Effective Fault Tolerance for Robust Robotics under Radiation Exposure

Jörg Müller Andreas Riefert Wolfram Burgard Gaurav S. Sukhatme Bernd Becker

I. INTRODUCTION

The development of fault-tolerant autonomous robots for long-term deployment has been an area of active research for decades. Many researchers have focused on the tolerance to failures of sensors and actuators of various types of robots. In contrast to these failures, robots encounter transient faults on all levels including the bit level of microprocessors when operating in space or in nuclear facilities under high radiation. Additionally the shrinking nanometer technology of modern microprocessors and the supply voltage down-scaling drastically increase the risk of transient faults. Consequently even in nonhazardous environments modern processors encounter faults at an interval of days or hours.

To cope with such soft errors, the VLSI community has developed numerous hardware hardening and low-level software protection schemes. Entire protection, however, makes processors usually slower, more expensive, and more energy-consuming. These standard techniques or an additional lead shielding are usually not desirable for autonomous robots.

In this paper, we present novel techniques that bridge the gap between the low-level protection of microprocessors and high-level tolerance to sensor and actuator failures in robotics. Our methods enable a detailed analysis of applications under fault injection to identify critical parts of the combined hardware and software system. In our experiments, we demonstrate how our approach can be used to develop an efficient combination of selective hardening of processors and lightweight protection schemes of robotics software.

The detection and isolation of actuator and sensor failures have been widely studied in robotics. There exist model-based approaches that consider fault detection as a classification or regression problem and apply corresponding techniques [3, 4]. Furthermore the Bayesian approach has been popular to reason about discrete fault states during state estimation [8]. These techniques, however, apply severe model approximations or restrict the detection to faults that are specified in advance. Similar to our approach, Christensen et al. [2] apply fault injection during operation. As opposed to our method, they only inject numeric faults into the actuator and sensor data to generate data to train a classifier for fault detection.

In contrast to the above mentioned approaches, Nagatani et al. [5] examined the influence of radiation on robotic devices. In their preliminary tests, most devices appeared to be robust against persistent hardware faults. However, they did not examine the effect of transient faults since their robots were tele-operated and could be rebooted on suspicious behavior.

To cope with transient faults, there exist various hardware [1] and software protection schemes [6]. They basically add redundancy to the circuit or the program in order to detect and correct transient faults. Modern microprocessors, however, implicitly mask a lot of the transient faults [9] and often plenty of the available registers of a processor are only rarely used by the compiler. Additionally, certain software applications, such as the state estimation techniques usually applied in robotic systems, already partially imply fault tolerance. Consequently, our approach focuses on determining the actual impact of transient faults on a running application to design selective and lightweight protection schemes.

II. TRANSIENT FAULT SUSCEPTIBILITY ANALYSIS

In the first stage, we automatically identify critical parts of a combined hardware software system through a fault injection approach [7]. Our generic FPGA-based fault injection platform enables a designer to test arbitrary applications on a microprocessor running at high speed under fault injection in its native environment. It provides the opportunity to evaluate designs and provides insight into the soft error characteristics of the application. Usually silent data corruptions, which become visible as calculation errors, are hard to detect. Therefore we focus on examining the susceptibility of applications with respect to serious calculation errors rather than system flow errors and crashes. Our approach is able to automatically identify the most critical registers of a given microprocessor and the most critical variables of an arbitrary application. In particular, our framework tracks a fault from its injection into a register up to the source code that is currently executed. This enables us to statistically evaluate the criticality value of each variable in terms of the number of runs in which the variable was hit by a fault and a serious calculation error occurred.

III. EFFICIENT APPROACHES TO FAULT TOLERANCE

The statistical evaluation of the criticality of the registers of the processor and the variables of the software application provides useful information for protecting the system. We propose a protection approach on two levels. First, we apply hardware protection to the most critical registers such as the one that keeps the program counter of the processor. Second, given the criticality value of the software variables, we
apply suitable protection schemes to the software application. This yields a protection scheme that is efficient in terms of computation, energy, and cost overhead and matches the required level of protection.

In this paper, we present two efficient schemes for software protection. Our generic approach is a selective duplication scheme for error correction of critical variables and is applicable to arbitrary applications [7]. We duplicate each considered variable and its computation and add an equality check of the results of both computations. Upon detection of an inequality, the corresponding computations are repeated. By protecting only the $k$ most critical variables, we can trade the protection level off against the performance overhead.

In many robotics applications, there are certain parts or structures that are inherently robust to transient faults. For example, probabilistic state estimation techniques such as the Kalman filter are robust to noise in the input data. For such applications, often more sophisticated approaches than protecting individual variables are beneficial for fault tolerance. In particular, an application engineer could use the criticality values of the variables together with his expert knowledge to create an efficient application-specific software protection scheme.

IV. EXPERIMENTS

To evaluate our methods, we implemented and experimentally evaluated a 32-bit MIPS processor on an Altera Cyclone II FPGA starter board. As applications under test, we consider two popular approaches to probabilistic state estimation for mobile robots (see Fig. 1). Our first application is the three-dimensional orientation estimation of a blimp using the measurement data of an inertial measurement unit. In our second application, we estimate the distance of a Pioneer 3-DX wheeled robot to its base station. We fuse the measurement data of the wheel encoders with the distance measurements of a miniature sonar sensor in a particle filter (PF).

We identified the critical variables of the applications by first executing several thousand runs under fault injection. The criticality values of the variables result from the analysis of the application output. Using our selective duplication scheme on the three most critical variables of the EKF application had a runtime overhead of 18%. With that, we significantly reduced the number of critical calculation errors during 7,000 runs from 9 to 1.

In the PF application, we demonstrated how to develop an efficient protection scheme with expert knowledge. The criticality analysis (Fig. 2, left) indicates that especially calculating and summing up the weights of the particles are critical parts of the PF. Our application-specific protection scheme evenly divides the particle set into two groups, processes the computation for each group individually, and applies a plausibility check by comparing the results of both groups. If the results suspiciously differ, we assume that the computation of one of the groups was hit by a soft error and repeat the computation for the whole set of particles. This protection causes a computation overhead of only 3.7%. The experimental results of our highly effective protection scheme (Fig. 2, right) demonstrate a significantly reduced number of critical calculation errors.

REFERENCES


