A Fine-Grained Soft Error Resilient Architecture under Power Considerations

Sajjad Hussain*, Muhammad Shafique†, Jörg Henkel*
*Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany
{sajjad.hussain, henkel}@kit.edu
†Department of Computer Engineering Vienna University of Technology, Austria
{muhammad.shafique}@tuwien.ac.at

***Abstract*—Besides the limited power budgets and the dark-silicon issue, soft error is one of the most critical reliability issues in computing systems fabricated using nano-scale devices. During the execution, different applications have varying performance, power/energy consumption and vulnerability properties. Different trade-offs can be devised to provide required resiliency within the allowed power constraints. To exploit this behavior, we propose a novel soft error resilient architecture and the corresponding run-time system that enables power-aware fine-grained resiliency for different processor components. It selectively determines the reliability state of various components, such that the overall application reliability is improved under a given power budget. Our architecture saves power up to 16% and reliability degradation up to 11% compared to state-of-the-art techniques.**

I. Introduction and Related Work

In the nano era, due to the shrinking feature sizes and rapid technology scaling, reliability has emerged as one of the primary design criteria for on-chip systems [1]. Soft errors for a chip are predicted to increase with future technology nodes due to smaller feature sizes, operating voltages, increased clock speed and low critical charge [2], [3]. Reliability enhancing techniques [2], [4]–[8] to mitigate soft errors mostly rely on coarse-grained redundancy (i.e., full-scale duplication or triplication) and hence, incur significant performance and/or power overhead. They do not consider the soft error vulnerability and power variations at a finer-granularity (i.e., selective and/or component-level redundancy) that can be leveraged to achieve better reliability management under tighter power constraints. Moreover, these techniques cannot exploit the variations during different execution phases of a given application at run time. *To address above challenges, we study the following research questions in this paper, which have not been addressed by earlier state-of-the-art.*

- How the soft error vulnerability vary at a fine-grained level for several components of a given processor?
- How does the soft error vulnerability vary across different execution phases of a given application?
- How does the power consumption of different applications temporarily varies at the fine-grained level for different hardware components of a given processor?
- How can different reliability enhancing techniques affect the overall vulnerability and the power consumption of the processor? How can the designers effectively utilize the vulnerability-power trade-off with a fine-granularity control to ensure power constraints at run time?

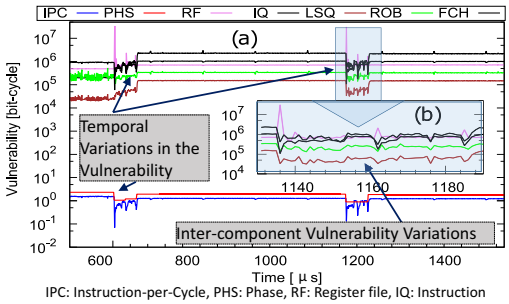

Figure 1: (a) Vulnerability variation in different components (b) and their inter-component variations in a particular execution time phase.

A. Motivational Case Study: Analyzing the Vulnerability, Power and Performance Variations

To address the above research questions, we thoroughly analyzed the vulnerability and power of different applications in different processors components at a fine-granularity. Our analysis in Fig. 1-3 illustrates the following key observations.

- Different processor components like instruction queue (IQ), reorder-buffer (RO), register file (RF), load-store queue (LS), fetch unit (FC), etc. depict diverse soft error vulnerability during the FFT application execution.
- Different applications may have distinct resilience to soft errors; see CRC and SUSAN in Fig. 2. Even, different architectural components may have different vulnerability and power consumption behavior during an application execution, as shown by the variations in Fig. 1.
- An application program shows varying instruction-per-cycle (IPC) behavior which can be used to distinguish different performance phases, as shown in Fig. 1.
- For different reliability features, each application shows distinct and varying power, performance, energy and vulnerability characteristics, for example, see labels (1), (2) and (3) in Fig. 2. The analysis illustrates this behavior for benchmark applications using two reliability features, i.e., dual-modular redundancy with re-execution (DMR-RE) and triple-modular redundancy (TMR).
- Finally, the analyses in Fig. 3 show run-time variations in performance, power, and vulnerability across different applications.

