Reliability has emerged as a first order concern for system designers and applications developers. The ever decreasing transistor sizes and lower threshold voltages have made modern processors more vulnerable to various types of errors, more prominently towards soft errors due to particle strikes [1]. This fact has given rise to a rich body of research that tries to increase a system’s reliability through hardware [11, 9] and software [8, 7] techniques. The main drawback of these approaches is that they target reliability from a solely hardware or software point of view.

In my opinion, reliability mechanisms should be designed and implemented following a holistic approach where hardware/software interactions will allow a tailored and more efficient reliability solution at low hardware costs and performance overheads. The notion of abstraction, due to which the hardware level has no knowledge of the running application, its semantics and requirements, is hurting the efficiency of reliability policies because it is robbing the hardware from necessary information that can be exploited in the implementation of efficient and low-cost error protection mechanisms.

It has been shown that each application has different reliability properties and requirements. For example, it has been shown that an reliability levels vary across different applications and across the execution time of the same application [10, 6]. Applied compiler optimizations play an important role in this variable behavior, as they modify the executing code and change the application’s behavior and performance. Thus, we cannot ignore reliability as a constraint when developing compiler techniques or software applications.

In [3], we study the relationship between compiler optimizations and application reliability. We tie the compiler’s effects with the resulting effects on the application’s micro-architectural behavior and as a result to the processor’s structures occupancies and their reliability levels. It is observed that compiler optimizations that improve performance, lead to more reliable applications as the average expected number of failures during execution is lower. At the same time though, optimizations lead to more prominent variations in structures occupancies and as a result to less predictable behavior and more prominent variations in the applications reliability. Our methodology can be adopted by hardware designers and compiler developers to make it possible to analyze the relationship between the reliability hardware features and compiler optimization. Such an awareness can lead to more reliable hardware or more reliable compiler-optimized code, in addition to the ability to dynamically adjust code reliability for a fixed architecture and compiler. Additionally, we can ensure that the system operates within its Silent Data Corruption (SDC) and Detectable and Unrecoverable Errors (DUE) limits without heavy overheads on performance or hardware complexity.

Moreover, reliability towards intermittent errors due to wearout phenomena is directly affected by the stress imposed on the processor and the circuits utilization. An important fact to systems architects and designers is the fact that if an intermittent error is identified in the early stages of its manifestation, then it is possible to prevent permanent failures if the stress on the transistors is relieved. Hence, addressing reliability concerns due to intermittent errors can result in a longer overall processor lifetime.
For computer architects and compiler developers the most important abstraction layer in the chip is the instruction set architecture (ISA). Benchmarking the susceptibility of ISA to intermittent failures is the focus of this paper. The knowledge of the effect that ISA has on the reliability of a system against intermittent errors can be a valuable tool for system design and implementation. In [4], we use the SPARC V9 Instruction Set Architecture (ISA) to demonstrate our methodology for benchmarking ISA vulnerability to intermittent failures. We use the notion of On-Chip Footprint to measure the amount of chip area an instruction activates as it traverses the pipeline. We call this chip footprint the Vulnerability to Intermittent Faults (VIF). As has been shown in many prior studies [2, 5], the susceptibility of a logic to wearout depends on how well the logic is utilized during program execution. Hence, we use VIF to measure the amount of wearout each instruction in the ISA can cause by its execution. We translate the VIF measure to Time-To-Failure metric to quantify how the chip area utilization translates into wearout related failures.

Thanks to the cross-layer experimental infrastructure and the metrics proposed in [3, 4], we can estimate the relative reliability of different applications against transient errors and the relative reliability of any instruction within an ISA against intermittent errors.

In the future, we plan to utilize these metrics to propose a reliability-aware system that is steering its reliability decisions taking into consideration, not only the micro-architectural events that occur during its operation, but also the type of applications that are being executed, their semantics and reliability and performance requirements. The end-goal of my research is the design and implementation of a reliability-aware system that remains resilient to failures due to transient or intermittent errors while maintaining its performance and/or hardware cost budget.

References