs efficient core allocation for RMT while considering the aging state of the processor as well as process variation. It achieves up to 63% improvement in the reliability of a given application. Similarly, another approach [18] utilizes different software versions and RMT to improve the reliability of a system while considering the effects of soft errors and aging on the processor cores, to achieve an improved aging balancing.
2. **Variability-aware reliability-heterogeneous processor [21]:** This work leverages techniques at the hardware and run-time system layers to mitigate the reliability threats. In particular, this work focuses on TMR-based solutions to (partially) harden the cores for developing a many/multi-core reliability-heterogeneous processor. It uses a run-time controller to handle multiple cores with different reliability modes while considering the reliability requirements of the applications. In addition, it also exploits the dark silicon property in multi/many-core processors to offer a wide range of different performance-reliability trade-offs by over-provisioning the processor with reliability-heterogeneous cores.
3. **Aging-aware reliability-heterogeneous processor [10]:** This technique exploits the dark silicon property of the multi/many-core processors to design a run-time approach for balancing the application load to mitigate the reliability threats, i.e., temperature-dependent aging while also considering variability and current age of the cores in order to improve the overall system performance for a given lifetime constraint. The analysis shows that this run-time solution can improve the overall aging of the multi/many-core processor by 6 months to 5 years depending upon the provided design constraints and power overheads. Furthermore, this work also developed a fast aging evaluation methodology based on multi-granularity simulation epochs, as well as lightweight run-time techniques for temperature and aging estimation that can be used for an early estimation of temperature-dependent aging of multi/many-core processors.

There are other techniques which can exploit the functional and timing reliability in real-time systems to improve the application by generating the reliable application

versions or respective thread with different performance and reliability properties [38]. These reliable applications or respective thread can jointly be used with hardware techniques to improve the overall reliability of the multi/many-core heterogeneous processor. Another solution is to exploit the dynamic voltage and frequency scaling to generate the dynamic redundancy and voltage scaling with respect to the effects of process variations, application vulnerability, performance overhead, and design constraints [40]. This technique demonstrates up to 60% power reductions while improving the reliability significantly. Similarly, in addition to redundancy, multiple voltage-frequency levels are introduced while considering the effects of dark silicon in multi/many-core heterogeneous processor [39]. This technique also considers the effects of soft errors and process variations in their reliability management system that provides up to 19% improved reliability under different design constraints [35]. Most of the abovementioned approaches are focused on general purpose microprocessors; however, in application-specific instruction set processors (ASIPs), the hardware hardening and corresponding runtime software assisted recovery techniques can be used to improve the soft error vulnerabilities in ASIP-based multi/many-core systems. For example, dynamic core adaptation and application specificity can be exploited to generate a processor configuration which performs the error (caused by soft error) recovery for a particular application under the given area, power, and performance constraints [24–26]. Moreover, the baseline instruction set of the targeted ASIPs can also be modified or extended to enable the error recovery functionality [23].

5 Conclusion

This chapter discusses the building blocks of computing systems (both embedded and superscalar processors) with different heterogeneous fault-tolerant modes for the memory components like caches as well as for the in-order and out-of-order processor designs. We provide a comprehensive vulnerability analysis of different components, i.e., embedded and superscalar, processors and caches, considering the soft errors and aging issues. We also discuss the methodologies to improve the performance and power of such systems by exploiting these vulnerabilities. In addition, we briefly present that a reliability-aware compiler can be leveraged to comprehend software-level heterogeneous fault-tolerance by generating different reliable versions of the application with respective reliability and performance properties. Further details on reliability-driven compilation can be found in Chap. 5. Towards the end, we also analyze fault-tolerance techniques for application-specific instruction set processors (ASIPs).

Acknowledgments This work was supported in parts by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500—spp1500.itec.kit.edu). We would like to thank Arun Subramaniyan, Duo Sun and Segnon Jean Bruno Ahandagbe for their contributions to parts of the works cited in this chapter.

