IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 4, 2019, accepted September 23, 2019, date of publication October 4, 2019, date of current version October 17, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2945622

Architectural-Space Exploration of Heterogeneous Reliability and Checkpointing Modes for Out-of-Order Superscalar Processors

BHARATH SRINIVAS PRABAKARAN[1], (Student Member, IEEE), MIHIKA DAVE[2], FLORIAN KRIEBEL[1], SEMEEN REHMAN[3], AND MUHAMMAD SHAFIQUE[1], (Senior Member, IEEE)

[1]Institute of Computer Engineering, Technische Universität Wien (TU Wien), 1040 Vienna, Austria
[2]Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL 61820, USA
[3]Institute of Computer Technology, Technische Universität Wien (TU Wien), 1040 Vienna, Austria

Corresponding author: Bharath Srinivas Prabakaran (bharath.prabakaran@tuwien.ac.at)

This work is supported in part by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500 - http://spp1500.itec.kit.edu), and in part by the TU Wien Bibliothek for financial support through its Open Access Funding Program.

ABSTRACT State-of-the-art reliability techniques and mechanisms deploy full-scale redundancy, like double or triple modular redundancy (DMR, TMR), on different layers of the computing stack to detect and/or correct such transient faults. However, the techniques relying on full-scale redundancy incur significant area, performance, and/or power overheads, which might not always be feasible/practical due to system constraints such as deadlines and available power budget for the full chip (or a processor core). In this work, we propose a novel design methodology to generate and explore the architectural-space of heterogeneous reliability modes for out-of-order superscalar multi-core processors. These heterogeneous modes enable varying reliability and power/area trade-offs, from which an optimal configuration can be chosen at run time to meet the reliability requirements of a given system while reducing the corresponding power overheads (or solving the inverse problem, i.e., maximizing the reliability under a given power constraint). Our experimental results show that a pareto-optimal heterogeneous reliability mode reduces the core vulnerability by 87%, on average, across multiple application workloads, with area and power overheads of 10% and 43%, respectively. To further enhance the design space of heterogeneous reliability modes, we investigate the effectiveness of combining different processor state compression techniques like Distributed Multi-threaded Checkpointing (DMTCP), Hash-based Incremental Checkpointing (HBICT) and GNU zip, such that the correct processor state can be recovered once a fault is detected. We reduced the checkpoint sizes by a factor of $\sim 6\times$ using a unique combination of different state compression techniques.

INDEX TERMS Reliability, multi-cores, heterogeneity, fault-tolerance, AVF, hardening, microprocessors, superscalar, resilience, design space exploration, checkpointing, out-of-order, architecture.

I. INTRODUCTION

Aggressive transistor scaling has led to an increased susceptibility towards several reliability problems, such as soft errors, at the hardware layer [1]. Soft errors are *transient faults* in the hardware that cause bit-flips in the micro-architecture, which may propagate to the application output and corrupt its state, or may terminate the application's execution [2], [3]. The rate of occurrences of these soft errors is expected to increase with each new generation of microprocessor being released, due to aggressive shrinking of the transistor's feature sizes and imperfection in the fabrication process [4], [5] (see Section II).

Plenty of research works focusing on techniques like full-scale redundancy and checkpointing have been proposed towards prevention, detection, and/or mitigation of soft errors across the computing stack, i.e., the hardware and software layers [6], [7]. Reliability at the hardware layer is ensured through redundancy of execution paths and/or hardening of pipeline components, i.e., full-scale Double or Triple Modular Redundancy (DMR, TMR). Software-layer techniques realize full-scale spatial/temporal redundancy by executing

The associate editor coordinating the review of this manuscript and approving it for publication was Cristian Zambelli.

multiple redundant instructions or threads of an application, thereby ensuring a reliable output [8]–[10]. However, these full-scale redundancy techniques incur significant performance and energy overheads (e.g., in case of temporal redundancy), and area/power/energy overhead (e.g., in case of spatial redundancy).

Therefore, we propose to investigate the individual properties and requirements of an application workload to determine the component-level vulnerabilities of an out-of-order superscalar processor at design-time, i.e., enabling reliability provisions at a much finer granularity. Based on this analysis, we develop a *wide range of heterogeneous reliability and checkpointing modes* that enable efficient control over the achieved reliability and the incurred overhead, especially when considering the diverse resilience properties of different executing applications at different run time instances. Our previous work [11] provides an initial proof-of-concept of this work and preliminary results for the feasibility of reliability-heterogeneous cores. In this work, we significantly extend this concept, and provide a systematic methodology to integrate such reliability heterogeneous modes on a chip along with other different types of reliability mechanisms like checkpointing and state compression to expand the space of design trade-offs for reliability vs. overhead.

In a nutshell, **we make the following novel contributions:**

(1) *Component-Level Vulnerability Analysis*: We leverage the Architectural Vulnerability Factor (AVF) metric to perform a comprehensive vulnerability analysis for different pipeline components of a single-core and quad-core out-of-order superscalar processor when executing diverse application workloads.

(2) *A Methodology for Architectural-Space Generation and Exploration*: We propose a novel methodology that:
   (a) analyzes the architectural vulnerability of an out-of-order superscalar microprocessor;
   (b) generates a wide range of heterogeneous reliability modes, such that each mode deploys distinct reliability measures in different pipeline components;
   (c) enables reliability-power trade-offs that can be used to *optimize the applications' reliability requirements under the given power constraint*, or vice-versa, *minimize the power consumption under the given reliability constraints*.

(3) *A Run-Time System*: We evaluate the run-time benefits of our heterogeneous reliability modes by executing various application workload mixes on our heterogeneous multi-core processor. We propose an evaluate two task mapping heuristics, namely, Vulnerability-Constrained Power Minimization and Power-Constrained Vulnerability Minimization.

(4) *Efficient State Compression:* To further enhance the processor reliability and to increase the design space, we analyze and investigate combinations of state-of-the-art compression techniques to effectively reduce the storage requirements of checkpointing data.

(5) *Evaluation & Discussion*: We evaluate the effectiveness of our heterogeneous reliability modes under diverse application workloads using a modified version of the cycle-accurate simulator *gem5* to offer the required functionality.

Fig. 1 illustrates an overview of our contributions in a design-flow for developing heterogeneous multi-core processors.

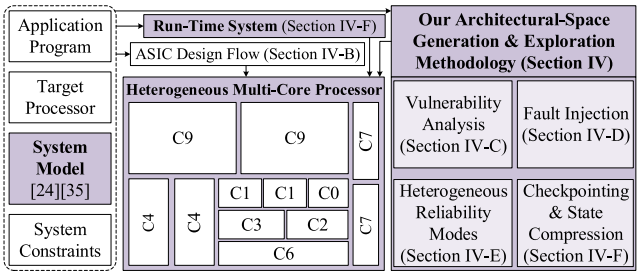

FIGURE 1. An overview of our contributions (highlighted boxes) in the processor design flow.

Paper Organization: Section II presents the preliminaries and background information required to understand our proposed contributions. We discuss the system models in Section III. Section IV presents our methodology for generation and exploration of the architectural-space of heterogeneous reliability modes, including results that illustrate the benefits of the proposed approaches. Section V presents the related work on state-of-the-art reliability techniques and heterogeneous reliability approaches, followed by the conclusion presented in Section VI.

II. PRELIMINARIES AND BACKGROUND

A. SOFT ERRORS

In the era of nanometer technology nodes, reliability threats like manufacturing-induced *process variation*, *device aging*, and *transient faults* are increasingly challenging the functional correctness and safety-critical aspects of the systems where these electronic devices are deployed [1]. These electrical disturbances that disrupt the normal operation of a circuit are called Single Event Effects (SEEs). It can be caused by the passage of a single ion through a circuit node. These disturbances can be either destructive or non-destructive. An example of a non-destructive SEE is Single-Event Upsets (SEUs). These errors can be single-bit or multi-bit depending upon various factors like the particle's energy, transistors dimensions, and electrical properties, operating scenarios (e.g., altitude of the device under usage), etc. These SEUs are *transient faults* (e.g., **soft errors**), which have emerged as a serious threat to the reliability of a digital system. These soft errors are generated at the hardware layer, due to four key factors, namely,

(1) *Alpha Particles*, which are positively charged composite particles emitted during radioactive decay.

These particles travel through the semiconductor device thereby disturbing the electron distribution of the transistor [12].

(2) *Cosmic Rays*, which are a flux of energetic neutrons that are constantly emitted by the solar system [13], [14].

(3) *Thermal Neutrons*, which are neutrons that have attained thermal equilibrium after dissipating all kinetic energy [15].

(4) *Internal Factors* such as random noise, signal integrity issues, cross-talks, and electromagnetic interference [3].

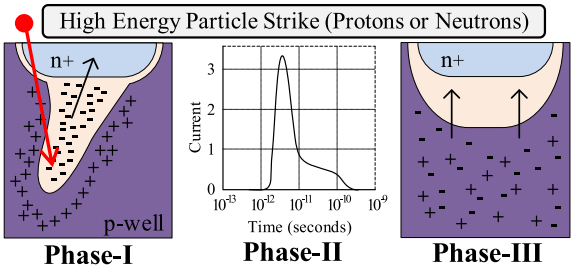

FIGURE 2. The three phases of the soft error Phenomenon (adapted from [2]).

Soft errors cause temporary bit-flips either in the control or data path of a micro-architecture, or in the on-chip memory cells, which may propagate to the application output (incorrect output), or may crash (incorrect instruction execution), hang (application enters an unresponsive state), or terminate the application execution [2]. Fig. 2 illustrates the soft-error phenomenon, which can be broken into three phases.

(1) First, in the *ion-track formation* phase (phase-I), a high energy particle (such as the cosmic rays discussed earlier) strikes the transistor to generate multiple electron-hole pairs, which in turn increase the concentration of carriers along the ion's path.

(2) In phase-II (*current pulse generation*), the ions collected at the depletion region form a "temporary" channel that funnels the current from source to drain, which could toggle the transistor state for tens of picoseconds. This can result in a bit flip in (i) the memory cell, which can be latched to the incorrect value until and unless it is overwritten by another value; or (ii) the logic gate that can potentially propagate to the final output of the circuit, thereby corrupting the output of the circuit.

(3) In the *ion diffusion* phase (phase-III), for tens or hundreds of picoseconds, the charges diffuse into the depletion layer, thereby disintegrating the temporary channel.

B. INCREASING SOFT ERROR RATES

In the earlier generation technology nodes, the transistor dimensions were large enough that a temporary channel could not funnel the current from source to drain. Furthermore, due to reducing transistor dimensions, the rate of soft error occurrences is increasing with each new generation of processors being released into the market, due to their fabrication using continuously smaller technology nodes [4], [5] (see Fig. 3). This is a major threat to the current world infrastructure, which heavily relies on electronics for all activities, such as work, communication, transportation, socializing, internet, etc. Even the day-to-day devices and services that people use, e.g., wearable devices such as smart-watches and fitness trackers, mobile computing platforms such as mobile phones and laptops, and on-demand cloud services offered by large-scale data centers, heavily rely on the reliability of electronic devices. This becomes even more crucial for safety-critical application domains like aerospace, automotive, healthcare, industry 4.0, smart grids, smart homes, etc.

