An FPGA-Based Experimental Evaluation of Microprocessor Core Error Detection with Argus-2

Patrick J. Eibl
IBM
3039 E. Cornwallis Rd., Box 12195
Building 062/J211, RTP, NC 27709
patrickeibl@gmail.com

Albert Meixner
NVIDIA
2701 San Tomas Expy
Santa Clara, CA 95050
albert.meixner@gmail.com

Daniel J. Sorin
Duke University
PO Box 90291
Durham, NC 27708
sorin@ee.duke.edu

ABSTRACT

Recently, several researchers have proposed schemes for low-cost, low-power error detection in the processor core. In this work, we demonstrate that one particular scheme, an enhanced implementation of the Argus framework called Argus-2, is a viable option for industry adoption. Using an FPGA prototype, we experimentally evaluate Argus-2's ability to detect errors due to (a) all possible single stuck-at faults in a given core and (b) a statistically significant number of double stuck-at faults, including pairs of faults that are randomly located and pairs that are spatially correlated on the chip.

Categories

C.4 [Performance of Systems] Fault tolerance

General Terms

Measurement, Reliability

Keywords

computer architecture, error detection, dynamic verification

1. INTRODUCTION

Fault tolerance is becoming increasingly desirable as CMOS technology progresses to ever-smaller feature sizes and increasing susceptibility to errors [2]. Recently, several researchers have proposed schemes to address the need for low-cost, low-power error detection in the processor core. Some notable schemes include redundant multithreading [6], DIVA [1], and Argus [4]. Of these schemes, Argus appears to be the most power-efficient, especially for detecting errors in the simple, low-power cores that are expected to dominate many-core processors. However, we are unaware of any company moving to adopt any of these schemes, despite their promise, and one factor in this reluctance is a lack of conclusive experimental or analytical evidence that these low-cost schemes provide sufficient error coverage. The goal of our work is to conclusively demonstrate that one particular low-cost error detection scheme, an enhanced implementation of the Argus framework called Argus-2, is a viable option for industry adoption.

2. ARGUS

We first explain the Argus framework for detecting errors and then describe the one particular implementation of the Argus framework that we evaluate in this paper.

SIGMETRICS'11, June 7–11, 2011, San Jose, California, USA.
ACM 978-1-4503-0262-3/11/06.

2.1 High-Level Overview of Argus

Argus is a framework for detecting errors within a processor core by checking invariants at run-time. By checking invariants, instead of checking components, implementations of Argus can be significantly less costly than DMR. The key insight is that, at a high level, a Von Neumann core performs only four activities: choosing the sequence of instructions to execute ("control flow"), performing the computation specified by each instruction ("computation"), passing the result of each instruction to its data-dependent instructions ("dataflow"), and interacting with memory ("memory"). The Argus framework consists of checking each of these four activities at runtime. An implementation of Argus consists of four checkers—for control flow, dataflow, computation, and memory—and it has been proven that a perfect implementation of Argus can detect almost all possible processor errors. Implementations are likely to be imperfect, however, due to cost/reliability tradeoffs. For example, a checker that uses lossy signatures or checksums is imperfect, yet may be an appropriate design decision due to its low cost.

Control Flow Checking. Control flow checkers [8] periodically compare the static control flow graph of the program binary to the dynamic control flow graph of the runtime execution. If the static and dynamic control flow graphs differ, an error has been detected. When used in isolation, a control flow checker detects errors in fetch logic, branch destination computation, and PC update logic.

Dataflow Checking. A dataflow checker [5] compares the static dataflow graph of the program binary to the dataflow graph reconstructed at runtime. If they differ, an error has been detected. When used in isolation, a dataflow checker detects errors in many activities, some of which only apply to superscalar processors; these activities include fetch, decode, register rename, register read and write, and instruction scheduling (reorder buffer, load-store queue, reservation stations, etc.).

Computation Checking. There is a long history of research in low-cost checkers for the functional units that perform the computations in processor cores. The key to these checkers is that it is fundamentally easier to check a computation than to perform it in the first place. Sellers et al.'s book [7] on error detecting logic provides an excellent survey of checkers for adders, multipliers, dividers, bit-wise logic units, etc.

Memory Checking. The main error hazard in the memory is due to data corruption. Error detecting codes (EDC) and error correcting codes (ECC) are well-known, low-cost solutions for protecting the integrity of caches and DRAM.

2.2 Argus-2 Implementation

Argus is a framework, and there are many implementations that satisfy this framework. We started with the original Argus-1 implementation [4] and we created the Argus-2 implementation by enhancing Argus-1 to reduce its implementation costs and to plug numerous error detection holes that were revealed during preliminary experiments.

Both Argus-2 and Argus-1 are based on the same baseline core, the OpenRISC 1200 core [3]. The OR1200 processor core is a 32-bit scalar (1-wide), in-order RISC core with a 4-stage pipeline. This core represents the low-end of the simple cores that are expected to be used, perhaps in conjunction with a small number of superscalar cores, in multicore chips. The Argus approach to error detection applies to any Von Neumann core, not just the OR1200, but we chose the OR1200 because it is representative of the simple, power-efficient cores that are attractive for embedded applications and many-core architectures.