References

1. Agarwal, S., Garg, R., Gupta, M.S., Moreira, J.E.: Adaptive incremental checkpointing for massively parallel systems. In: Proceedings of the 18th Annual International Conference on Supercomputing, pp. 277–286. ACM, New York (2004)
2. Agarwal, M., Paul, B.C., Zhang, M., Mitra, S.: Circuit failure prediction and its application to transistor aging. In: 25th IEEE VLSI Test Symposium (VTS'07), pp. 277–286. IEEE, Piscataway (2007)
3. Ansel, J., Arya, K., Cooperman, G.: DMTCP: transparent checkpointing for cluster computations and the desktop. In: 2009 IEEE International Symposium on Parallel and Distributed Processing, pp. 1–12. IEEE, Piscataway (2009)
4. Baumann, R.C.: Radiation-induced soft errors in advanced semiconductor technologies. IEEE Trans. Device Mater. Reliab. **5**(3), 305–316 (2005)
5. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., et al.: The gem5 simulator. ACM SIGARCH Comput. Archit. News **39**(2), 1–7 (2011)
6. Chen, C.L., Hsiao, M.: Error-correcting codes for semiconductor memory applications: a state-of-the-art review. IBM J. Res. Dev. **28**(2), 124–134 (1984)
7. Esmaeilzadeh, H., Blem, E., Amant, R.S., Sankaralingam, K., Burger, D.: Dark silicon and the end of multicore scaling. In: 2011 38th Annual International Symposium on Computer Architecture (ISCA), pp. 365–376. IEEE, Piscataway (2011)
8. Gailly, J.: Gzip—the data compression program (1993)
9. Geist, A.: Supercomputing's monster in the closet. IEEE Spectr. **53**(3), 30–35 (2016)
10. Gnad, D., Shafique, M., Kriebel, F., Rehman, S., Sun, D., Henkel, J.: Hayat: harnessing dark silicon and variability for aging deceleration and balancing. In: Proceedings of the 52nd Annual Design Automation Conference, San Francisco, June 7–11, pp. 180:1–180:6 (2015)
11. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N.D., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., Swanson, S., Sylvester, D.: Underdesigned and opportunistic computing in presence of hardware variability. IEEE Trans. CAD Integr. Circuits Syst. **32**(1), 8–23 (2013). https://doi.org/10.1109/TCAD.2012.2223467
12. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: 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), pp. 3–14. IEEE, Piscataway (2001)
13. Henkel, J., Bauer, L., Becker, J., Bringmann, O., Brinkschulte, U., Chakraborty, S., Engel, M., Ernst, R., Härtig, H., Hedrich, L., Herkersdorf, A., Kapitza, R., Lohmann, D., Marwedel, P., Platzner, M., Rosenstiel, W., Schlichtmann, U., Spinczyk, O., Tahoori, M.B., Teich, J., Wehn, N., Wunderlich, H.: Design and architectures for dependable embedded systems. In: Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2011, part of ESWeek '11 Seventh Embedded Systems Week, Taipei, 9–14 October, 2011, pp. 69–78 (2011). https://doi.org/10.1145/2039370.2039384
14. Henkel, J., Bauer, L., Dutt, N., Gupta, P., Nassif, S., Shafique, M., Tahoori, M., Wehn, N.: Reliable on-chip systems in the nano-era: lessons learnt and future trends. In: Proceedings of the 50th Annual Design Automation Conference, p. 99. ACM, New York (2013)
15. Jeyapaul, R., Shrivastava, A.: Enabling energy efficient reliability in embedded systems through smart cache cleaning. ACM Trans. Des. Autom. Electron. Syst. **18**(4), 53 (2013)
16. Kang, K., Gangwal, S., Park, S.P., Roy, K.: NBTI induced performance degradation in logic and memory circuits: How effectively can we approach a reliability solution? In: Proceedings of the 2008 Asia and South Pacific Design Automation Conference, pp. 726–731. IEEE Computer Society Press, Silver Spring (2008)
17. Kessler, R.E.: The alpha 21264 microprocessor. IEEE Micro **19**(2), 24–36 (1999)
18. Kriebel, F., Rehman, S., Shafique, M., Henkel, J.: ageOpt-RMT: compiler-driven variation-aware aging optimization for redundant multithreading. In: Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, June 5–9, 2016, pp. 46:1–46:6 (2016). https://doi.org/10.1145/2897937.2897980