978-3-9819263-2-3/DATE19/©2019 EDAA

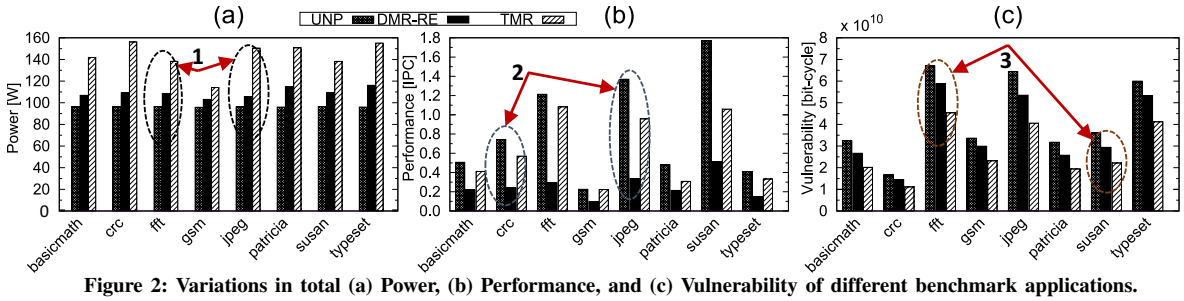

Figure 2: Variations in total (a) Power, (b) Performance, and (c) Vulnerability of different benchmark applications.

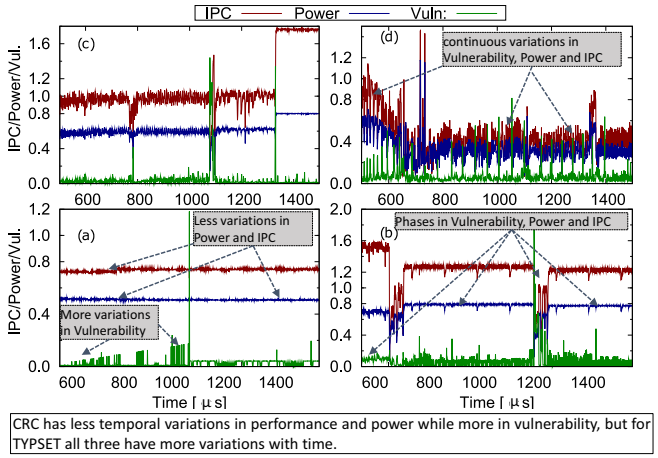

Figure 3: Variations in run-time performance, power and vulnerability of (a) CRC, (b) FFT, (c) SUSAN and (d) TYPESET

Therefore, the main research question that we target in this paper is: how can one leverage the temporal variations in soft error vulnerability of different processor components for different execution phases of applications, as well as the resulting diverse performance and power profiles, to dynamically manage the overall system's reliability in a highly-efficient way, while respecting the given constraint on the total power budget.

B. Our Novel Contributions

This paper proposes a novel fine-grained soft error resilient architecture and run-time system that leverages the aforementioned variations in performance, power and vulnerabilities to maximize the overall reliability under a given power constraint. To enable this, we make the following key contributions.

1) **Fine-Grained Reliability Analyses (Section I-A):** A thorough and precise power, vulnerability and performance analysis has been performed for different applications with different architectural configurations and application settings.
2) **Architectural Support for Fine-Grained Resilience (Section III-A):** The processor components are equipped with different reliability features, which can be controlled at a finer granularity at run time.
3) **Phase Identification (Section III-B):** We employ a lightweight phase detection technique that identifies different execution phases of a given application based on instruction-per-cycle.
4) **Power-Aware Fine-Grained Reliability Management (Section IV):** For each identified phase, a novel run-time system determines the appropriate reliability mode of each component, so that the overall reliability is maximized under a given power budget.

II. Models

A. Processor Model

This paper considers an out-of-order processor having fetch, decode, re-name, issue, execute, write-back and commit stages. Our processor model consists of *n* different pipeline components, i.e., $C = c_1, c_2, .., c_n$ equipped with different *k* reliability features/modes, i.e., $R = r_1, r_2, .., r_k$, where r_1 is the completely unreliable mode and r_k is the fully reliable mode.

B. Application Program Model

We assume that an application program consists of *m* different execution phases, such that $F = f_1, f_2, .., f_m$. Each phase f_i has to meet its execution time budget e_i to respect the overall deadline constraint. Thus, the execution time budgets of *m* phases of an application can be denoted as $E = e_1, e_2, .., e_m$.