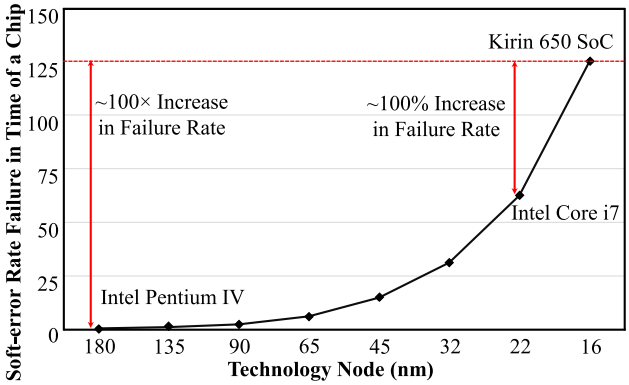

FIGURE 3. Increase in soft-error rate of a chip for Multiple Technology Nodes (adapted from [5]).

C. PROCESSOR HARDENING

Reliability at the hardware layer is typically ensured by the use of full-scale redundancy, which involves instantiating multiple instances of the hardware unit with the same set of inputs, to generate outputs that can be compared with each other to detect (in DMR) or correct errors using a voter circuit (in case of TMR), which we refer to as *hardware hardening*. Besides these hardware redundancy measures, techniques like software-level redundancy, application checkpointing and rollback, shadow latches, etc. (see Section V for an overview of the related work) can be also used to detect and mitigate soft errors.

An overview of these hardware-level redundancy techniques is presented in Fig. 4. DMR and TMR incur significant area and power overheads caused by the redundant hardware units and the additional circuitry used to detect or correct errors. Furthermore, since the additional hardware components execute in parallel, the throughput of the system is not affected, with a minimal gate-level increase in delay caused by the voter circuit. Typically, to ensure very high reliability, the entire processor pipeline (full-scale) is hardened, i.e., all the pipeline components are instantiated thrice with the same

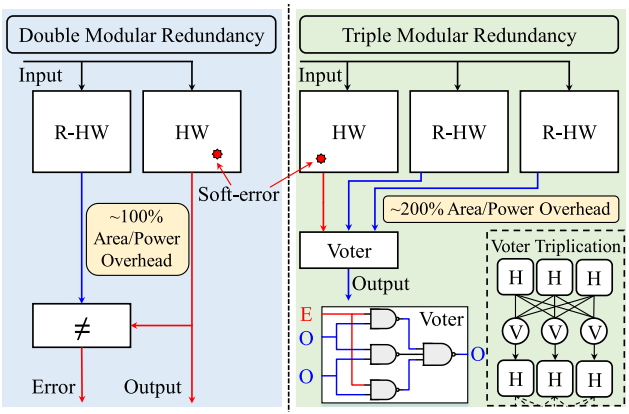

FIGURE 4. An overview of the redundancy techniques at the hardware layer.

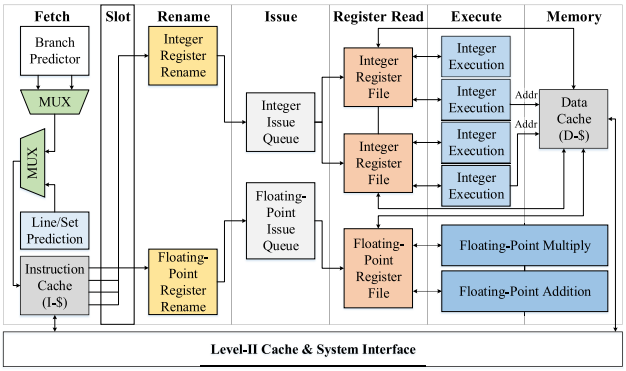

FIGURE 5. ALPHA 21264 out-of-order superscalar processor architecture (adapted from [19]).

set of inputs and a voter circuit to elect the majority output, as illustrated by Gaisler's completely hardened LEON3-FT microprocessor that deploys redundancy in the register file and cache memory [16]. Fig. 4 also illustrates the gate-level implementation of the voter circuit, and how, in the case of soft errors, the majority output is elected and generated as the final output. Note, this leads to the possibility of the voter circuit becoming a single point of failure, which is mitigated by triplicating the voter circuit as well, and has been deployed, for example, in the Saturn Launch Vehicle Digital Computer [17], [18]. In this work, without the loss of generality, we advocate the enabling of fine-grained reliability at different component level that can facilitate the instantiation of different hardening modes for different processor cores, thereby providing a wide range of reliability-power trade-offs. As a proof of concept, we will showcase an example of using component-level TMR with a single majority voter circuit. However, any other reliability mechanism can be deployed as a knob at the component level.

D. OUT-OF-ORDER SUPERSCALAR PROCESSORS

Besides transistor scaling, architectural innovations such as deep pipelining, instruction-level parallelism, out-of-order execution, speculative execution, branch prediction, etc. have tremendously increased the computing capabilities of microprocessors. Almost all the current generation microprocessors are designed with such functionalities to ensure high system performance. For example, *superscalar processors* exploit an application's instruction-level parallelism to execute multiple instructions in parallel during the same clock-cycle on multiple different execution units [20]. *Out-of-Order processors* execute instructions out-of-order, as opposed to the typical sequential execution, by exploiting the interdependency, or the lack thereof, of program instructions and the data processed by them [21]. This allows for executing "independent" instructions in clock-cycles that would be otherwise lost in pipeline stalls caused by control- or data-flow dependencies. Fig. 5 illustrates the control- and data-path of the ALPHA 21264 out-of-order superscalar microprocessor [19], which is widely used in the architecture research community.

Alpha 21264, or Alpha 7, is a four-issue, seven pipeline stage superscalar processor architecture that is capable of executing up to six (four integer and two floating-point) instructions per cycle (IPCs) while sustaining four instructions simultaneously. During a program's execution, the processor can accommodate up to 80 instructions in the pipeline, which is kept track of using the processor's re-order buffer (ROB). The Alpha 7 processor also includes two cache levels, i.e., the primary and secondary caches. The processor uses a modified Harvard architecture that implements separate primary instruction (I-cache) and data caches (D-cache), typically of size 64KB each. The D-cache is dual-ported to allow simultaneous read and write on both rising and falling edge of the clock. This feature allows for reducing the area and power overheads associated with duplicating the cache, as in the Alpha 21164 microprocessor. The secondary cache, or B-cache, is usually a direct-mapped cache that is located off-chip and shared by all processor cores. Typically, L2-cache has a maximum capacity of 16MB and is constructed using synchronous static random access memory (SSRAM), which is accessed using a dedicated 128-bit high-bandwidth bus [22]. Branch prediction in this microprocessor is implemented using a hybrid two-level branch prediction algorithm called tournament prediction, with a minimum branch misprediction penalty of 7 clock-cycles [22]. The processor was built using 15.2 million transistors, roughly 40% of which was occupied by the core processing unit and the rest of which was consumed by the caches and branch history tables [23].

III. SYSTEM MODEL

A. ARCHITECTURE MODEL

To cater for different application workloads with varying reliability requirements, we envision a reliability-heterogeneous multi-core processor (*HMC*):

$$HMC = \{PC_1, PC_2, \ldots, PC_M\}$$

where PC_j denotes the j^{th} processor core, such that, $j \in \{1, 2, \ldots, M\}$, with a total of M processor cores in the HMC. Each processor core has L different architectural components, denoted as:

$$PC_j = \{C_{(j,1)}, C_{(j,2)}, \ldots, C_{(j,L)}\}$$

where $C_{(j,k)}$ denotes the k^{th} component in the j^{th} processor core. Each architectural component (like re-order buffer, register file, instruction queue, etc.) in each processor core ($C_{(j,k)}$) can be hardened by using mechanisms like TMR, DMR, Checkpointing, and Rollback, Error-Correcting Codes, or Razor latches. We denote the i^{th} reliability technique of the component $C_{(j,k)}$ as:

$$RT(C_{(j,k)}) = i$$

Without loss of generality, in this work, we explore the applicability of TMR for designing the heterogeneous reliability modes. This leads to $i = \{0, 1\}$, where $RT(C_{(j,k)}) = 0$ denotes the unprotected component without any type of hardening and $RT(C_{(j,k)}) = 1$ denotes a component that has been hardened by triple modular redundancy, thereby enabling heterogeneous hardening.

The area of each processor core is denoted as $A(PC_j)$, which is the summation of area of all the processor components, including the overhead of hardening certain components. Note, only a selective subset of the different heterogeneous reliability modes can be activated at run-time due to the total power constraint of a system while considering the application's reliability requirement. An overview of the symbols used in this work and their denotations have been presented in Table 1.

TABLE 1. Symbols and denotations.

Symbol	Denotation
HMC	Heterogeneous Multi-core Processor
M	Total Number of Cores in HMC
PC_j	j^{th} Processor Core
$C_{(j,k)}$	k^{th} Architectural Component in j^{th} Processor Core
L	Total Number of Architectural Components in PC_j
$RT(C_{(j,k)})$	The Reliability Technique used to Harden Component $C_{(j,k)}$
$A(PC_j)$	Area of the Processor Core PC_j
T	Set of Tasks
Z	Total Number of Tasks
E	Set of Task Dependencies
$P(T_q, PC_j)$	Power Consumption for the q^{th} Task when executing on j^{th} Processor Core
$L(T_q, PC_j)$	Average Execution Time for the q^{th} Task when executing on j^{th} Processor Core
$AVF_{C_{(j,k)}}$	Architectural Vulnerability Factor of Component $C_{(j,k)}$
$FPVF(T_q, PC_j)$	Full-Processor Vulnerability Factor of Processor Core PC_j for Task T_q
N	Number of Clock Cycles
$VulnerableBits$	Total Number of Vulnerable Bits
$VulnerableTime$	Time Duration of Vulnerable Bits
$TotalBits$	Total Number of Output Bits
$TotalTime$	Total Duration of Application Execution
P_{error}	Error Rate of a Transient Fault
P_{flip}	Probability of High-Energy Particle Strike leading to a Soft Error
N_{error}	Number of Program Failures
N_{FI}	Spatial Vulnerability of Component
ORM	Optimal Reliability Modes

B. APPLICATION MODEL

The applications are modeled as a set of task graphs $\{T, E\}$ containing task and dependency information for all application workloads. T is denoted as $T = \{T_1, T_2, \ldots, T_Z\}$ for a set of Z tasks. E is defined as $E = \{E_{xy} \mid (T_x, T_y) \in T\}$ for the set of task dependencies. For the given processor core (PC_j) each task T_q has the following execution properties:

- $P(T_q, PC_j)$, which denotes the peak power consumption,
- $L(T_q, PC_j)$, which denotes the average performance in terms of execution time, and
- $FPVF(T_q, PC_j)$, which denotes the full-processor vulnerability factor.

