

Preface

Embedded systems have become an important part of our day-to-day life because of their pervasive deployment in various application domains such as consumer electronic devices like smartphones and tablets, medical healthcare, telecommunication, automotive, aircrafts and space-applications, etc. Due to the shrinking transistor dimensions, embedded computing hardware is getting increasingly susceptible to different reliability threats like *transient faults* (such as soft errors due to high-energy particle strikes) and *permanent faults* due to design-time process variations and run-time aging effects. Therefore, reliability has emerged as one of the primary design criteria in the nano-era. Soft errors manifest as spurious bit flips in the underlying hardware that may jeopardize the correct software execution. However, design-time manufacturing process variability manifests as frequency and leakage power variations in different cores in a multi-/manycore processor, while run-time aging effects result in frequency degradation over the period of time. A majority of state-of-the-art hardware-level reliability techniques employ full-scale redundancy or error correction blocks in processor components that result in significant overhead in terms of area, performance, and/or power/energy, which may be prohibitive within the stringent design constraints of embedded systems. To alleviate the overhead of hardware-level techniques or targeting the low-cost unreliable hardware, several software-level reliability improving techniques have evolved that are based on the concept of full-scale redundancy at the code or data level and, therefore, incur significant performance and memory overhead ($\geq 2\times$ – $3\times$). In general, state-of-the-art software-level reliability techniques have, by far, not exploited their potential since the common belief, so far, was that reliability problems when occurring at the hardware level should be addressed at the hardware level. However, a lot of hardware-level faults can potentially be masked at the higher software layers. Furthermore, a manycore processor is subjected to multiple reliability threats that need to be considered when providing functional and timing correctness.

To enable a highly reliable software system for embedded computing, this manuscript introduces novel concepts, strategies, and implementations to leverage

multiple system layers in an integrated fashion for reliability optimization under user-provided tolerable performance overhead constraints. To enable this, this work addresses the key challenge of *bridging the gap between the hardware and software* by quantifying the effects of hardware-level faults at the software level while accounting for the knowledge of the processor architecture and layout. It is important to understand which instructions lead to which type of errors in the application software program when faults happen in the underlying hardware and how these faults are masked/propagated to higher system layers. In particular, this work develops novel techniques for cross-layer software program reliability modeling and optimization at different levels of granularity (e.g., instruction and function) and at different system design abstractions in order to compose and execute application software programs in a reliable fashion. Important highlights of the novel contributions of this manuscript are:

Cross-Layer Software Program Reliability Modeling and Estimation: This work develops cross-layer reliability analysis, modeling, and estimation concepts, techniques, and tools. An extensive program reliability analysis is performed to understand the manifestation of hardware-level faults at the software level. This analysis is leveraged to devise software-level reliability models that account for the hardware-level knowledge in order to bridge the gap between the hardware and software for accurate reliability estimation at the software level. At the instruction granularity, the following models are developed to capture key reliability aspects of a software program:

1. The *Instruction Vulnerability Index* estimates the probability of an instruction's output being erroneous due to soft errors. It accounts for *spatial vulnerabilities* (i.e., area-wise error probabilities) and *temporal vulnerabilities* (i.e., time-wise error probabilities) of different instructions executing in different pipeline stages of a given processor while accounting for hardware-level information, e.g., the probability of faults in different processor components obtained through a detailed gate-level soft error analysis.
2. The *Instruction Error Masking Index* estimates the probability that an error at an instruction will ultimately be masked until the final program output, i.e., does not become visible at the application output and therefore is denoted as "masked."
3. In case the error is not masked, the *Error Propagation Index* estimates how many outputs will be affected by the unmasked error.

These instruction-level estimates are then used to obtain the reliability estimates at basic block and function/task levels. In the optimization flow, these models are leveraged to quantify the reliability-wise importance of different instructions, basic blocks, and functions to enable selective reliability optimization at different system layers under tolerable performance overhead constraints.

Cross-Layer Software Program Reliability Optimization: This manuscript develops concepts and techniques for cross-layer reliability optimization and leverages multiple system layers for reliable composition and execution of application software programs. First multiple versions of a software program are obtained that

enable run-time trade-offs between reliability and performance properties. This is done through the following two means:

1. *Different reliability-driven software transformations and instruction scheduling techniques* are proposed that lower spatial/temporal vulnerabilities and probabilities of software program failures and Incorrect Outputs by reducing the number of executions of *critical instructions* (like load, store, branches, jumps, and calls). Applying these transformations in constrained scenarios provides on average 60 % lower software program failures (i.e., crashes, halt, hang, abort) and thus increased software reliability.
2. *Reliability-driven selective instruction redundancy* is proposed that selects a set of reliability-wise important instructions in different functions for redundancy-based protection depending upon the instruction vulnerabilities, instruction-level error masking and propagation, and protection overhead under user-provided tolerable performance overhead constraint. The key is to give more protection to the less-resilient part of the software program and less protection to more-resilient part to achieve a high degree of reliability in constrained scenarios. Compared to state of the art, the proposed selective instruction protection provides 4.84x improved reliability at 50 % tolerable performance overhead constraint.

Afterwards, multiple reliable versions are exploited by a reliability-driven run-time system that enhances the reliability of multiple concurrently executing applications in a manycore processor while accounting for the frequency variations and degradation due to design-time process variation and run-time aging-induced effects. It performs the following key operations to facilitate reliable software program execution:

1. *Adaptively activating and deactivating the redundant multithreading for different applications* in a manycore processor in area-constrained scenarios. It accounts for variable resilience properties and deadline requirements of different applications along with a history of the encountered errors.
2. *Dynamically selecting an appropriate reliable version* for each application considering cores' frequency variations due to design-time process variations and run-time aging-induced performance degradation.
3. Mapping the selected application version on the cores used for redundant multithreading at run time such that the execution properties of the redundant threads closely match the frequency properties of allocated cores considering core-to-core frequency variations.

Compared to state-of-the-art single-layer reliability optimizing techniques, the proposed cross-layer approach achieves 16 %–57 % improved software reliability on average for different chip configurations, various process variation maps, and different aging years.

In addition to the above-discussed scientific contribution, several tools for gate-level soft error analysis, aging analysis, an integrated fault generation, and an injection system for instruction set simulators have been developed in the scope of this work and are made available at <http://ces.itec.kit.edu/846.php>.

<http://www.springer.com/978-3-319-25770-9>

Reliable Software for Unreliable Hardware

A Cross Layer Perspective

Rehman, S.; Shafique, M.; Henkel, J.

2016, XXVIII, 237 p. 121 illus., 83 illus. in color.,

Hardcover

ISBN: 978-3-319-25770-9