C. Vulnerability Model

We use architectural vulnerability factor (AVF) as soft-error vulnerability model defined as the probability of failures in a system, as the proportion of hardware bits in the processor carrying useful data for the system [10], [11]. In other words, a bit *"b"* in the processor is considered vulnerable at time *"t"* if an error in this bit will be consumed by the processor at any time *"t+τ"*, with the potential to cause failures in the execution and/or program output. The system vulnerability (bit-cycles) is the sum of the bits in the processor that are vulnerable during program execution. The total vulnerability is $V = \sum_{i=1}^{n} v_i$ Where v_i is the vulnerability of a component *i*. From [15], the reliability of any architectural component having vulnerability v_i is estimated using:

$$R_i(v_i, t, \lambda) = e^{(-\lambda * v_i * t)} \quad (1)$$

And the vulnerability of different pipeline component during a application phases *t* is $v_i(t)$, *s.t.,* $t\epsilon[1, ..m], i\epsilon[1, ..n]$.

D. Power Model

The power consumption of a pipeline component, i.e., $P_i(t)$, *s.t.,* $t\epsilon[1, ..m], i\epsilon[1, ..n]$ during different application phases f_m, is estimated using McPAT power model.

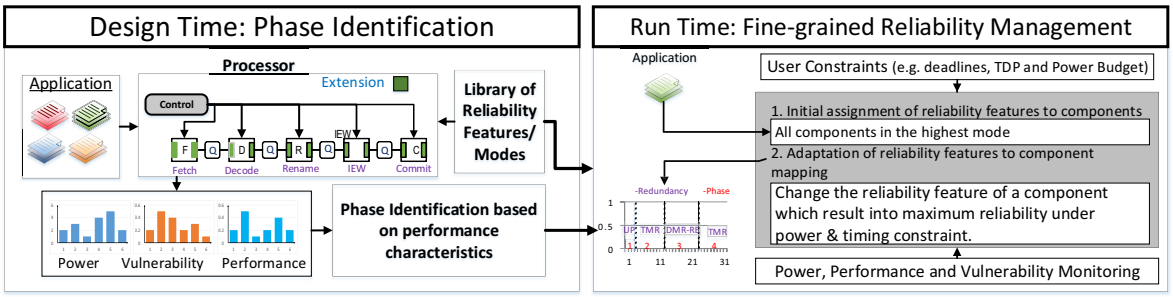

Figure 4: Overview of our proposed technique.

III. Power-Aware Fine-Grained Resiliency

Fig.4 shows an overview of our novel approach for power-aware fine-grained soft error resiliency. In our scheme, each application is executed on a processor, and performance phases are identified at design time. At run time, vulnerability and power consumption of different processor components is measured during each execution phase to (de-)activate supported reliability features for these component to maximize the overall application reliability for a given power constraint.

A. Architectural Support for Fine-Grained Management

To have a fine-grained control between different reliability levels, different redundancy mechanisms i.e., DMR-RE and TMR are implemented for each component at their respective pipeline stages. The voting and comparison is performed at the corresponding stage, while in case of DMR-RE, stalls are introduced if re-execution is required. Our concepts of fine-grained reliability are orthogonal to the available reliability mechanisms.

B. Phase Identification

Different applications have distinct performance characteristics during run time, see Fig. 1. This constitutes phases which are the basis of fine-granularity in our reliability manager.

IV. Our Proposed Reliability Manager

This run time manager utilizes different redundant modes for different components in the processor in an adaptive and fine-grained way such that soft error resilience is maximized under a power constraint. This is achieved by activating a particular reliability mechanism dynamically during a phase of an application considering its diverse power and vulnerability properties (see overview in Fig. 4).

A. Problem Formulation

Using the above mentioned models and notations in Section II, we formulate the problem of selecting one of the m reliability feature for various n component as the 0/1 ILP problem. The processor vulnerability, power and reliability feature is represented by the matrix $V \epsilon \mathcal{V}^{nxm}$, $P \epsilon \mathcal{P}^{nxm}$ and $X \epsilon \{0,1\}^{nxm}$ respectively in which each element V_{ij}, P_{ij} refers to the vulnerability and power of the component i when the reliability feature j is selected for that component. For a features assigned j to a component i, the assignment would be $X_{ij} = 1$. The goal of our run-time reliability manager is to adapt the available reliability modes of various components to optimally minimize overall vulnerability in each phase of the application under a given power constraint.