C. RELIABILITY MODEL

The Architectural Vulnerability Factor (AVF) of a hardware component is defined as the probability of a fault to propagate to the final output resulting in an execution error [24]. We compute the *AVF* of a component $C_{(j,k)}$ as the fraction of bits vulnerable in each cycle (*Vulnerable-Bits*) to the total number of output bits (*TotalBits*) generated by component $C_{(j,k)}$ for a duration of N cycles. AVF of a component $C_{(j,k)}$ is '0' if the component is hardened, or produces no architecturally incorrect bits [24]. Note, all bits of a branch predictor are always architecturally correct, therefore a branch predictor's AVF is always '0'. Similarly, all bits of the program counter (PC) are always vulnerable, therefore the AVF of a PC is always '100' [24]. AVF is estimated using the following equation:

$$AVF_{C_{(j,k)}} = \frac{\sum_{n=0}^{N} VulnerableBits(C_{(j,k)})}{TotalBits \times N} \times 100$$

To study the impact of component hardening on the full-processor, we extend the *AVF* to define the *Full-Processor Vulnerability Factor* (*FPVF*) for a given application workload. We define *FPVF* 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 pipeline (*TotalBits*) for the total duration of application execution (*TotalTime*). It is computed using the following equation:

$$FPVF(T_q, PC_j) = \frac{\sum_{\forall C_{(j,k)}} VulnerableBits(C_{(j,k)}) \times VulnerableTime(C_{(j,k)})}{\sum_{\forall C_{(j,k)}} TotalBits(C_{(j,k)}) \times TotalTime(C_{(j,k)})} \times 100$$

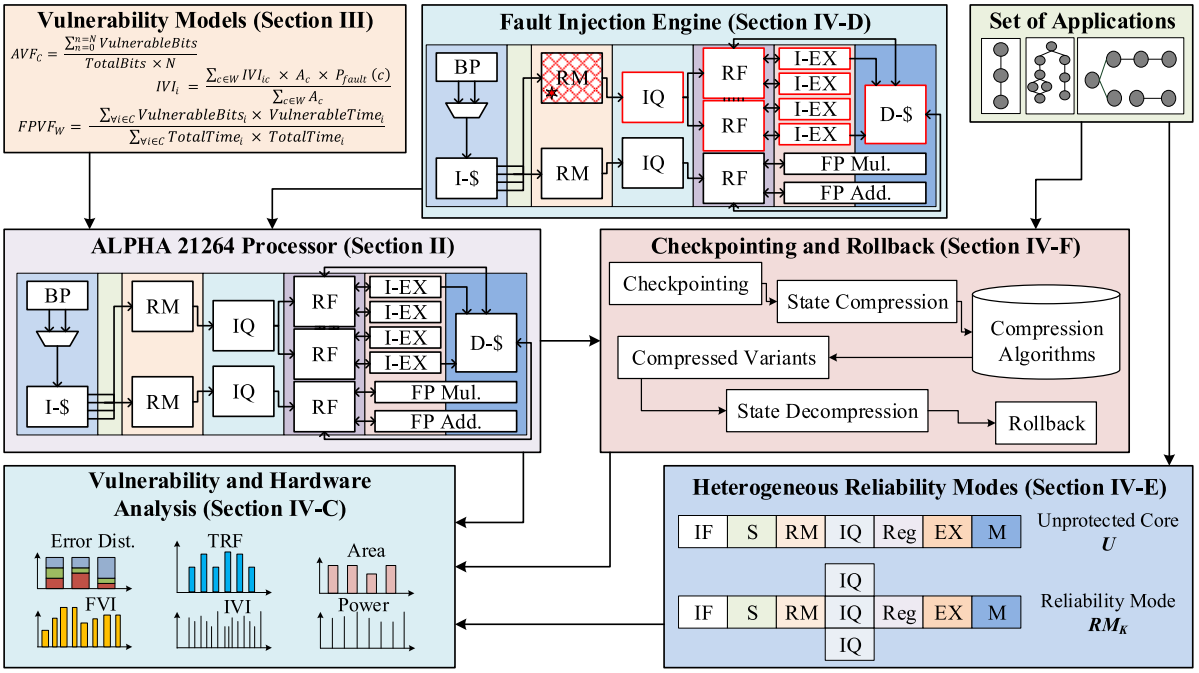

FIGURE 6. Overview of our architecture-space generation and exploration methodology for hardening out-of-order superscalar heterogeneous multi-core processors.

IV. HETEROGENEOUS RELIABILITY MODES OF OUT-OF-ORDER SUPERSCALAR CORES

A. METHODOLOGY OVERVIEW

Fig. 6 presents an overview of our methodology for designing and exploring heterogeneous reliability modes for out-of-order superscalar multi-core processors. Our methodology targets two approaches for designing heterogeneous reliability modes: (1) Redundancy, and (2) Checkpointing. To ensure reliable execution at the hardware layer, we propose hardening the processor's highly vulnerable pipeline components. These pipeline components are selected based on the initial fault-injection experiments, or on the AVF values that are estimated based on the number of vulnerable bits and vulnerable time of each component (see model description in Section III). Furthermore, we ensure reliability by investigating state compression techniques that can reduce the size of checkpoint data. Before moving on to our fault-injection and vulnerability analyses, we will present our experimental setup for better understanding.

B. EXPERIMENTAL SETUP

To evaluate the vulnerability, power and area requirements of the proposed heterogeneous reliability modes, we have modified the well-established open-source tools like the cycle-accurate system simulator, gem5 [25] and HP's power and area estimator tool McPAT [26]. Our extensions to these toolchains provide the following functionality: (1) estimate the vulnerability of all pipeline components by determining their AVFs [24], (2) support for heterogeneous reliability modes by hardening key pipeline components using component-level redundancy [11], but not full-scale pipeline triplication all the time, and (3) checkpoint processor states using mechanisms like Distributed Multi-Threaded Checkpointing (DMTCP) [27], [28] and Hash-Based Incremental Checkpointing Tool (HBICT) [29], [30]. Due to its high customization capability, we use the Alpha 21264 four-issue out-of-order superscalar core [19] as our target platform. We use the primitive Linux kernel 2.6 that is available with the default installation of gem5 as the operating system for our ALPHA 21264 micro-processor.

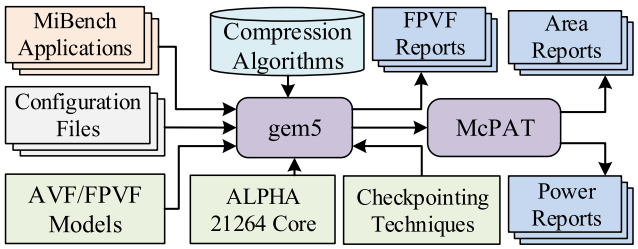

FIGURE 7. Overview of our experimental setup.

Furthermore, we extend the concept of AVF towards the FPVF metric (see Section III) to evaluate the impact of component hardening on the reliability mode, for a given application workload. To account for a wide range of applications, we evaluate the proposed heterogeneous reliability modes using the MiBench application benchmark suite. Fig. 7 presents an overview of our experimental setup.

C. VULNERABILITY ANALYSIS

We evaluate the vulnerability of an O3 superscalar Alpha 21264 core components [19] for the `Bit-counts`, `SHA`,

`Dijkstra`, and `Patricia` application workloads [31]. We analyze the vulnerability of the following key pipeline components:

- Re-order Buffer (ROB),
- Issue Queue (IQ),
- Load Queue (LQ),
- Store Queue (SQ),
- Integer, Floating Pt. Register Files (RF),
- Rename Map (RM),
- Integer ALU (Int. ALU),
- Floating Point ALU (FP ALU),
- Integer Multiply/Divide (Int. MD), and
- Floating Point Multiply/Divide (FP MD).

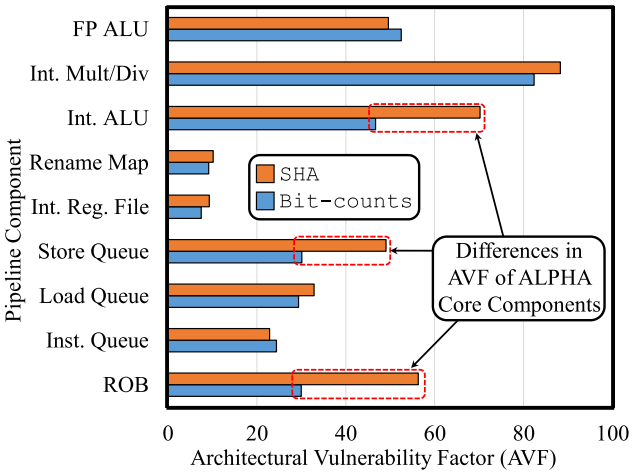

FIGURE 8. Differences in AVF of Alpha 7 Pipeline components under (`SHA` and `Bit-counts` Workloads).

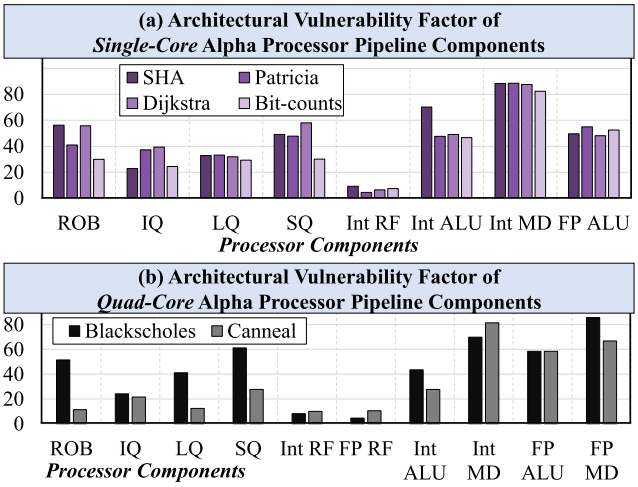

FIGURE 9. AVF distribution of Key Pipeline components in single- and Multi-Core Alpha 7 processors.

The results of our vulnerability analyses are presented in Figs. 8 and 9.

From the results obtained, we make the following ***key observations***:

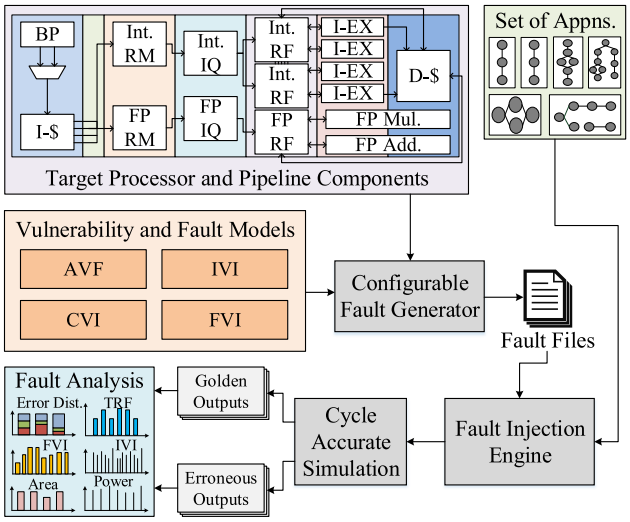

FIGURE 10. Overview of the Fault Injection Methodology for analyzing processor component Vulnerabilities.