19. Kriebel, F., Rehman, S., Subramaniyan, A., Ahandagbe, S.J.B., Shafique, M., Henkel, J.: Reliability-aware adaptations for shared last-level caches in multi-cores. ACM Trans. Embed. Comput. Syst. **15**(4), 67:1–67:26 (2016). https://doi.org/10.1145/2961059
20. Kriebel, F., Rehman, S., Sun, D., Shafique, M., Henkel, J.: ASER: Adaptive soft error resilience for reliability-heterogeneous processors in the dark silicon era. In: Proceedings of the 51st Annual Design Automation Conference, pp. 1–6. ACM, New York (2014)
21. Kriebel, F., Shafique, M., Rehman, S., Henkel, J., Garg, S.: Variability and reliability awareness in the age of dark silicon. IEEE Des. Test **33**(2), 59–67 (2016)
22. Kriebel, F., Subramaniyan, A., Rehman, S., Ahandagbe, S.J.B., Shafique, M., Henkel, J.: R^2cache: reliability-aware reconfigurable last-level cache architecture for multi-cores. In: 2015 International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2015, Amsterdam, October 4–9, 2015, pp. 1–10 (2015)
23. Li, T., Shafique, M., Ambrose, J.A., Henkel, J., Parameswaran, S.: Fine-grained checkpoint recovery for application-specific instruction-set processors. IEEE Trans. Comput. **66**(4), 647–660 (2017)
24. Li, T., Shafique, M., Ambrose, J.A., Rehman, S., Henkel, J., Parameswaran, S.: RASTER: runtime adaptive spatial/temporal error resiliency for embedded processors. In: 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–7. IEEE, Piscataway (2013)
25. Li, T., Shafique, M., Rehman, S., Ambrose, J.A., Henkel, J., Parameswaran, S.: DHASER: dynamic heterogeneous adaptation for soft-error resiliency in ASIP-based multi-core systems. In: 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 646–653. IEEE, Piscataway (2013)
26. Li, T., Shafique, M., Rehman, S., Radhakrishnan, S., Ragel, R., Ambrose, J.A., Henkel, J., Parameswaran, S.: CSER: HW/SW configurable soft-error resiliency for application specific instruction-set processors. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 707–712. EDA Consortium, San Jose (2013)
27. McPherson, J.W.: Reliability challenges for 45 nm and beyond. In: 2006 43rd ACM/IEEE Design Automation Conference, pp. 176–181. IEEE, Piscataway (2006)
28. Mitra, S., Seifert, N., Zhang, M., Shi, Q., Kim, K.S.: Robust system design with built-in soft-error resilience. Computer **38**(2), 43–52 (2005)
29. Mukherjee, S.S., Emer, J., Reinhardt, S.K.: The soft error problem: an architectural perspective. In: 11th International Symposium on High-Performance Computer Architecture, pp. 243–247. IEEE, Piscataway (2005)
30. Mukherjee, S.S., Weaver, C., Emer, J., Reinhardt, S.K., Austin, T.: A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36, pp. 29–40. IEEE, Piscataway (2003)
31. Pagani, S., Khdr, H., Munawar, W., Chen, J.J., Shafique, M., Li, M., Henkel, J.: TSP: thermal safe power: efficient power budgeting for many-core systems in dark silicon. In: Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis, p. 10. ACM, New York (2014)
32. Portolan, M., Leveugle, R.: A highly flexible hardened RTL processor core based on LEON2. IEEE Trans. Nucl. Sci. **53**(4), 2069–2075 (2006)
33. Prabakaran, B.S., Dave, M., Kriebel, F., Rehman, S., Shafique, M.: Architectural-space exploration of heterogeneous reliability and checkpointing modes for out-of-order superscalar processors. IEEE Access **7**, 145324–145339 (2019)
34. Rehman, S., Kriebel, F., Prabakaran, B.S., Khalid, F., Shafique, M.: Hardware and software techniques for heterogeneous fault-tolerance. In: 24th IEEE International Symposium on On-Line Testing And Robust System Design, IOLTS 2018, Platja D'Aro, July 2–4, 2018, pp. 115–118 (2018). https://doi.org/10.1109/IOLTS.2018.8474219
35. Rehman, S., Kriebel, F., Shafique, M., Henkel, J.: Reliability-driven software transformations for unreliable hardware. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **33**(11), 1597–1610 (2014)

