Efficient Software Checking for Fault Tolerance *

Jing Yu, María Jesús Garzarán and Marc Snir
Department of Computer Science
University of Illinois at Urbana-Champaign
201 N.Goodwin ave, Urbana, IL 61820
\{jingyu, garzaran, snir\}@cs.uiuc.edu

Abstract

Dramatic increases in the number of transistors that can be integrated on a chip make processors more susceptible to radiation-induced transient errors. For commodity chips which are cost- and energy-constrained, software approaches can play a major role for fault detection because they can be tailored to fit different requirements of reliability and performance. However, software approaches add a significant performance overhead because they replicate the instructions and add checking instructions to compare the results.

In order to make software checking approaches more attractive, we use compiler techniques to identify the “unnecessary” replicas and checking instructions. In this paper, we present three techniques. The first technique uses boolean logic to identify code patterns that correspond to outcome tolerant branches. The second technique identifies address checks before loads and stores that can be removed with different degrees of fault coverage. The third technique identifies the checking instructions and shadow registers that are unnecessary when the register file is protected in hardware. By combining the three techniques, the overheads of software approaches can be reduced by an average 50%.

1. Introduction

Dramatic increases in the number of transistors that can be integrated on a chip will deliver great performance gains. However, it will also expose a major roadblock, namely the poor reliability of the hardware. Indeed, in the near-future environment of low power, low voltage, relatively high frequency, and very small feature size, processors will be more susceptible to transient errors. Transient faults, also known as soft errors are due to impacts from high-energy particles or other random external events that change the logic values of latches or logic structures. Error detection schemes are needed to ensure that a soft error does not go undetected and result in an erroneous computation. Once errors are detected reliably, it is often possible to use software schemes for error correction – the performance of error correction schemes is not critical, as long as errors are not too frequent; however, error detection adds an overhead to all computations and has to perform efficiently. For this reason, we focus in our project on error detection.

Hardware error detection is used on modern microprocessors to detect errors in storage and buses: for example, ECC memory and parity bits for caches and various buses. Such error correcting codes generally add a low overhead to performance and chip size. On the other hand, it is much harder to detect errors in the computation pipeline: one needs to replicate significant fractions of the CPU logic, in order to do so. Such a replication is used in high-end fault-tolerant systems, such as IBM mainframes, HP NonStop or mission-critical computers. It is unclear whether the cost of essentially doubling hardware complexity is acceptable for commodity systems. For such systems, software error detection may be a preferable solution. The main advantage of software checking is its flexibility: different trade-offs between performance and reliability can be achieved on the same hardware, using different software approaches; fault tolerant hardware cannot offer the same flexibility. Such flexibility can be used, for example, for achieving a higher level of reliability for large clusters, built out of commodity components: A PC might be built to have a mean time between undetected failures (MTTUF) of, say, 10 years; this would result in an unacceptable MTTUF of half a week for a 1000 node PC cluster. Alternatively, the flexibility may be used to achieve different levels of reliability for different software components: One may not care about undetected errors that will affect the PC display during a game, but may want to avoid errors that will corrupt the file system metadata.

Current software approaches address the problem by replicating the instructions and adding checking instruc-

\*This material is based upon work supported by the National Science Foundation under the CSR-AES program Award No. 0615273