- The AVFs of the different pipeline components vary for different application workloads.
- 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`.
- Similarly, the re-order buffer is 27% and 46% less vulnerable to soft errors during the execution of `Patricia` and `Bit-counts`, when compared to workloads like `SHA` and `Dijkstra`.
- Similar differences in component-AVFs can be observed when varying multi-threaded application workloads, from the PARSEC benchmark suite, are executed on a multi-core processor, as shown in Fig. 9.

These components have different AVFs because of the type of instructions being executed and their application-specific properties (compute or memory-intensive, instruction-level parallelism, cache hit/miss rate, etc.). For example, components like the Re-order Buffer and the Store Queue are more vulnerable in `SHA` because of higher levels of instruction-level parallelism and more store instructions.

Based on this analysis, we can infer that hardening certain components of the pipeline increase the reliability of a core more than hardening the other components. Therefore, we generate a wide range of reliability-heterogeneous Alpha cores, and explore this architectural-space in terms of reliability, power, and area, to select a configuration that increases the reliability of application executions while decreasing the area/power overhead.

D. FAULT INJECTION

Fault injection techniques are typically used to study, analyze and evaluate the behavior of a system susceptible to faults [32]–[34]. The fault model for the ALPHA core components is based on single- and multi-bit transient faults.

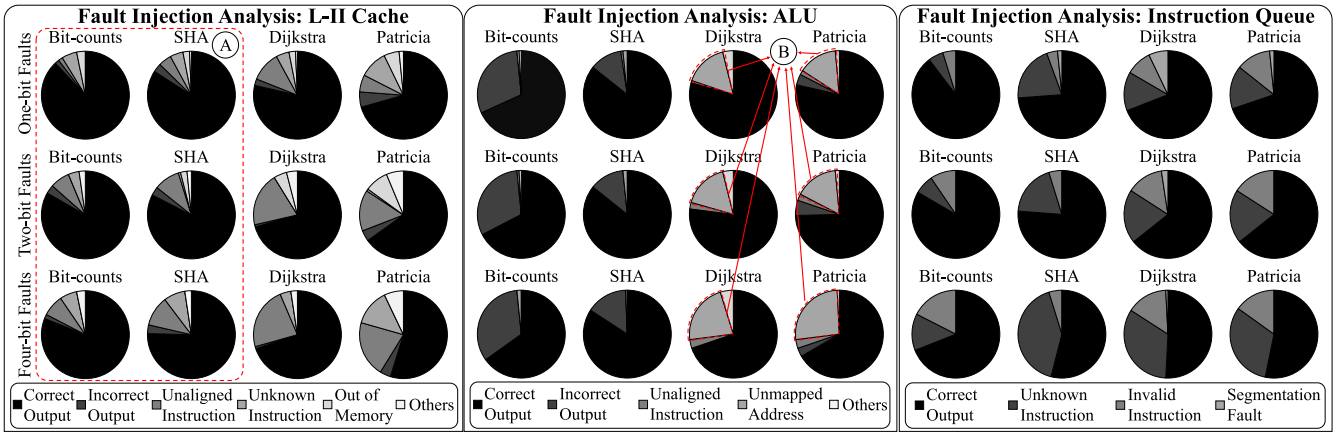

FIGURE 11. Error rate of three Pipeline components (L2 Cache, ALU, instruction Queue) in the Alpha 7 Processor.

The soft error rate for each component is defined as the product of error rate and the component's AVF. The soft error rate of the processor's pipeline components have been derived from the works presented in [35], [36]. To account for a component's spatial vulnerability (N_{FI}), the number of faults injected in a pipeline component is proportional to its on-chip area. We define P_{flip} as the probability that a high-energy particle strike leads to a change in the logic state of a pipeline component. Furthermore, to facilitate fast simulation, the faults are injected in the region of interest, the components, registers, and cache lines used by the application. The application output is classified into 3 major categories, namely, (1) correct output, (2) incorrect output, and (3) program failures (N_{error}), which comprise of multiple scenarios such as unaligned instruction, unmapped address, and segmentation fault. The error rate (P_{error}) of a transient fault in the component leading to an error in the application execution is defined as follows:

$$P_{error} = P_{flip} \times \frac{N_{error}}{N_{FI}} \tag{1}$$

An overview of the methodology used to inject and analyze faults in various pipeline components is presented in Fig. 10. Based on the vulnerability and fault models presented in Section III and the configuration of the target processor, including its pipeline components, we generate a list of fault files, that is provided as an input to the fault injection engine. This is used to insert faults/bit-flips into the target processor platform during the application's execution using a cycle-accurate simulator, i.e., gem5. Though 1-bit and 2-bit faults are common, we tried to evaluate our techniques under multiple fault cases to study the efficacy of the proposed contributions. For instance, 4 MBUs are indeed rare and may only occur when a very high energy particle strikes a nano-scale transistor at high altitude. However, besides our fault cases, we used this aggressive case in our fault injection experiments as well to identify the criticality of pipeline components in extreme cases, i.e., the components that are highly vulnerable to soft errors and to observe the error rates

TABLE 2. Processor parameters for vulnerability analyses experiments.

Parameter	Value
Core	Alpha 21264
Frequency	2 GHz
Simulation Mode	Syscall Simulation
L1 Cache (I-$ and D-$)	32kB, 2-way, 64B, 2 cycles
L2 Cache	256kB, 2-way, 64B, 20 cycles
Cache Policy	Snooping Coherence, LRU
TLB	Data: 64, Instruction: 48
Re-order Buffer	192 Entries
Instruction Queue	64 Entries
Load-Store Queues	32 Entries
Register File	Integer: 256, Floating-Pt.: 256

and types when injecting single- vs. multi-bit faults, and whether a similar fault trend is observed. The architectural parameters for the Alpha processor and the fault injection experiments are illustrated in tables 2 and 3. We study the output obtained from these simulations, which contains a list of correct and erroneous outputs. These outputs are then compared against the golden execution to estimate the type of error and the frequency of these error occurrences for various pipeline components. A subset of the results obtained from this experiment is illustrated in Fig. 11.

The results in Fig. 11 depict the error rate of three pipeline components, namely, Level-II Cache, Integer Arithmetic Logic Unit, and Instruction Queue. Faults injected in the L2-cache lead to four major types of error and correct output. The rest of the types are classified into the "others" category. The four major error categories are: (1) incorrect output, (2) unaligned instruction, (3) unknown instruction, and (4) out of memory. The label *A* depicts the applications with a higher percentage of correct output when compared to the others. On average, the `Bit-counts` and `SHA` applications produce a correct output more than 80% of the time, whereas `Dijkstra` and `Patricia`, on average produce a correct output less than 70% and 60% of the time. The changes in L2-cache vulnerability can be attributed to two factors, i.e., the amount of data being accessed and the

TABLE 3. Parameters for fault injection experiments.

Parameter	Description	Properties/ Values
Distribution	Distribution models for fault generation	Random
Bit Flips	Minimum/Maximum number of bits flipped	1/1, 1/2, ...
Fault Probability	Probability that strike becomes a fault [2]	10%-100%
Fault Location	List of target processor components	Register file, PC, IQ, etc.
Processor Layout/ Area	Size of the complete target device	Gate-equivalents or mm^2 [38]
Component Area	Area of different processor components given as percentage of processor area	0%-100%
Place and Altitude	City and altitude at which the device is used to determine the flux rate (N_{Flux})	Oslo, 1-20km
Frequency	Operating frequency of the processor	50, 100 MHz

number of load/store instructions in the application. The reduced vulnerability of the L2-cache, in the applications depicted by label *A*, can be attributed to the decreased amount of data being accessed from the L2-cache and the lower number of load/store instructions in the `Bit-counts` and `SHA` applications. This directly corresponds to a higher number of L1-cache hits, thereby reducing the criticality of the data present in the L2-cache and reducing its architectural vulnerability. Therefore, the probability of a soft error in L2-cache leading to an error during the execution is higher in an application with a relatively higher number of load and store instructions, and the amount of data accessed from L2, as compared to the others. Similarly, the label *B* depicts the percentage of fault injection experiments that lead to an unmapped address. As explained in the earlier example, due to the higher number of load and store instructions in `Dijkstra` and `Patricia`, the large number of unmapped addresses can be attributed to the corruption of bits during address generation. Similarly, due to their compute-intensive nature, a higher number of incorrect outputs are generated by faults injected in an ALU during the execution of applications like `bit-counts` and `SHA`. Faults injected in the Instruction Queue cause three major types of error, namely, (1) unknown instruction, (2) invalid instruction, and (3) segmentation fault.

E. HETEROGENEOUS RELIABILITY MODES FOR ALPHA CORES

As discussed in Section IV-C, the AVF of the pipeline components varies for the different application workloads. Hence, we propose to harden a combination of the key pipeline components in out-of-order superscalar processors, instead of employing full-scale TMR across the complete pipeline, to increase core reliability while reducing the area and power overheads of full-scale TMR. This generates a design space of multiple heterogeneous reliability modes (RM), nine of which are illustrated in this work (and unprotected core). Table 4 presents our list of nine proposed heterogeneous RM and the components that are hardened in these modes using TMR. Hardened components have three instances with the same inputs, and a voter circuit at the output to determine the majority. An overview of the proposed heterogeneous reliability modes for Alpha 7 processor is presented in Fig. 12.

TABLE 4. Proposed heterogeneous reliability modes.

Reliability Mode	Components Hardened
U	Unprotected
RM1	RF
RM2	IQ, RM
RM3	IQ, LQ, SQ
RM4	IQ, LQ, SQ, RM, ROB
RM5	RF, IQ, LQ, SQ
RM6	RF, RM
RM7	RF, RM, ROB
RM8	RM, ROB
RM9	RF, IQ, LQ, SQ, RM

We evaluate the vulnerability of our heterogeneous reliability modes by executing applications from the MiBench application benchmark to estimate the FPVF for each scenario. We also evaluate the area and power overheads incurred by each reliability mode. These results are illustrated in Fig. 13.