3. EXPERIMENTAL METHODOLOGY

The primary purposes of the experimental evaluation are to determine Argus-2's ability to detect errors and its costs.

3.1 Experimental Testbed

We performed all of our experiments on Altera DE2 prototyping boards that contain a Cyclone II FPGA. The primary advantage of using the prototyping boards, rather than simulation, is speed.

3.2 Fault Model

Our fault model consists of single and double stuck-at faults, and these faults can occur at the output of any gate in the circuit, *including Argus-2 logic itself*. For single faults, we exhaustively explore every possibility. For the double stuck-at faults, we sample from the enormous space of all possible double faults so as to achieve 95% confidence that our results are within a small percentage of the exhaustive (un-sampled) results. We consider both random fault locations and spatially correlated locations (i.e., the two faults are spatially near each other).

3.3 Workload

Each experiment involves injecting of one or two faults from the fault model described in Section 3.2 and then observing the behavior of the system while the processor runs a given software workload. The workload that we have chosen is the decoding of jpeg images. The benchmark runs for a significant amount of time (2.4 seconds) on real hardware, which is enough time to decode two images and execute approximately 60 million instructions.

4. EXPERIMENTAL RESULTS

We evaluated error detection coverage, area, and performance.

Error Detection Coverage. Of all possible single stuck-at faults, only 0.013% of them lead to errors that are both unmasked and undetected by Argus-2 (i.e., silent data corruptions or SDCs). Furthermore, for our large sample of double stuck-at faults, fewer than 0.023% of them lead to SDCs. Among the double stuck-at faults, the spatially correlated fault pairs were slightly less likely to lead to SDCs. These small fractions of SDCs show that Argus-2 is a viable option for industry. Figure 1 shows results for single and double stuck-at-one faults. The large fraction of faults that are masked (i.e., have no impact on application behavior) are primarily due to faults in components that are either never or rarely used (e.g., multiply-accumulate unit) and faults in Argus-2's hardware.

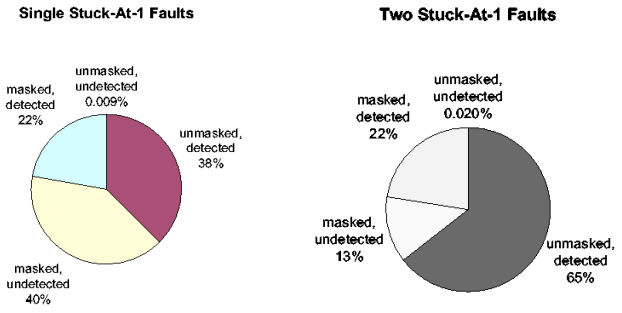

Figure 1. Error detection results

Area Overhead. The hardware for Argus-2 consumes some amount of chip area. Our results show that Argus-2's core overhead (not including the caches) is 12.2%, and its total chip area overhead (with the caches) is 3.7%. These results confirm that Argus-2's overhead is far less than DMR.

Performance. Argus's performance impact is due to the NOP instructions it adds into the binary for purposes of checking control flow and dataflow. Argus-2 adds, on average, 7% to the static instruction count and 3.5% to the dynamic instruction count. The increase in dynamic instruction count has a performance overhead that averages less than 4%.

5. CONCLUSIONS

We conclude that Argus-2 is ready to be adopted by industry. An extremely small fraction of single and double faults cause silent data corruptions, which we believe is sufficient for the vast majority of commodity processors. Argus-2 is the first approach to provide complete coverage of permanent and transient faults at low cost, even for simple cores.

6. ACKNOWLEDGMENTS

This material is based on work supported by the National Science Foundation under grant CCR-044516.

REFERENCES

[1] T. M. Austin. DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design. In *Proc. 32nd Annual IEEE/ACM Int'l Symposium on Microarchitecture*, Nov. 1999.

[2] International Technology Roadmap for Semiconductors, 2007.

[3] D. Lampret. OpenRISC 1200 IP Core Specification, Rev. 0.7. http://www.opencores.org, Sept. 2001.

[4] A. Meixner, M. E. Bauer, and D. J. Sorin. Argus: Low-Cost, Comprehensive Error Detection in Simple Cores. In *Proc. of the 40th Annual Int'l Symp. on Microarchitecture*, Dec. 2007.

[5] A. Meixner and D. J. Sorin. Error Detection Using Dynamic Dataflow Verification. In *Proc. of the Int'l Conf. on Parallel Architectures and Compilation Techniques*, Sept. 2007.

[6] E. Rotenberg. AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors. In *Proc. of the 29th Int'l Symposium on Fault-Tolerant Computing Systems*, June 1999.

[7] F. F. Sellers, M.-Y. Hsiao, and L. W. Bearnson. *Error Detecting Logic for Digital Computers*. McGraw Hill Book Company, 1968.

[8] N. J. Warter and W.-M. W. Hwu. A Software Based Approach to Achieving Optimal Performance for Signature Control Flow Checking. In *Proc. of the 20th Int'l Symposium on Fault-Tolerant Computing Systems*, June 1990.