RetroDMR: Troubleshooting Non-Deterministic Faults with Retrospective DMR

Ting Wang, Yannan Liu and Qiang Xu
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong

Zhaobo Zhang, Zhiyuan Wang and Xinli Gu
Huawei Technologies, Inc.
Santa Clara, CA

***Abstract*—The most notorious faults for diagnosis in post-silicon validation are those that manifest themselves in a non-deterministic manner with system-level functional tests, where errors randomly appear from time to time even when applying the same workloads. In this work, we propose a novel diagnostic framework that resorts to dual-modular redundancy (DMR) for troubleshooting non-deterministic faults, namely RetroDMR. To be specific, we log the essential events (e.g., the sequence of thread migration) in the faulty run to record the mapping relationship between threads and their corresponding execution units. Then in the following diagnosis runs, we apply redundant multithreading (RMT) technique to reduce error detection latency, while at the same time we try to follow the thread migration sequence of the original run whenever possible. By doing so, RetroDMR significantly improves the reproduction rate and diagnosis resolution for non-deterministic faults, as demonstrated in our experimental results.**

I. Introduction

Due to the ever-increasing design complexity and the inaccurate abstract models used in integrated circuit (IC) design and verification, post-silicon validation has become an essential step in today's IC design flow to uncover those bugs escaped from pre-silicon verification [1]. During post-silicon validation, engineers test silicon prototypes in actual application environment to validate their correctness across various operating conditions.

Silicon debug and diagnosis are extremely challenging, especially when tests do not fail consistently over multiple runs [2]. In order to reproduce a failure of the original faulty run, both *fault activation* and *fault propagation* conditions need to be satisfied for the following diagnosis runs. However, even when applying the same workloads, the actual execution of two different runs on the system can vary significantly, due to the fact that today's electrical systems are essentially adaptive systems. For example, thread scheduling in a multi-core processor is usually performed dynamically according to information collected from various on-chip sensors. Consequently, it is likely that the failing thread in one run is scheduled and executed in a different core in another run and does not activate the fault. Moreover, faults may be caused by electrical bugs originating from subtle interaction between a design and its "electrical" state, such as signal integrity issues and thermal effects [3]. Electrical bugs manifest themselves only under specific operation conditions (e.g., certain voltage, frequency and temperature corner), and hence they may not be activated even if the logic circuit operates the same way between different runs (but under different electrical conditions). Similarly, even when faults are activated in the diagnosis run, they may get masked [4] or result in a different observable failure due to the discrepancy between different runs [2].

In this paper, we propose a novel retrospective diagnostic framework that resorts to dual-modular redundancy (DMR) for troubleshooting non-deterministic faults, namely *RetroDMR*, which facilitates to reproduce failures in multiprocessor systems and ensures low-latency fault detection. To be specific, for fault activation, RetroDMR logs the essential events (e.g., the sequence of thread migration) in the faulty run and tries to replay it in the following diagnosis runs, and we employ redundant multithreading (RMT) techniques to construct DMR in the diagnosis runs for low-latency fault detection [5]. By scheduling the original threads based on the logged sequence in the faulty run whenever possible, there is a high probability that we could reproduce the failure and improve the diagnosis resolution for non-deterministic faults.

The remainder of the paper is organized as follows. Section II introduces existing works related to RetroDMR and the motivation of this work. The details of RetroDMR is presented in Section III. Next, we conduct simulation studies to evaluate the efficacy of RetroDMR in Section IV. Finally, Section V concludes this paper.

II. Related Works and Motivation

Existing works related to RetroDMR can be classified into two categories as follows:

Quick Error Detection (QED): In fault-tolerant computing field, RMT [5, 6] is a widely-used technique for transient fault detection and correction. QED [7] borrowed this concept for post-silicon validation, by duplicating the original workload with some transformations for bug localization, referred as RMT-V. QED is shown to be able to reduce error detection latency by up to six orders of magnitude, from billions of cycles to a few thousand cycles. However, QED does not consider the discrepancy between the original faulty run and the following diagnosis runs. Consequently, QED may not fulfill the fault activation condition, especially for tricky electrical bugs.