From the results obtained, we make the following ***key observations***:

- Different heterogeneous reliability modes can reduce the full-processor vulnerability to different extents depending upon the properties of the executing application. For example, reliability modes like RM2, RM6, and RM9 reduce the processor vulnerability of `SHA` by more than 50%, but not of `Dijkstra`, even though they have similar vulnerabilities in all other reliability modes.
- Hardening specific components in the pipeline can significantly reduce the overall processor vulnerability. For example, key components like Rename Map (RM) and Reorder Buffer (ROB) effectively reduce the FPVF for all applications, as shown by the heterogeneous reliability modes RM4, RM7 and RM8. However, utilizing these hardening modes incurs significant area and power overheads.
- Certain heterogeneous reliability modes are very effective in reducing the FPVF by a large margin for very small area/power overhead. For example, RM2 and RM6 reduce the FPVF by more than 50% for <75% area and power overheads when executing `SHA`.
- Hardening all pipeline components without hardening the most highly vulnerable component of the system introduces very high overheads without reducing the vulnerability of the system significantly. This is illustrated by the reliability mode RM9, in which the ROB is not hardened. This reliability mode has area and power overheads close to ∼200% with insignificant reductions in FPVF when compared to RM4, which significantly reduces the FPVF for comparatively lower overheads.

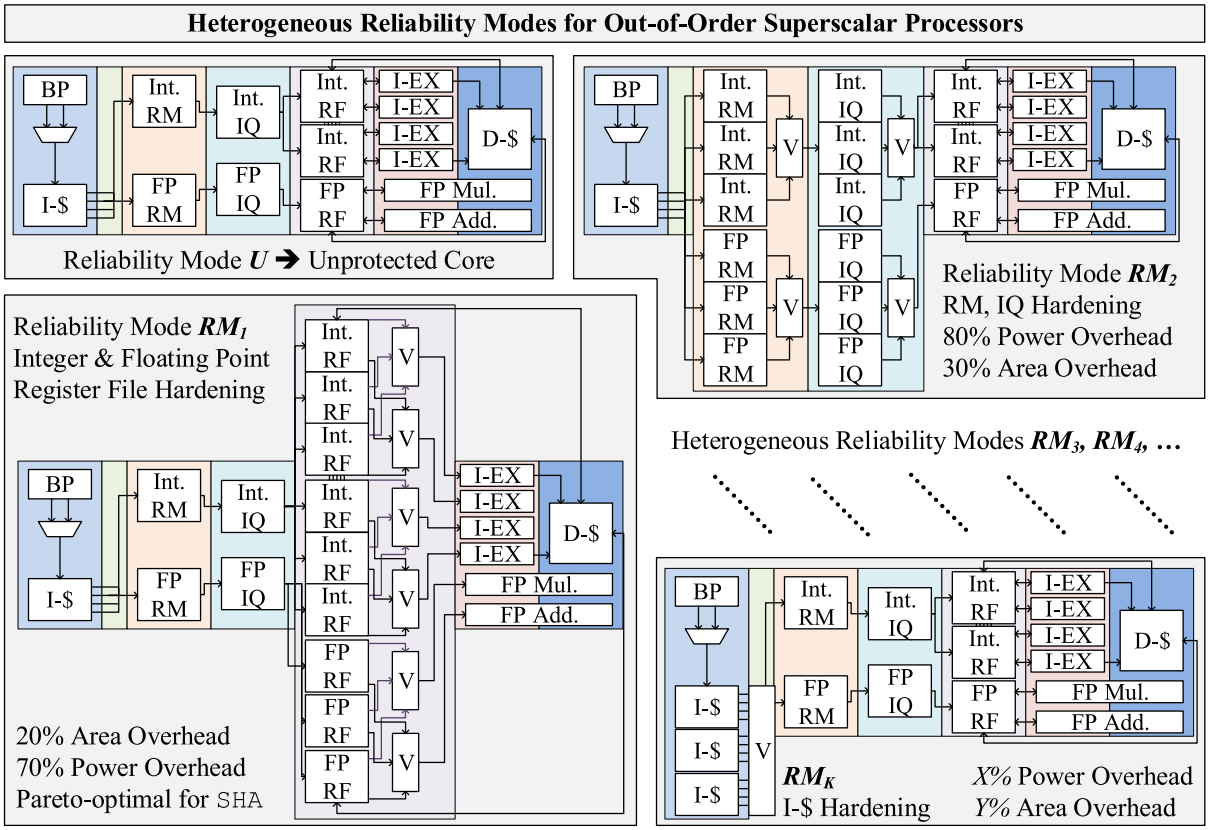

FIGURE 12. The Heterogeneous reliability modes and their Micro-architectural configurations.

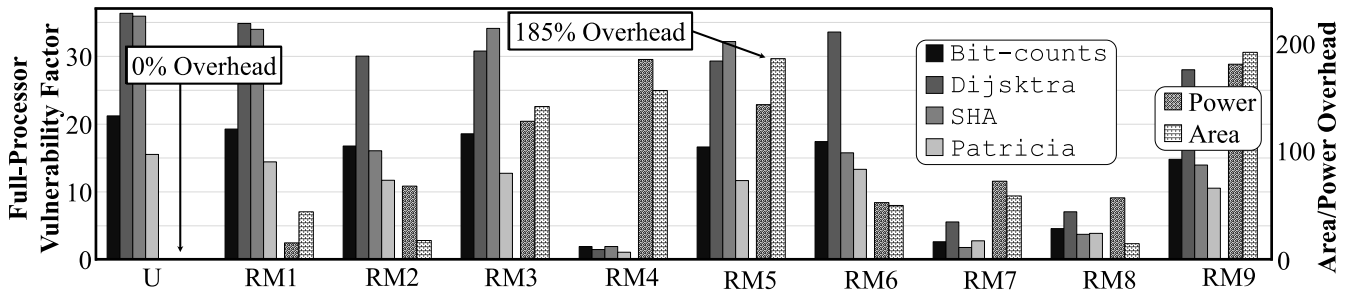

FIGURE 13. Full-Processor vulnerability factor (FPVF) and Power/Area Trade-off of Our Heterogeneous reliability modes for different MiBench applications.

Using the data gathered from the simulation of our designs, we perform a design space exploration that trades-off FPVF, area, and power overheads to extract the pareto-optimal designs that suit the target application best. The pseudo-code of the pareto-frontier extraction algorithm is presented in Algorithm 1. The corresponding results are illustrated in Fig. 14. The x-axis denotes the FPVF, whereas the y- and z-axes denote the power and area overheads, respectively. The design labeled ***U*** in all applications is the unprotected core that is highly vulnerable to soft errors. As it does not deploy any redundancy measures, it has zero area and power overhead, and hence lies on the pareto-front.

TABLE 5. Pareto-optimal reliability modes for MiBench applications.

Application	Pareto-Optimal Reliability Modes
Bit-counts	U, RM4, RM7
Dijkstra	U, RM4, RM7, RM8
Patricia	U, RM4, RM7
SHA	U, RM2, RM6, RM7, RM8
All	U, MR4, RM7, RM8

The pareto-optimal reliability modes for the applications are presented in Table 5. RM4 is pareto-optimal for all applications except SHA. The register file is highly vulnerable to soft errors during the execution of SHA and needs to be

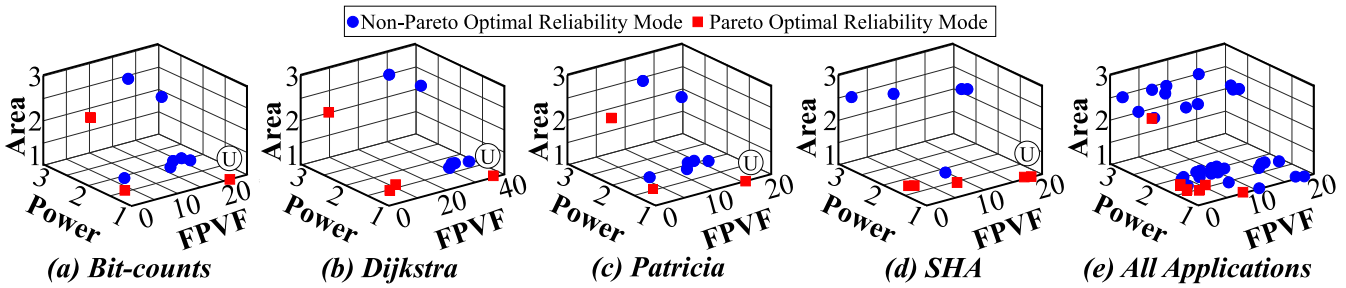

FIGURE 14. Design space exploration of our heterogeneous reliability modes for MiBench applications.

Algorithm 1 Pareto-Frontier Extraction

Input: $\{FPVF, A, P\}\forall RM_{\forall i\in[1,K]}$
Output: *OptimalReliabilityModes* (*ORM*)

```
TempSignal = 0;
TempArray(3, K) = 0;
TempArray2(3, K) = 0;
B = [FPVF, Area, Power];
for k ← 1 to 3 do
  j = 0;
  temp = B(k, :);
  for i ← 1 to 3 do
    if i! = k then
      j = j + 1;
      TempArray2(j, :) = temp − B(i, :);
    end if
  end for
  if TempArray2(1 : j, :) < 0 then
    TempSignal = TempSignal + 1;
    TempArray(TempSignal, :) = temp;
  end if
end for
if TempSignal >= 1 then
  ORM = TempArray(1 : TempSignal, :);
end if
```

hardened to reduce its vulnerability. The reliability mode ***RM7*** *is pareto-optimal for all four applications* and reduces the FPVF on average by 87% with average area and power overheads of 10% and 43%, respectively.

A super-set of the pareto-optimal reliability modes for all these applications can be selected to design a heterogeneous multi-core processor. We can build the chip by selecting the reliability modes from this super-set such that the form-factor and cost constraints are adhered to. At run-time, the required reliability modes can be switched-on/-off depending upon the power constraints of the system.

Overhead Analysis: The design-time methodology for architecture-space exploration is, fundamentally, a heuristic and is very fast in identifying a design-time configuration of the microprocessor, typically in terms of minutes. The run-time system is also a very simple heuristic and therefore requires only a few hundred cycles to reach a run-time solution, where the exact time depends upon the number of cores in the system, number of protected components, types of reliability modes, and number of executing applications.

The simulation time (different from the simulated cycles of the processor in gem5) of each experiment is in the order of several tens of minutes, and since we execute numerous fault injection campaigns, the overall time of testing is over multiple weeks. Note, the computations and simulations performed inside gem5 also depend on the computational resources of the host platform, the number of simultaneous tasks executing on the host, and resources dedicated to the simulation environment.

TABLE 6. Workload mixes and their application compositions.