36. Rehman, S., Kriebel, F., Sun, D., Shafique, M., Henkel, J.: dTune: leveraging reliable code generation for adaptive dependability tuning under process variation and aging-induced effects. In: Proceedings of the 51st Annual Design Automation Conference, pp. 1–6. ACM, New York (2014)
37. Rehman, S., Shafique, M., Henkel, J.: Reliable Software for Unreliable Hardware: A Cross Layer Perspective. Springer, Berlin (2016)
38. Rehman, S., Toma, A., Kriebel, F., Shafique, M., Chen, J.J., Henkel, J.: Reliable code generation and execution on unreliable hardware under joint functional and timing reliability considerations. In: 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 273–282. IEEE, Piscataway (2013)
39. Salehi, M., Shafique, M., Kriebel, F., Rehman, S., Tavana, M.K., Ejlali, A., Henkel, J.: dsReliM: power-constrained reliability management in Dark-Silicon many-core chips under process variations. In: 2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 75–82. IEEE, Piscataway (2015)
40. Salehi, M., Tavana, M.K., Rehman, S., Kriebel, F., Shafique, M., Ejlali, A., Henkel, J.: DRVS: power-efficient reliability management through dynamic redundancy and voltage scaling under variations. In: 2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pp. 225–230. IEEE, Piscataway (2015)
41. Shafique, M., Garg, S., Henkel, J., Marculescu, D.: The EDA challenges in the dark silicon era: temperature, reliability, and variability perspectives. In: Proceedings of the 51st Annual Design Automation Conference, pp. 1–6. ACM, New York (2014)
42. Shafique, M., Rehman, S., Aceituno, P.V., Henkel, J.: Exploiting program-level masking and error propagation for constrained reliability optimization. In: Proceedings of the 50th Annual Design Automation Conference, p. 17. ACM, New York (2013)
43. Shivakumar, P., Kistler, M., Keckler, S.W., Burger, D., Alvisi, L.: Modeling the effect of technology trends on the soft error rate of combinational logic. In: Proceedings International Conference on Dependable Systems and Networks, pp. 389–398. IEEE, Piscataway (2002)
44. Srinivasan, J., Adve, S.V., Bose, P., Rivers, J.A.: The case for lifetime reliability-aware microprocessors. In: ACM SIGARCH Computer Architecture News, vol. 32, p. 276. IEEE Computer Society, Silver Spring (2004)
45. Subramaniyan, A., Rehman, S., Shafique, M., Kumar, A., Henkel, J.: Soft error-aware architectural exploration for designing reliability adaptive cache hierarchies in multi-cores. In: Design, Automation and Test in Europe Conference and Exhibition, DATE 2017, Lausanne, March 27–31, 2017, pp. 37–42 (2017)
46. Vadlamani, R., Zhao, J., Burleson, W., Tessier, R.: Multicore soft error rate stabilization using adaptive dual modular redundancy. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 27–32. European Design and Automation Association (2010)
47. Wang, W., Mishra, P.: Dynamic reconfiguration of two-level cache hierarchy in real-time embedded systems. J. Low Power Electron. **7**(1), 17–28 (2011)