Deterministic Replay based Debug Methods: Several research efforts have been dedicated for tackling non-determinism of the replay runs in the literature [8, 9]. For

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

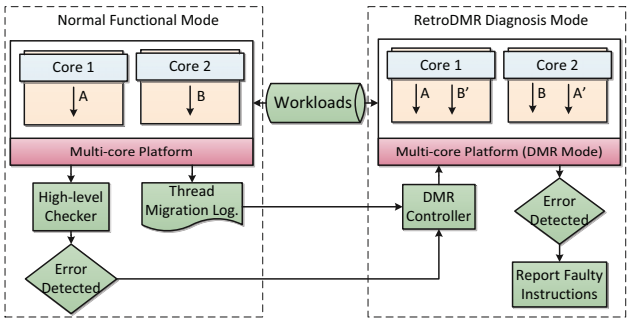

Fig. 1. RetroDMR diagnosis framework, thread A' and B' are check threads of main threads A and B.

instance, [8] exploit log-based deterministic replay mechanism to improve the reproducibility of non-deterministic faults in software. CADRE [9] characterizes the sources of non-determinism in computer systems and address them by ensuring deterministic communication on source-synchronous buses. However, none of these works take detection latency into consideration, which not only degrades bug coverage due to the masking effects during fault propagation but also makes fault localization very challenging.

Motivated by the above, we propose a novel diagnostic framework for non-deterministic faults, namely RetroDMR. To the best of our knowledge, this is the first *holistic* work that explicitly takes both non-deterministic fault reproduction and low-latency fault detection into consideration, as detailed in the following sections.

III. The Proposed Methodology

The proposed RetroDMR framework for diagnosing multi-core platforms is shown in Figure 1, which contains two operation modes: normal functional mode and RetroDMR diagnosis mode.

In normal functional mode, we create operation conditions similar with read cases so as to activate the non-deterministic faults. We run normal system workloads (Thread A and B in Figure 1) at various operational conditions (e.g., different frequencies and temperatures) and at the same time log thread migration sequence. Once errors are detected by *high-level checkers*, for example, checking incorrect outputs, segmentation faults, system crash and so on, we would transform the system to RetroDMR diagnosis mode.

Since high-level checkers usually involve very long latency to detect the errors, it is difficult to locate faulty instructions for root cause analysis. In RetroDMR diagnosis mode, we duplicate partial or all the original threads in the normal functional mode to construct DMR for quick error detection. In order to activate original faults occurred in normal functional runs, we need to reduce the *intrusiveness*[1] introduced. In RetroDMR diagnosis mode, we ensure migration sequences of the main threads (A and B) are the same as those in the normal functional mode based on the log and the check threads (A' and B') are carefully scheduled among cores, which can be done with *DMR Controller* in our framework. The outputs of the main thread and its corresponding check thread are compared and once an error is detected, RetroDMR reports faults within several instructions due to short detection latency.

Apparently, diagnosis runs in RetroDMR diagnosis mode are associated with more threads and thus would not execute in exactly the same way as the faulty runs in normal functional mode. RetroDMR may miss some original faults. However, by scheduling the main threads the same as in normal functional runs and detect errors with low latency DMR, it is highly likely that we could reproduce the original failure and improve diagnosis resolution for non-deterministic faults.

Needless to say, DMR construction and log-based faithful replay are crucial parts for our diagnosis framework. We detail them in the following sections.

A. DMR Construction

DMR is a widely used technique in fault-tolerant computing domain, which provides fine-grained golden traces for fault detection. In RetroDMR, we take advantage of the existing RMT techniques [5] to construct thread-level DMR, in which the memory accesses (e.g., the load and store instructions) are checked between the main threads and their corresponding check threads. For the details of RMT implementation, please refer to [5].

RMT is effective for fault detection with less fault masking effects and short detection latency, which facilitates to improve diagnosis resolution for non-deterministic faults with RetroDMR.

B. Log-based Faithful Replay

In this subsection, we discuss the log-based faithful replay mechanism used in RetroDMR in detail.