Application Mix	Composition
MIX-1	[Bit-counts, Dijkstra, SHA, Patricia, Bit-counts, Dijkstra, SHA, Patricia]
MIX-2	[Bit-counts, Bit-counts, Bit-counts, Bit-counts, Dijkstra, Dijkstra, Dijkstra, Dijkstra]
MIX-3	[Bit-counts, SHA, Patricia, Bit-counts, SHA, Patricia, Bit-counts, Patricia]
MIX-4	[SHA, Patricia, SHA, Patricia, SHA, Patricia]
MIX-5	[SHA, SHA, SHA, SHA, SHA, Dijkstra]

F. RUN-TIME SYSTEM

Although this work focuses mostly on the design-time aspects of achieving heterogeneous reliability in out-of-order superscalar processors and studying their reliability vs. power/area trade-offs. In this sub-section, we present a brief overview of a run-time system for our proposed heterogeneous multi-core processor that aims at selecting the set of Pareto-optimal modes for cores such that the vulnerability of their respective applications can be minimized while satisfying their power constraints. For evaluation, we consider a 10-core processor that is composed of all the 10 heterogeneous reliability modes discussed in sub-section IV-E. We illustrate the benefits of our reliability modes by executing 5 application workload mixes, the compositions of which are presented in Table 6, on the 10-core heterogeneous processor to evaluate the power-overheads and FPVF of the multi-core system for each workload mix. The task-to-core mapping can be done using one of the following heuristics:

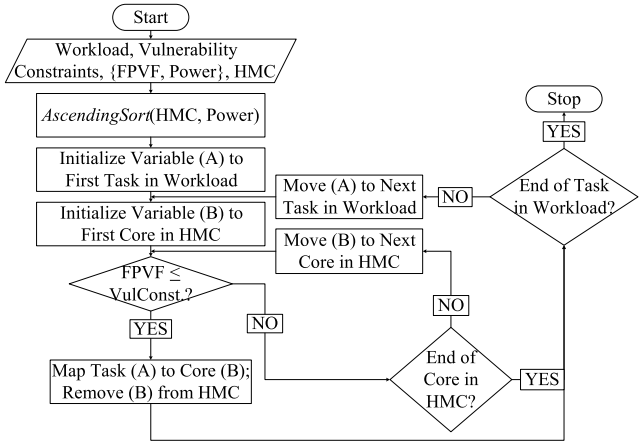

FIGURE 15. Flowchart illustrating the vulnerability-constrained power minimization task-to-core mapping policy.

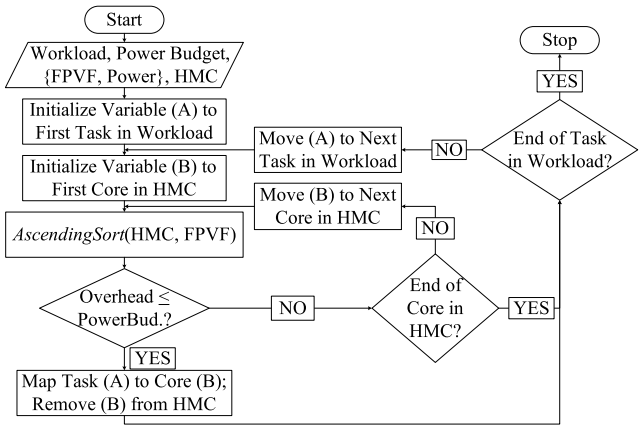

FIGURE 16. Flowchart illustrating the power-constrained vulnerability minimization task-to-core mapping policy.

(1) **Vulnerability-Constrained Power Minimization:** In this technique (see Fig. 15), we impose a vulnerability constraint on each task in the mix, i.e., each task is only mapped sequentially to a core that can successfully execute the task under the imposed vulnerability constraint. If a convenient core (one that satisfies vulnerability constraint) is not available, then the task is not scheduled immediately. The goal of this approach is to minimize the power overhead of the complete processor.

(2) **Power-Constrained Vulnerability Minimization:** This approach (Fig. 16) imposes a constraint on the maximum power overhead of the whole processor, i.e., the task-to-core mapping is stalled when the power constraint is exceeded, which is an overhead of 100% for each task in the mix. The goal of this task mapping policy is to minimize the FPVF.

The results of this evaluation are presented in Fig. 17, in which we make the following ***key observations***:

- The proposed reliability modes can be deployed in a heterogeneous multi-core processor to reduce the power overheads of the executing application workloads, based on the application's workload requirement.
- The proposed reliability modes can either be used to minimize the power overhead or the full-processor vulnerability factor as illustrated by the two task mapping policies.

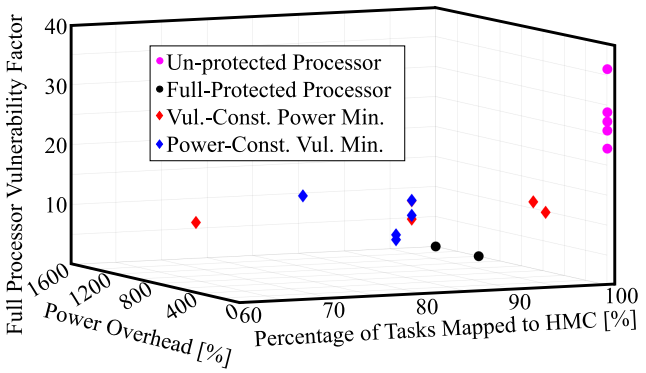

FIGURE 17. Run-time task mapping analysis of HMC.

Although 100% task mapping is not achieved as in the un-protected or full-protected case, this can be resolved by efficiently selecting the reliability modes to be deployed in the *HMC* considering the potential application workloads and/or by using a task mapping algorithm that can efficiently schedule the tasks to processor cores.

G. STATE COMPRESSION TECHNIQUES

Checkpointing and Rollback is an effective way of guaranteeing reliability at the software layer by means of providing both spatial and temporal redundancy. A checkpoint is a snapshot of the processor state at any instant in time. Checkpoints allow the system to roll back to the previous safe states in case a failure is detected and re-execute instructions.

Fig. 18 presents an overview of the methodology that we use for checkpointing and state compression. Checkpoints are typically inserted intermittently into the target application for periodic state retention and, if required, rollback to an earlier processor state, i.e., in case of faulty execution. Typically, the collected processor's state information is stored in the main memory or off-chip non-volatile memory, which can still be used for a rollback in case of power-off. In our case, to reduce the size of checkpointing data, we introduce another stage of state compression, that utilizes state-of-the-art compression techniques to generate a wide range of compressed checkpoint variants. The optimal compressed variant can be selected based on the system's resource constraints and available on-/off-chip memory. In case a fault is detected in the current processor state, during the application execution, the previous safe-state is decompressed and rolled back to ensure the correct execution of the application.

The standard checkpointing mechanism deployed by gem5 comes with certain caveats. This technique does not preserve cache and pipeline states in a checkpoint because

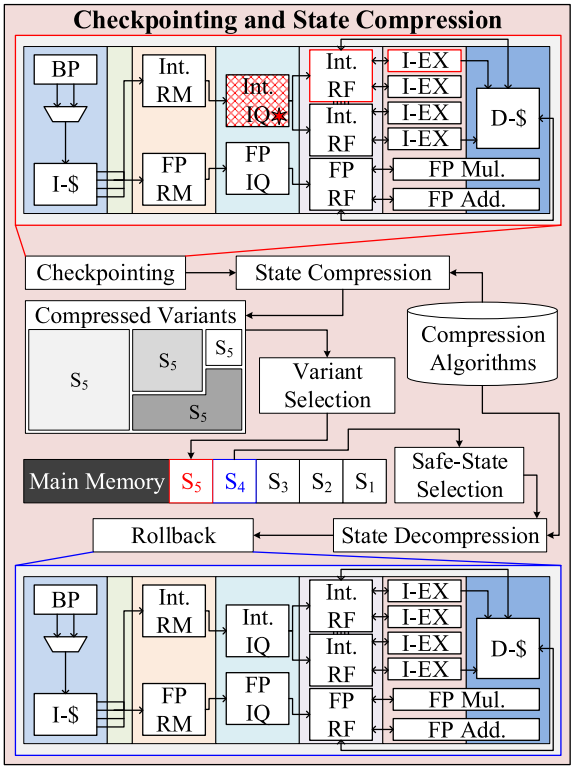

FIGURE 18. Overview of the methodology for checkpointing and state compression.

of which frequent restoration from such checkpoints results in a performance loss if deployed in real-world systems. Therefore, we explore techniques like DMTCP [27], [28] that implement checkpoints in the Linux process to store the processor state as well as data present in the cache hierarchy. The backend checkpointing mechanism of DMTCP is accessible to the programmer via numerous APIs. These APIs can be used in conjunction with the front-end gem5 pseudo-instructions for checkpoint creation/recovery. Since these software-based checkpoints are often large, the checkpoint is compressed using gzip and HBICT to save memory. HBICT [29], [30] provides DMTCP support for delta-compression (relative to the previous compression), which is further compressed using gzip (a combination of lossless data compression algorithms like LZ77 and Huffman coding).

We investigate the effectiveness of these techniques in all possible combinations, by applying them one after the other, on applications from the MiBench application benchmark suite by simulating them on the ALPHA core using gem5. The results of this experiment are presented in Fig. 19. From these results, we make the following ***key observations:***

- the combination of DMTCP and gzip is highly successful in reducing the checkpoint size by $\sim 6\times$
- the combination of DMTCP, HBICT, and gzip techniques reduce the checkpoint size by $\sim 5.7\times$.

HBICT, which utilizes delta-compression, requires all previous checkpoints for efficient rollback. Since the base file

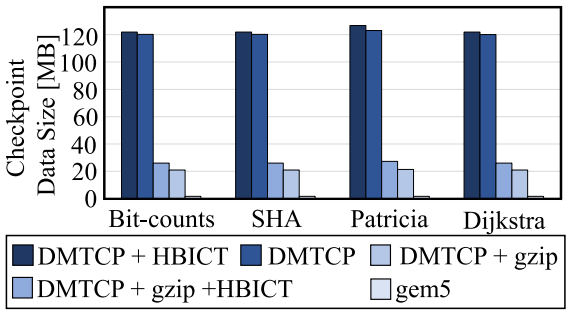

FIGURE 19. Effectiveness of state compression techniques in reducing state size.

size of HBICT+DMTCP is $1.03\times$ larger than the file size of DMTCP, the effectiveness of the combined state compression technique (DMTCP+HBICT+gzip), with respect to DMTCP, is reduced.

V. RELATED WORK

Reliability is a major research challenge that is being tackled by the community at large via global initiatives like the NSF's Variability Expedition[1] and DFG's SPP 1500 Priority Program.[2] Research works from the academia and industry alike have addressed the challenges associated with technology scaling across the layers of the computing stack.

A. MITIGATION STRATEGIES

The work in [38] presents the *Razor* approach, which can be used to dynamically detect and correct timing errors by monitoring the error rate at run-time to tune the circuit's supply voltage. The adaptive approach presented in [39] enables per-core dual modular redundancy (DMR) through the means of DVFS to offer a stable soft error rate (SER). An OS-level dynamic reliability management system for heterogeneous architectures for achieving an optimal trade-off between reliability (lifetime) and power/performance efficiency is presented in [40]. A software-level technique is presented in [9], which is used to detect errors by duplicating instructions during compile time by using different variables and registers for new instructions. A software-controlled fault-tolerance scheme is proposed in [41] that allows programmers and designers to trade-off between performance and reliability based on the system's requirement. Luo *et al.* [42] quantify the tolerance of application to memory errors to propose several new hardware/software heterogeneous-reliability memory systems to reduce their vulnerabilities and data-center costs. A hardware-software co-design approach for soft error mitigation in embedded systems has been proposed in [43], which includes a generic software hardening environment that is used to generate a "hardened" code variant and a hardening infrastructure called *FTUnshades* in FPGAs, which is used to access the reliability of the complete hardware-software stack of the embedded system.