Optimization Goal: Minimize system vulnerability defined by

$$\text{minimize} \quad \prod_{i,j} X_{ij} V_{ij} \tag{2}$$

since the selection is a 0-1 assignment, we have:

$$\forall i,j : X_{ij} \epsilon \{0,1\} \tag{3}$$

Power Constraint: The power consumption of the processor should be less than the core TDP constraint.

$$X_{ij} P_{ij} \leq P_{TDP} \tag{4}$$

B. Algorithm

Due to its run-time nature, this algorithm employs a lightweight greedy heuristics to select an appropriate available reliability mechanism for different architectural components during a execution phase of an application such that the power and timing constraints are met, and the overall system vulnerability is minimized. Symbolically, the goal is to find: $R_{ij}, \forall i \epsilon [1,..n], j \epsilon [1,..k]$ Such that the total vulnerability of the system is minimized and still the power constraint is also fulfilled. That is:

$$min \sum_{i=1}^{n} v_i(t) \qquad s.t. \quad \sum_{i=1}^{n} P_i(t) \leq \mathbf{P} \tag{5}$$

and also meeting the application deadline time for each phase: $e_i \leq deadline_i, \forall i \epsilon [1,..m]$.

V. Result and Evaluation

A. Experimental Setup

We evaluated our proposed technique for various MiBench applications. Each application is executed on a cycle-accurate simulator GemV [12] to get performance, execution-time and vulnerability statistics. The performance statistics are then processed and were input to McPAT to get power statistics. GemV is also extended for DMR-RE and TMR redundancies of different architectural components which can be controlled during each phase to achieve fine-grained power-efficient reliability management. The vulnerability and power statistics are dumped after every 1000µs and the first 500µs are wasted to let the cache fill first. The reliability is calculated using Eq.1, where fault error rate is taken as $\lambda = 10^{-6}$ [14].

Algorithm 1: POWER AWARE FINE GRAINED RELIABILITY MANAGER

Input: n, m, k, **P**, P_{ij}, R_{ij} and V_{ij} matrix
Output: S:(selected modes for n components)

$S_{(1..n)} \leftarrow Highest\ Reliable\ Mode$
$RM_{(1..n)(1..m)} \leftarrow R_{(1..n)(1..m)}/P_{(1..n)(1..m)}$
$Cost \leftarrow Power_{(1..n)(1..m)}$
for *i=1 to n* **do**
  while $cost > budget$ **do**
    $\delta RM_{(1..n)} \leftarrow \delta R_{(1..n)S(n)}/\delta P_{(1..n)S(n)}$
    $min \leftarrow minimum(\delta RM_{(1..n)})$
    if $length(min) > 1$ **then**
      $index \leftarrow maximum(min)$
    else
      $index \leftarrow index(min)$
    end
    if $S(index) > 1$ **then**
      $S(index) \leftarrow S(index) - 1$
    else
      $S(index) \leftarrow S(index)$
    end
    $Cost \leftarrow Power_{(1..n)(1..m)}$
  end
  $S.update()$
end
return S

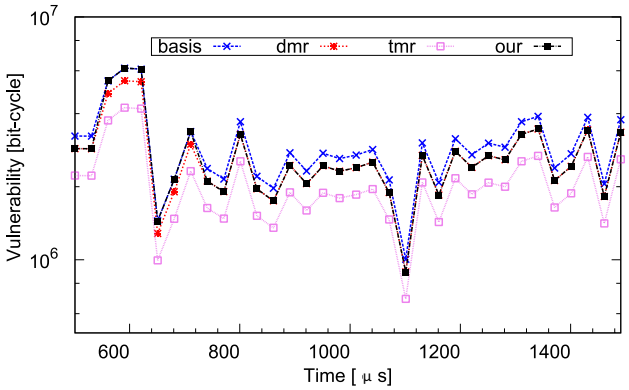

Figure 5: Run-time vulnerability comparison for FFT and TYPESET.