1) Log in Normal Functional Mode: In this work, we consider to record the thread migration sequence in the normal functional run for two reasons. Firstly, the thread migration sequence determines the processor core used for instruction execution. Usually, the thread migration policy is sensitive to the operation conditions, such as temperature and resource usage. If we do not guarantee the same thread migration sequence in RetroDMR replay runs, the failing thread triggering the original fault may be scheduled on another error-free core and hence does not fail in RetroDMR replay runs. Secondly, the interval between two thread migration operations usually consists of million cycles [10] due to the migration overhead. As a result, the corresponding logging overhead is quite small.

To record the thread migration sequence in normal functional run, we log the committed instruction count (CIC) executed on the current core for each thread when thread migration occurs. Logging CIC within each thread migration transaction can deterministically decide the relationship between instructions and their corresponding execution cores. This is because, instructions are committed with the same order between different runs and CIC value monotonously increases as the thread executes.

[1]Intrusiveness is loosely defined as the amount of "deviation" in the execution behaviors of following RetroDMR replay runs and the normal functional run.

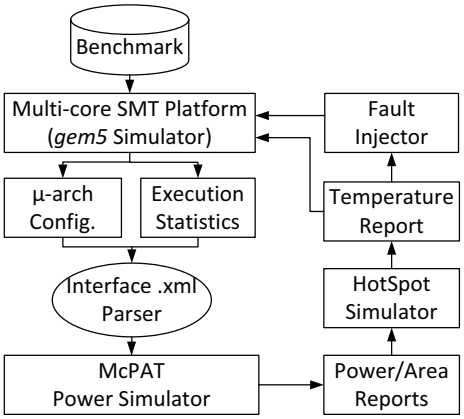

Fig. 2. Simulator design.

2) Faithful Replay in RetroDMR Mode: Based on the logged thread migration events, in RetroDMR replay mode we can "faithfully replay" the original thread to reproduce faults appeared in the normal functional mode. In RetroDMR, duplicating original threads would inevitably introduce intrusiveness. Intuitively, the more threads duplicated, the more intrusiveness introduced. Clearly, we can configure the number of duplicated threads to control the introduced intrusiveness. However, if we do not duplicate all the original threads in a single replay run, we have to conduct replay runs multiple times, so that all the original threads are duplicated to achieve full check capability. It is worth noting that, due to thread communication, fault coverage may be degraded if related threads are not all duplicated in one run. In practice, we can configure different numbers of duplicated threads for RetroDMR based on our diagnosis needs.

In RetroDMR replay mode, we would force the migration sequence of the main thread to be the same as that in the normal functional mode according to our log. The check thread and its corresponding main thread should be executed on different cores whenever possible, in order to achieve high fault coverage.

IV. Evaluation of RetroDMR

In this section, we evaluate the efficiency and effectiveness of RetroDMR with simulation studies.

A. Simulation Framework

1) Simulator: The overview of our simulation framework is shown in Figure 2. First, we use *gem5* to simulate the workload execution. To construct DMR, we exploit the SMT mechanism in the simulator to run multiple threads on each core, and the checker part only compares the Load-Store results between the main thread and check thread.

Next, to get the runtime temperature information of the system, we periodically dump the execution traces and transfer those information together with micro-architecture configurations to a power simulator McPAT via the interface parser. The output of McPAT, i.e., the power and area reports (e.g., the power consumption of each functional blocks and their areas), serve as the inputs to the temperature simulator, HotSpot. The simulated temperature reports are used for two purposes: one for default thread migration policy, and the other for injecting temperature-sensitive faults.

2) Simulation Workloads: We use MiBench [12] as our benchmark. However, the runtime of the provided benchmark programs are short, and the default thread migration policy is sensitive to temperature, whose change is a slow process compared to the benchmark execution. To resolve this issue, we regard the benchmarks in MiBench as unit workloads and generate longer workloads randomly. We group several benchmark applications as a benchmark set (BS) for multi-core system simulation.