[1] http://www.variability.org/
[2] http://spp1500.itec.kit.edu

B. RELIABILITY MODELING

The work in [44] demonstrates the concept of Program Vulnerability Factor, which captures the architecture-level fault-masking properties of the underlying program while exhibiting workload-driven changes in the AVF for all architectural components. Li *et al.* [45] analyze the correlation between the soft error rate and the energy consumption behavior of on-chip data caches. This involves analyzing (1) the leakage energy optimizations on soft errors, and (2) the energy overheads of protecting on-chip memories against soft errors. A software-level technique proposed in [46] introduces transient fault tolerance in a multi-core system by exploiting process-level redundancy (PLR) to create multiple application threads and compare them to ensure correct execution of the application. A software-level approach to enable self-adaptive reliability for multi-/many-core systems is proposed in [47] by activating redundancy measures based on the application's dependability requirements. A simultaneous and redundantly threaded (SRT) processor is presented in [48], which provides transient fault tolerance with significantly higher performance. Redundant copies of the program threads are executed simultaneously on the SRT to ensure accurate application execution. Kriebel *et al.* [49] analyze and present the reliability issues of on-chip memory systems to propose a reliability-aware reconfigurable last-level cache architecture that adapts the cache parameters to concurrently execute multi-threaded workloads at run-time to minimize their vulnerabilities. A soft error-aware cache architectural space-exploration methodology is presented in [50] for varying the application workloads and cache parameters for the complete cache hierarchy. An adaptive soft-error resilience (ASER) approach is presented in [51] by proposing and managing reliability-heterogeneous dark silicon many-core processors (darkRHPs). The proposed darkRHPs deploy redundancy at the architecture level, i.e., hardening either the full-processor pipeline of an in-order LEON3 processor and/or caches. The work in [52] presents an approach that exploits the on-chip dark-silicon to synergistically mitigate reliability and variability challenges associated with transistor technology scaling. An overview of different heterogeneous fault-tolerance schemes for both hardware and software layers is presented in [11], which also provides an initial proof-of-concept of this work.

This work, on the other hand, focuses on generating and exploring a wide range of heterogeneous reliability modes using two key approaches, i.e., (1) Redundancy, by hardening different combinations of the pipeline components for an out-of-order superscalar processor, and (2) Checkpointing, by reducing the size of the checkpoint data using efficient compression techniques.

VI. CONCLUSION

In this work, we presented a novel architectural-space generation and exploration methodology that is used to develop a wide range of heterogeneous reliability modes for out-of-order superscalar processors. By analyzing the architectural vulnerability of key pipeline components, we have observed that the pipeline components have varying architectural vulnerability factors for different application workloads. Based on this observation, we propose to harden the pipeline components in multiple different combinations with varying levels of reliability to cater to the application's requirement while minimizing the power/area overhead. We have also extended the AVF metric to define the Full-Processor Vulnerability Factor (FPVF), which can be used to estimate the processor's vulnerability as a whole, for a given application workload, instead of analyzing the vulnerability of each component. The pareto-optimal reliability mode RM7 is successful in reducing the FPVF by 87% on average, with area and power overheads of 10% and 43%, respectively. We have also illustrated the benefits of our proposed approach at run-time by evaluating two simple task-mapping strategies, which can be used to either minimize power or processor vulnerability based on the system's constraints. To further enhance our design space for heterogeneous reliability, we also investigate effective state-compression techniques to reduce the data size of a checkpoint by $\sim 6\times$. Our studies illustrate that in power-constrained scenarios, enabling reliability at a fine granularity, and deploying reliability-heterogeneous super-scalar out-of-order processors bear a significant potential for real-world systems, especially when considering diverse vulnerability profiles of different applications, which can further vary depending upon their input workloads.

ACKNOWLEDGMENT

The authors would like to thank Arun Subramaniyan, Segnon Jean Bruno Ahandagbe, and Hariharan Sivaraman for the initial technical discussions.

REFERENCES

[1] 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 *Proc. 50th Annu. Design Automat. Conf.*, Austin, TX, USA, May/Jun. 2013, p. 99. doi: 10.1145/2463209.2488857.

[2] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Trans. Device Mater. Rel.*, vol. 5, no. 3, pp. 305–316, Sep. 2005.

[3] G. P. Saggese, N. J. Wang, Z. T. Kalbarczyk, S. J. Patel, and R. K. Iyer, "An experimental study of soft errors in microprocessors," *IEEE Micro*, vol. 25, no. 6, pp. 30–39, Nov. 2005. doi: 10.1109/MM.2005.104.

[4] S. Feng, S. Gupta, A. Ansari, and S. A. Mahlke, "Shoestring: Probabilistic soft error reliability on the cheap," in *Proc. 15th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Pittsburgh, PA, USA, Mar. 2010, pp. 385–396. doi: 10.1145/1736020.1736063.

[5] S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, Nov./Dec. 2005. doi: 10.1109/MM.2005.110.

[6] T. Li, R. Ragel, and S. Parameswaran, "Reli: Hardware/software Checkpoint and Recovery scheme for embedded processors," in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, Dresden, Germany, Mar. 2012, pp. 875–880. doi: 10.1109/DATE.2012.6176621.

[7] C.-C. J. Li and W. K. Fuchs, "CATCH-compiler-assisted techniques for checkpointing," in *Proc. 20th Int. Symp. Fault-Tolerant Comput.*, Newcastle Upon Tyne, U.K., Jun. 1990, pp. 74–81. doi: 10.1109/FTCS.1990.89337.

[8] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt, "Detailed design and evaluation of redundant multi-threading alternatives," in *Proc. 29th Annu. Int. Symp. Comput. Archit.*, Anchorage, AK, USA, May 2002, pp. 99–110. doi: 10.1109/ISCA.2002.1003566.

[9] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Error detection by duplicated instructions in super-scalar processors," *IEEE Trans. Rel.*, vol. 51, no. 1, pp. 63–75, Mar. 2002. doi: 10.1109/24.994913.
[10] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August, "SWIFT: Software implemented fault tolerance," in *Proc. 3rd Int. Symp. Code Gener. Optim.*, San Jose, CA, USA, Mar. 2005, pp. 243–254. doi: 10.1109/CGO.2005.34.
[11] S. Rehman, F. Kriebel, B. S. Prabakaran, F. Khalid, and M. Shafique, "Hardware and software techniques for heterogeneous fault-tolerance," in *Proc. IEEE 24th Int. Symp. On-Line Test. Robust Syst. Design*, Platja d'Aro, Spain, Jul. 2018, pp. 115–118. doi: 10.1109/IOLTS.2018.8474219.
[12] T. C. May and M. H. Woods, "Alpha-particle-induced soft errors in dynamic memories," *IEEE Trans. Electron Devices*, vol. 26, no. 1, pp. 2–9, Jan. 1979.
[13] G. R. Srinivasan, P. C. Murley, and H. K. Tang, "Accurate, predictive modeling of soft error rate due to cosmic rays and chip alpha radiation," in *Proc. IEEE Int. Rel. Phys. Symp.*, Apr. 1994, pp. 12–16.
[14] T. J. O'Gorman, "The effect of cosmic rays on the soft error rate of a DRAM at ground level," *IEEE Trans. Electron Devices*, vol. 41, no. 4, pp. 553–557, Apr. 1994.
[15] P. Hazucha and C. Svensson, "Impact of CMOS technology scaling on the atmospheric neutron soft error rate," *IEEE Trans. Nucl. Sci.*, vol. 47, no. 6, pp. 2586–2594, Dec. 2000.
[16] Gaisler. *Leon3ft Fault-Tolerant Processor*. Accessed: May 16, 2019. [Online]. Available: https://www.gaisler.com/index.php/products/processors/leon3ft
[17] R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM J. Res. Develop.*, vol. 6, no. 2, pp. 200–209, Apr. 1962. doi: 10.1147/rd.62.0200.
[18] M. M. Dickinson, J. B. Jackson, and G. C. Randa, "Saturn V launch vehicle digital computer and data adapter," in *Proc. Fall Joint Comput. Conf. I*, New York, NY, USA, Oct. 1964, pp. 501–516. doi: 10.1145/1464052.1464099.
[19] R. E. Kessler, "The Alpha 21264 microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24–36, Mar./Apr. 1999. doi: 10.1109/40.755465.
[20] M. Johnson, *Superscalar Microprocessor Design* (Prentice Hall Series in Innovative Technology). Upper Saddle River, NJ, USA: Prentice-Hall, 1991.
[21] W. Hwu and Y. N. Patt, "HPSm, a high performance restricted data flow architecture having minimal functionality," in *Proc. 13th Annu. Int. Symp. Comput. Archit.*, Los Alamitos, CA, USA, 1986, pp. 297–306. doi: 10.1145/17356.17391.
[22] C. C. Corporation. (Jul. 1999). *Alpha 21264 Microprocessor Hardware Reference Manual*. Alpha 21264 Manual. [Online]. Available: http://www.archive.ece.cmu.edu/~ece447/s13/lib/exe/fetch.php?media=21264hrm.pdf
[23] P. E. Gronowski, W. J. Bowhill, R. P. Preston, M. K. Gowan, and R. L. Allmon, "High-performance microprocessor design," *IEEE J. Solid-State Circuits*, vol. 33, no. 5, pp. 676–686, May 1998.
[24] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. M. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture*, San Diego, CA, USA, Dec. 2003, pp. 29–42. doi: 10.1109/MICRO.2003.1253181.
[25] 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," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011. doi: 10.1145/2024716.2024718.
[26] 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 *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, New York, NY, USA, Dec. 2009, pp. 469–480. doi: 10.1145/1669112.1669172.
[27] J. Ansel, K. Arya, and G. Cooperman, "DMTCP: Transparent checkpointing for cluster computations and the desktop," in *Proc. 23rd IEEE Int. Symp. Parallel Distrib. Process.*, Rome, Italy, May 2009, pp. 1–12. doi: 10.1109/IPDPS.2009.5161063.
[28] *DMTCP*. Accessed: May 14, 2019. [Online]. Available: http://dmtcp.sourceforge.net/
[29] S. Agarwal, R. Garg, M. S. Gupta, and J. E. Moreira, "Adaptive incremental checkpointing for massively parallel systems," in *Proc. 18th Annu. Int. Conf. Supercomput.*, Saint Malo, France, Jun./Jul. 2004, pp. 277–286. doi: 10.1145/1006209.1006248.
[30] *HBICT*. Accessed: May 14, 2019. [Online]. Available: http://hbict.sourceforge.net/index.html
[31] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. 4th Annu. IEEE Int. Workshop Workload Characterization*, Dec. 2001, pp. 3–14.
[32] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. S. Emer, "SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Santa Rosa, CA, USA, Apr. 2017, pp. 249–258. doi: 10.1109/ISPASS.2017.7975296.
[33] H. Ziade, R. A. Ayoubi, and R. Velazco, "A survey on fault injection techniques," *Int. Arab J. Inf. Technol.*, vol. 1, no. 2, pp. 171–186, 2004. [Online]. Available: http://www.iajit.org/ABSTRACTS-2.htm#04
[34] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel, "Reliable software for unreliable hardware: Embedded code generation aiming at reliability," in *Proc. 7th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Taipei, Taiwan, Oct. 2011, pp. 237–246. doi: 10.1145/2039370.2039408.
[35] S. S. Mukherjee, J. S. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *Proc. 11th Int. Symp. High-Perform. Comput. Archit.*, San Francisco, CA, USA, Feb. 2005, pp. 243–247. doi: 10.1109/HPCA.2005.37.
[36] A. Dixit and A. Wood, "The impact of new technology on soft error rates," in *Proc. Int. Rel. Phys. Symp.*, Apr. 2011, pp. 5B.4.1–5B.4.7.
[37] J. Gaisler, "A portable and fault-tolerant microprocessor based on the SPARC v8 architecture," in *Proc. Int. Conf. Dependable Syst. Netw.*, Bethesda, MD, USA, Jun. 2002, pp. 409–415. doi: 10.1109/DSN.2002.1028926.
[38] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner, "Razor: Circuit-level correction of timing errors for low-power operation," *IEEE Micro*, vol. 24, no. 6, pp. 10–20, Nov./Dec. 2004. doi: 10.1109/MM.2004.85.
[39] R. Vadlamani, J. Zhao, W. Burleson, and R. Tessier, "Multicore soft error rate stabilization using adaptive dual modular redundancy," in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, Dresden, Germany, Mar. 2010, pp. 27–32. doi: 10.1109/DATE.2010.5457242.
[40] A. Baldassari, C. Bolchini, and A. Miele, "A dynamic reliability management framework for heterogeneous multicore systems," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst.*, Cambridge, U.K., Oct. 2017, pp. 1–6. doi: 10.1109/DFT.2017.8244440.
[41] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, D. I. August, and S. S. Mukherjee, "Software-controlled fault tolerance," *ACM Trans. Archit. Code Optim.*, vol. 2, no. 4, pp. 366–396, Dec. 2005. doi: 10.1145/1113841.1113843.
[42] Y. Luo, S. Govindan, B. Sharma, M. Santaniello, J. Meza, A. Kansal, J. Liu, B. Khessib, K. Vaid, and O. Mutlu, "Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Atlanta, GA, USA, Jun. 2014, pp. 467–478. doi: 10.1109/DSN.2014.50.
[43] S. Cuenca-Asensi, A. Martinez-Alvarez, F. Restrepo-Calle, F. R. Palomo, H. Guzman-Miranda, and M. A. Aguirre, "A novel co-design approach for soft errors mitigation in embedded systems," *IEEE Trans. Nucl. Sci.*, vol. 58, no. 3, pp. 1059–1065, Jun. 2011.
[44] V. Sridharan and D. R. Kaeli, "Eliminating microarchitectural dependency from architectural vulnerability," in *Proc. IEEE 15th Int. Symp. High Perform. Comput. Archit.*, Raleigh, NC, USA, Feb. 2009, pp. 117–128. doi: 10.1109/HPCA.2009.4798243.
[45] L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Soft error and energy consumption interactions: A data cache perspective," in *Proc. Int. Symp. Low Power Electron. and Design*, Newport Beach, CA, USA, Aug. 2004, pp. 132–137. doi: 10.1145/1013235.1013273.
[46] A. Shye, T. Moseley, V. J. Reddi, J. Blomstedt, and D. A. Connors, "Using process-level redundancy to exploit multiple cores for transient fault tolerance," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Edinburgh, U.K., Jun. 2007, pp. 297–306. doi: 10.1109/DSN.2007.98.
[47] C. Bolchini, M. Carminati, and A. Miele, "Self-adaptive fault tolerance in multi-/many-core systems," *J. Electron. Test.*, vol. 29, no. 2, pp. 159–175, 2013. doi: 10.1007/s10836-013-5367-y.
[48] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," in *Proc. 27th Int. Symp. Comput. Archit.*, Vancouver, BC, Canada, Jun. 2000, pp. 25–36. doi: 10.1109/ISCA.2000.854375.

