Signature-based Online Periodic Fault Tolerance for Microprocessors

Waleed Dweik and Murali Annavaram
Electrical Engineering Department, University of Southern California
Los Angeles, CA, USA
{dweik,annavara}@usc.edu

I. INTRODUCTION

Due to ever increasing process variations and wear-out processors cannot rely solely on post-fabrication testing. Online fault tolerance mechanisms that can detect, correct, and recover from different kinds of faults become a critical component for processors to meet their mean time to failure (MTTF) requirements. Online fault tolerance mechanisms can be divided into two main categories: Continuous and periodic.

Continuous online fault tolerance mechanisms rely on redundant execution (in time or space) to validate the execution of every instruction such as dual modular redundancy (DMR) and DIVA [1]. Although continuous fault tolerance mechanisms provide very high fault coverage, they cause huge area (100% for DMR), performance (in case of redundant execution in time), and power overheads. Such huge overheads render the continuous fault tolerance mechanisms inefficient, particularly for low-cost processors which have small area/performance/power budgets to handle faults.

Periodic online fault tolerance mechanisms divide the execution time into epochs. At the end of each epoch, the processor is tested to verify its functionality. If the test completes successfully, the processor is assumed non-faulty and a new checkpoint is created to be used as a recovery point in case the test fails at the end of the next epoch. If the test fails to complete, the processer state is recovered by rolling back to the latest correct check-pointed state. Software-based online detection of hardware defects [2] is an example of the latest proposed periodic fault tolerance mechanisms. There are three important factors that determine the efficacy of periodic mechanisms: test period, test fault coverage and test methodology. While an exhaustive test provides high fault coverage, the performance degradation due to periodic testing will be huge. Shorter tests sacrifice fault coverage to reduce performance overhead.

Multiple test methodologies can be used to generate tests for periodic online fault tolerance mechanisms: built-in self-test (BIST), functional testing, and software-based self-test (SBST). BIST uses detailed RTL description and gate-level netlist to generate test patterns with high fault coverage, but requires significant hardware support. Functional testing uses instructions with random operands as tests, but requires a very large number of instructions to achieve high fault coverage. SBST achieves the best tradeoff by generating high fault coverage patterns based on the RTL description and then mapping the patterns into instructions from the ISA. SBST requires no hardware modification and can still provide high fault coverage of up to 97.3% [5]. The performance overhead of SBST still can reach up to 30%. The reason behind this high performance overhead is the use of exhaustive tests to achieve high fault coverage.

In conclusion, the SBST periodic online fault tolerance approach can be made attractive for processors if performance overheads can be significantly reduced. In [4], Gupta et al. proposed an adaptive testing framework (ATF), at the granularity of an entire core in a chip multiprocessor, to reduce the performance overhead of SBST. Low-level sensors are employed to assess core health. If the core is assessed as healthy, low coverage test (i.e. short test period) is used to test the core. Using sensors with area overhead of 2.6% and target fault coverage of 97.3%, ATF causes 6% performance overhead. ATF health assessment is done at core granularity and hence this approach lacks visibility into the health of individual components (or stages) within the core. So, for example it could be the case that component X is the weakest component in a processor, yet the high fault coverage test may not focus on component X extensively because the test is selected based on core health status instead of components health status. This will increase the probability of faults being undetected during the test period.

We propose a signature-based mechanism that tracks the usage of different components within the processor during each epoch, and then generates a single signature of all tracking information using a hashing function. The signature is used to choose a test (group of tests) from a test pool that mainly focuses on components which have been used (stressed) the most during that epoch and hence more vulnerable to different kinds of faults. While still being able to achieve high fault coverage, the signature-based mechanism is expected to cause a minimal performance overhead of 2%. The tracking information is gathered using a grid of flip flops (FF's), called monitoring FF's, distributed to cover most components within the processor. Before deploying the processor in the field, the SBST approach is used to generate a pool of tests. These tests are stored in a database. Each test focuses on one (or more) components within the processor and provides high fault coverage for them with few instructions. The entire test pool must cover all processor components. Each test in the test pool is assigned a signature that reflects the components covered extensively by the test.

The main contribution of this work is the use of monitoring FF's (which have lower area and power overheads than low-level sensors) to track the usage history at the component level within a processor. All component utilization information is merged in the form of a compact signature. The signature is used as an index to choose the smallest number of tests in order to reduce the performance overhead of periodic testing while still providing high fault coverage. The high fault coverage comes from the fact that we choose the tests that focus on the most stressed

components instead of choosing tests based on the health of the entire core.