B. Fault Model

Due to the very nature of non-deterministic faults, there are no well-accepted models for such faults in the literature[2]. For the sake of simulation, we adopt the following three faulty scenarios based on the manifested behaviors of non-deterministic faults:

- **Temperature-sensitive faults**: Since electrical bugs induced non-deterministic faults only occur under specific operating conditions, we inject certain temperature-sensitive faults, where the faulty hardware unit will generate errors only when their operational temperature is greater than a specific value.
- **Intrusiveness-sensitive faults**: Intuitively, fault activation probability would decrease as intrusiveness increases. In this fault scenario, we randomly choose a few instructions in the normal functional run to be faulty, and assume faults will be activated on them only when intrusiveness in the diagnosis runs is smaller than a predefined threshold σ.
- **Faults from [11]**: These faults are coming from post-silicon validation of actual multi-core SoCs that take very long time to diagnose. In this experiment, we choose one scenario from the fault list shown in [11]. Its activation criterion is as follows: there are two branch instructions within 8 cycles, and its effect is that the core will jump to the incorrect address in the next cycle.

C. Results

In this section, we present experimental results by injecting faults according to the above fault models for different diagnosis runs, and compare fault activation and detection capabilities with QED proposed in [7].

1) Fault Activation: For temperature- and intrusiveness-sensitive faults, we assume one of the two FpMultDiv units of a specific core is faulty. In the normal functional run, we record the number of inserted faults and their locations (e.g., faulty instructions). We run RetroDMR with two configurations: RetroDMR_1 with only one duplicated thread (run 4 times to cover all the original threads) and RetroDMR_4 with all the original 4 threads duplicated. We record the percentage of

[2]If we were able to model non-deterministic faults in a satisfactory manner, they would not be so difficult to diagnose.

Benchmark Sets	Temperature-sensitive Faults			Intrusiveness-sensitive Faults			Faults from [11]		
	QED	RetroDMR_4	RetroDMR_1	QED	RetroDMR_4	RetroDMR_1	QED	RetroDMR_4	RetroDMR_1
BS1	35.0%	62.8%	98.9%	27.4%	64.7%	94.6%	41.8%	94.9%	100%
BS2	34.2%	69.1%	97.5%	26.7%	58.9%	90.8%	36.7%	95.2%	100%
BS3	38.3%	77.6%	97.4%	31.1%	62.0%	93.7%	50.8%	93.3%	100%
BS4	52.2%	74.0%	99.0%	29.4%	57.3%	94.2%	62.8%	90.7%	100%
BS5	45.9%	64.0%	98.2%	29.1%	59.2%	94.6%	56.1%	93.6%	100%

TABLE I
THE PERCENTAGE OF ORIGINAL FAULTS ACTIVATED BY QED [7] AND RETRODMR.

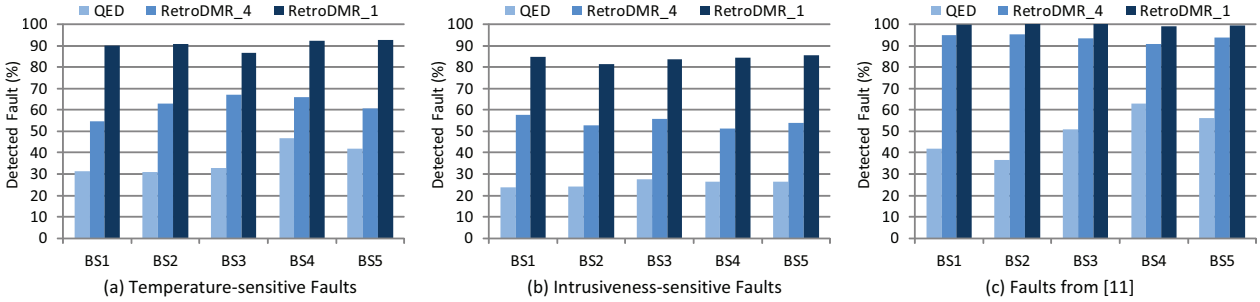

Fig. 3. Detection capability when injecting different faults at FpMultDiv.

original faults occurred at the normal functional runs activated by QED and RetroDMR as shown in Table I.