B. Comparison with the State-of-the-Art

We compared our proposed system with two state-of-the-art techniques: (1) dynamic DMR that activates/deactivates DMR mode for different applications according to their vulnerabilities [2], (2) full TMR where applications are fully executed in TMR mode [9]. Our technique decreases the power consumption up to 16% as compared to the highest reliability mode i.e., TMR, as shown in Fig. 6(a). For FFT and SUSAN, our techniques save 15 and 16% respectively while TMR violates the power constraint. For FFT and SUSAN, our technique decreases reliability as compared to TMR by 7% and 2% respectively. The worst degradation observed is for TYPESET which is 11%. The run-time fine-grained vulnerability adaptation for FFT and TYPESET application is shown in Fig. 5, which shows the advantages of our power-aware fine-grained reliability management over the state-of-the-art to ensure minimum vulnerability under power constraints.

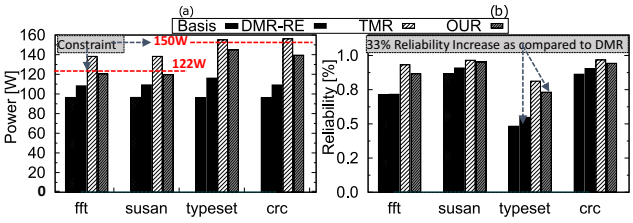

Figure 6: Comparison with state-of-the-art technique [2], [9].

VI. CONCLUSION

This paper presents a fine-grained and power-efficient reliability management system that leverages vulnerability and power variations during an application execution. We achieved this power-efficient reliability through a fine-grained DMR-RE and TMR implemented for different architectural components that can be controlled at each application phases based on vulnerability and power characteristics during this phase. Compared to the two state-of-the-art techniques, our proposed technique achieves significant reliability improvements up to 33% while reducing power up to 13%. This suggests that considerations of both vulnerability and power variations during different phases of the application provide opportunities for fine-grained and power-efficient reliability management.

ACKNOWLEDGEMENT

This work is supported in parts by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500 - spp1500.itec.kit.edu).

REFERENCES

[1] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, N. Wehn, "Reliable On-Chip Systems in the Nano-Era: Lessons Learnt and Future Trends," IEEE Design Automation Conference (DAC), 2013.

[2] R. Vadlamani et al.,"Multicore soft error rate stabilization using adaptive dual modular redundancy", DATE, pp. 27-32, 2010.

[3] J. Henkel et. al., "Design and architectures for dependable embedded systems," Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, 2011.

[4] D. Ernst et al., "Razor: circuit-level correction of timing errors for low-power operation," IEEE MICRO, vol. 24, no. 3, pp. 10-20, 2004.

[5] G. A. Reis et al., "SWIFT: Software Implemented Fault Tolerance", IEEE CGO, pp. 243-254, 2005.

[6] N. Oh et al., "Error detection by duplicated instructions in super-scalar processors", IEEE Transaction on Reliability, vol.51, no. 1, 2002.

[7] J. Hu et al., "In-Register Duplication: Exploiting Narrow-Width Value for Improving Register File Reliability," DSN, pp. 281-290, 2006.

[8] J. S. Hu et al., "Compiler-Directed Instruction Duplication for Soft Error Detection," DATE, vol.2, pp. 1056-1057, 2005

[9] S. S. Mukherjee, M. Kontz, S. K. Reinhardt. Detailed design and evaluation of redundant multithreading alternatives, In IEEE Int'l Symp. Comput. Arch. (ISCA), pp. 99110, 2002.

[10] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in HPCA. IEEE, 2005.

[11] R. Jeyapaul and A. Shrivastava, "Smart cache cleaning: Energy efficient vulnerability reduction in embedded processors," in CASES, 2011.

[12] Srinivas Karthik Tanikella et al. GemV: A Validated Toolset for the Early Exploration of System Reliability. ASAP, 2016.

[13] N. Binkert et al., "The gem5 simulator," ACM SIGARCH Computer Architecture News, vol. 39, no. 2, 2011.

[14] Huang et.al., "Low-Energy Standby Sparing for Hard Real-Time Systems," IEEE TCAD 2012.

[15] I. Koren, C.M. Krishna. 2007. Fault Tolerant Systems.