II. SIGNATURE-BASED ONLINE PERIODIC FAULT TOLERANCE MECHANISM

There are four steps in our signature-based periodic fault tolerance mechanism. The four steps are described below:

A. Test Generation

This step is only done once after the chip passes post-fabrication tests and before it is deployed in the field. We generate the tests using SBST methodology [5]. SBST extracts fundamental intellectual properties (IPs) from RTL description, then uses an automatic pattern generation tool (ATPG) to generate test patterns for each IP based on specific fault models, and finally maps the test patterns into instructions from the ISA. SBST also produces test routines for programmable visible registers, control logic, steering logic, pipeline registers and logic.

Each test contains a sequence of instructions and it covers one (or more) components of the processor, such as a pipeline stage. The goal is to have a group of short tests which when executed together provide high fault coverage for the entire processor. For example, we can have a test group that covers both fetch and decode stages as these stages are stressed together all the time. Another test group that covers the integer arithmetic logic units (ALU) and one that covers floating point units.

Each test is assigned a unique signature that reflects the components (stages) that are covered extensively by that test. To compute the signature we run each test on the processor and use the usage history information gathered from the monitoring FF's to create a signature for that test. After generating signatures for all tests, a signature-test mapping database table is kept in a permanent storage associated with that processor, such as flash memory.

B. Tracking Usage Information and Signature Generation

After the processor is deployed in the field the chip is periodically tested using the online periodic fault tolerance mechanism. In order to be able to choose the most appropriate test collection at the end of each epoch, we need to track the usage history of the processor components during the epoch. We achieve that by monitoring the inputs of individual components using flip flops (FF's). At the beginning of each epoch all FF's are reset to 0, then during the epoch if an input flips (i.e. a transition is detected at the input) we set the monitoring FF to 1. Periodically, during the epoch all monitoring FF's are captured as a single vector and then reset to 0 again. At the end of the epoch, we have a group of vectors where vector_i represents the state of monitoring FF's at time_i. The vectors are merged together to generate a single signature using a hashing function. As mentioned before, some components are used all the time (e.g. fetch and decode logic) and hence do not need to be monitored thereby reducing the hardware cost of the monitoring FF's.

C. Fault Detection and Recovery

Fault detection is done by applying test programs at the end of each epoch and validating their outputs. The usage signature is used to perform an efficient similarity search inside the signature-test mapping table. Tests that focus on components which were stressed the most are used during the test period. If all selected tests complete successfully, the core is assumed to be non-faulty and a new check-point is created before execution resumes. If one or more tests fail, we roll back to the latest check-point and start execution from there. There are a couple of checkpointing mechanisms that have small hardware overhead and can be used in our signature based mechanism.

As a precaution, one could run a comprehensive test using all the stored tests in the database once in a while to make sure no faults escaped the periodic short testing.

D. Fault Correction

Using the proposed fault detection setup, we cannot identify the location of the fault. As a result, when a fault is detected during the test period the failing test is replayed again to tolerate transient faults. If the test passes in the second run, the fault is assumed to be transient and no correction action is required. If the test fails again, we can safely assume that the detected fault is non-transient as the probability of having two transient faults back-to-back is very low. In this case one option would be to declare the processor as faulty. Another option would be to identify the fault location and then use a fault classification mechanism to further classify non-transient faults as intermittent or permanent faults [3]. This way, we may not need to declare the processor as faulty. Instead we isolate the faulty component and resume the execution. Notice that this case is only possible when the faulty component is redundant (e.g. four ALUs are available and only one of them failed). In order to identify the location of the fault, we would need to use the monitoring FF's to capture the internal state of the faulty processor while executing the failing test and compare it to the internal state of an error-free processor running the same test.

ACKNOWLEDMENTS

This work was supported by NSF grants CAREER-0954211, CCF-0834798.

REFERENCES

[1] T. Austin. Diva: a reliable substrate for deep submicron microarchitecture design. In Proc. of the 32nd Annual International Symposium on Microarchitecture, pages 196–207, 1999.

[2] K. Constantinides, O. Mutlu, T. Austin, and V. Bertacco. Softwarebased online detection of hardware defects: Mechanisms, architectural support, and evaluation. In Proc. of the 40th Annual International Symposium on Microarchitecture, pages 97–108, 2008.

[3] W. Dweik, M. Annavaram, and M. Dubois. Reliability Aware Exceptions. USC EE-Computer Engineering Technical Report, CENG 2011-2.

[4] S. Gupta, A. Ansari, Shuguang Feng, and S. Mahlke. "Adaptive online testing for efficient hard fault detection," Computer Design, 2009. ICCD 2009. IEEE International Conference on , vol., no., pp.343-349, 4-7 Oct. 2009.

[5] T.-H. Lu, C.-H. Chen, and K.-J. Lee. A hybrid software-based self-testing methodology for embedded processor. In 2008 ACM symposium on Applied computing, pages 1528–1534, New York, NY, USA, 2008. ACM.