According to the results, for temperature-sensitive faults, QED can only activate about 34%-52% of the original faults. This is because, faults are injected at floating point unit, and hence only those threads with floating point operations running on the faulty core can trigger the fault, while other threads cannot. The thermal distribution of QED runs are quite different from the normal functional run. Consequently, the thread migration sequence is also quite different, rendering lots of original faults unactivated. RetroDMR_4 can activate about as twice as many faults than QED. However, it still misses quite some faults occurred in the normal functional run. This is because, even though the faulty threads are running on the faulty core, the threads may not use the faulty unit due to the fact that there are multiple floating point units in one core. This can be compensated with RetroDMR_1 that can activate almost all of the original faults. Because the duplicated thread running on the error-free core has little effect on the faulty core which runs exactly the same workload as the normal functional run. Similarly, for intrusiveness-sensitive faults and faults from [11], RetroDMR can activate most of the original faults, much more than those activated with QED.

2) Fault Detection: We compare the fault detection for RestroDMR and QED. Figure 3 presents the percentage of detected faults, in which RetroDMR (including RetroDMR_1 and RetroDMR_4) obtains better results than QED for all benchmark sets, since RetroDMR activates more faults in the original run, even though they all use RMT for fault detection. At the same time, not all the activated faults can be detected. This is because, RMT only checks the memory accesses (load and store instructions) and some injected faults may be masked before propagating to these instructions.

With the help of RMT, the detection latency of RetroDMR is also quite small, when compared to try-and-error type of diagnosis methodology that relies on high-level checkers for fault detection, whose fault detection latency can easily reach millions of cycles [7].

V. CONCLUSION

In this work, we propose a novel retrospective diagnostic framework, named as RetroDMR, for troubleshooting non-deterministic faults. We log the thread migration sequence in the original functional run and then replay it with similar behaviors, and we take advantage of RMT techniques for short-latency error detection. By scheduling the original threads based on the logged information, there is higher probability to reproduce faults appeared in the original run. Experimental results based on *gem5* simulator show that our proposed diagnosis methodology is both effective at reproducing the original failures and efficient in terms of error detection latency.

REFERENCES

[1] S. Mitra, S.A. Seshia and N. Nicolici. Post-silicon validation opportunities, challenges and recent advances. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, 2010.

[2] A. DeOrio, D.S. Khudia and V. Bertacco. Post-silicon bug diagnosis with inconsistent executions. In *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 755-761, 2011.

[3] S. Park and S. Mitra. IFRA: Instruction Footprint Recording and Analysis for Post-Silicon Bug Localization in Processors. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 373-378, 2008.

[4] M. Li, et al. Accurate Microarchitecture-Level Fault Modeling for Studying Hardware Faults. In *Proc. IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 105-116, 2009.

[5] S.S. Mukherjee, M. Kontz and S.K. Reinhardt. Detailed design and evaluation of redundant multi-threading alternatives. In *Proc. IEEE International Symposium on Computer Architecture (ISCA)*, pp. 99-110, 2002.

[6] S.K. Reinhardt and S.S. Mukherjee. Transient Fault Detection via Simultaneous Multithreading. In *Proc. IEEE International Symposium on Computer Architecture (ISCA)*, pp. 25-36, 2000.

[7] T. Hong, et al. QED: Quick Error Detection Tests for Effective Post-Silicon Validation. In *Proc. IEEE International Test Conference (ITC)*, pp. 1-10, 2010.

[8] N. Honarmand and J. Torrellas. Replay Debugging: Leveraging Record and Replay for Program Debugging. In *Proc. IEEE International Symposium on Computer Architecture (ISCA)*, pp. 445-456, 2014.

[9] S.R. Sarangi, B. Greskamp and J. Torrellas. CADRE: Cycle-Accurate Deterministic Replay for Hardware Debugging. In *Proc. IEEE International Conference on Dependable Systems and Networks (DSN)*, pp. 301-312, 2006.

[10] T. Constantinou, et al. Performance implications of single thread migration on a chip multi-core. In *ACM SIGARCH Computer Architecture News*, 2005.

[11] D. Lin, et al. Quick detection of difficult bugs for effective post-silicon validation. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 561-566, 2012.

[12] M.R. Guthaus, et al. MiBench: A free, commercially representative embedded benchmark suite. In *IEEE International Workshop on Workload Characterization (WWC)*, pp. 3-14, 2001.