[49] F. Kriebel, S. Rehman, A. Subramaniyan, S. J. B. Ahandagbe, M. Shafique, and J. Henkel, "Reliability-aware adaptations for shared last-level caches in multi-cores," *ACM Trans. Embedded Comput. Syst.*, vol. 15, no. 4, p. 67, Aug. 2016. doi: 10.1145/2961059.

[50] A. Subramaniyan, S. Rehman, M. Shafique, A. Kumar, and J. Henkel, "Soft error-aware architectural exploration for designing reliability adaptive cache hierarchies in multi-cores," in *Proc. Design, Automat. Test Eur. Conf. Exhib.*, Lausanne, Switzerland, Mar. 2017, pp. 37–42. doi: 10.23919/DATE.2017.7926955.

[51] F. Kriebel, S. Rehman, D. Sun, M. Shafique, and J. Henkel, "ASER: Adaptive soft error resilience for reliability-heterogeneous processors in the dark silicon era," in *Proc. 51st Annu. Design Automat. Conf.*, San Francisco, CA, USA, Jun. 2014, p. 1–6. doi: 10.1145/2593069.2593094.

[52] F. Kriebel, M. Shafique, S. Rehman, J. Henkel, and S. Garg, "Variability and reliability awareness in the age of dark silicon," *IEEE Des. Test*, vol. 33, no. 2, pp. 59–67, Apr. 2016. doi: 10.1109/MDAT.2015.2439640.

BHARATH SRINIVAS PRABAKARAN (S'19) received the Bachelor of Engineering degree in electrical and electronics and the Master of Science degree in biological sciences from the Birla Institute of Technology and Science (BITS), Pilani, in 2017. He is currently pursuing the Ph.D. degree with the Computer Architecture and Robust Energy-Efficient Technologies (CARE-Tech.) Research Group, Institute of Computer Engineering, Technische Universität Wien (TU Wien), Austria. He was a Visiting Researcher with TU Dresden for a span of one year in 2016, where he completed his master thesis. His research interests include fault-tolerant computing, wearable architectures, healthcare systems, energy-efficient technologies, and embedded machine learning.

MIHIKA DAVE received the Bachelor of Engineering degree from BITS-Pilani, India, in 2016, where she secured the first rank in the Department of Electrical and Electronics Engineering and received a Bronze Medal in the entire batch of students across all the Science and Engineering Departments, and the Master of Science degree in computer science from the University of Illinois at Urbana-Champaign with a specialization in natural language processing, in 2018. She is currently a Software Engineer with Facebook, Inc. Her main research interests include heterogeneous fault-tolerance, machine learning, and natural language processing. She was a recipient of several scholarships and awards, such as the DAAD-WISE Scholarship, Michal S. Hughes Award in Software Engineering, and BITS-Pilani Merit Scholarship.

FLORIAN KRIEBEL received the M.Sc. degree in computer science from the Karlsruhe Institute of Technology (KIT), Germany, in 2013. He is currently a University Assistant with the Computer Architecture and Robust Energy-Efficient Technologies (CARE-Tech.) Research Group, Institute of Computer Engineering, Technische Universität Wien (TU Wien), Austria. His current research interests include dependable computing, cross-layer reliability modeling, and optimization. He has received the CODES+ISSS 2011 and 2015 Best Paper Awards.

SEMEEN REHMAN received the Ph.D. degree in computer science from KIT, Germany, in July 2015. Before that, she has been a Postdoctoral Researcher with the Technische Universität Dresden (TU Dresden) and Karlsruhe Institute of Technology (KIT), Germany, since 2015. She is currently on a Laufbahnstelle (Tenure-Track Assistant Professor) position with the Institute of Computer Technology (ICT), Faculty of Electrical Engineering and Information Technology, Technische Universität Wien (TU Wien). She has coauthored one book, multiple book chapters, and more than 30 publications in premier journals and conferences. Her main research interests include dependable systems, cross-layer design for error resiliency with a focus on run-time adaptations, emerging computing paradigms, such as approximate computing, hardware security, energy-efficient computing, embedded systems, MPSoCs, the IoT, and CPS. Dr. Rehman has contributed key ideas that have led to various DFG projects, such as GetSURE and GetSURE-II at the KIT, which focused on enabling reliability across multiple software and hardware layers. At the Chair for Processor Design at Technische Universität Dresden, Germany, she initiated the research on Reconfigurable Approximate Computing. She received the CODES+ISSS 2011 and 2015 Best Paper Awards, DATE 2017 Best Paper Award Nomination, several HiPEAC Paper Awards, Richard Newton Young Student Fellow Award at DAC 2015, and Research Student Award at KIT, in 2012. She has served on the TPC of multiple premier conferences on design automation and embedded systems (such as DATE and CASES) and has (co-)chaired sessions at the DATE 2017, 2018, and 2019 conferences.

MUHAMMAD SHAFIQUE (M'11–SM'16) received the Ph.D. degree in computer science from the Karlsruhe Institute of Technology (KIT), Germany, in January 2011. Before, he was with Streaming Networks Pvt. Ltd., where he was involved in research and development of video coding systems for several years. He has been a Full Professor with the Computer Architecture and Robust Energy-Efficient Technologies (CARE-Tech.), Institute of Computer Engineering, Technische Universität Wien (TU Wien), Austria, since November 2016. He holds one U.S. patent and has (co-)authored six books, more than ten book chapters, and over 200 articles in premier journals and conferences. His research interests include computer architecture, power-/energy-efficient systems, robust computing, hardware security, brain-inspired computing trends, such as neuromorphic and approximate computing, hardware and system-level design for machine learning and AI, emerging technologies and nanosystems, FPGAs, MPSoCs, and embedded systems. His research has a special focus on cross-layer modeling, design, and optimization of computing and memory systems, and their deployment in use cases from the Internet-of-Things (IoT), cyber-physical systems (CPS), and ICT for development (ICT4D) domains. Dr. Shafique is a member of the ACM, SIGARCH, SIGDA, SIGBED, and HIPEAC, and a Senior Member of the IEEE Signal Processing Society (SPS). He has given several Keynotes, Invited Talks, and Tutorials. He has served on the TPC of numerous prestigious IEEE/ACM conferences. He received the 2015 ACM/SIGDA Outstanding New Faculty Award, six gold medals in his educational career, and several best paper awards and nominations at prestigious conferences, such as CODES+ISSS, DATE, DAC and ICCAD, Best Master Thesis Award, DAC'14 Designer Track Best Poster Award, IEEE TRANSACTIONS OF COMPUTER "Feature Paper of the Month" Awards, and Best Lecturer Award. He has also organized many special sessions at premier venues and served as the Guest Editor for the *IEEE DESIGN AND TEST MAGAZINE* and the IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING.