Reverse Debugging with ETM

Wenxuan SHI, Xueying ZHANG and Haonan LI

The plan

Wenxuan SHI & Xueying ZHANG: find a research topic for Group Project.

Haonan LI: NULL

The work

Topic: **reverse debugging**

Wenxuan

1. Some research on gdb reverse debugging
2. Read a paper “DoublePlay: Parallelizing Sequential Logging and Replay”.

Xueying

1. Read the document of ETMv4 to catch up with the progress.

Haonan

1. Read a paper: “iReplayer: In-situ and Identical Record-and-Replay for Multithreaded Applications”

ETM

- real time
- data trace: not supporded on ARMv8
- instruction trace:
    - PE -(some instractions)-> trace unit(resources) -> filter(programmable) -(trace stream)-> trace analyzer
    - encode(trace unit) and decode(analyzer)
    - return stack
    - synchronize information
    - contains: virtual address and ‘system state’ (EL, securtity state, condition, etc.)

GDB: Reverse Debugging with Record and Replay

- GDB can **record** a log of process execution and save it.
- This record can be loaded later on, and used for debugging. This is called offline debugging.
- It offers the advantage that you can catch the issue once, and **replay** it as much as needed to find the root cause and fix it.

Performance issue

To realize this functionality, GDB is in fact executing the software, one assembly instruction after another and **recording relevant registers and memory locations**.

This is a slow operation that can drastically change the timing of process execution, and thus **change the conditions that raise the bug.**

GDB solution

- Use SoC IPs to accelerate the operation.
- GDB has support for “Processor Trace (PT)” and “Branch Trace Store (BTS)” IP on Intel processors.

Limitation

- It doesn’t support ARM
- If hardware acceleration is enabled, only **execution flow** is record. (branch record)

Our work

- Use ETM to accelerate recording
- Try to trace **data flow** (knowing the exact value change in memory and register)
- Try to support debugging on **multi-core**

Multicore: issue on recording

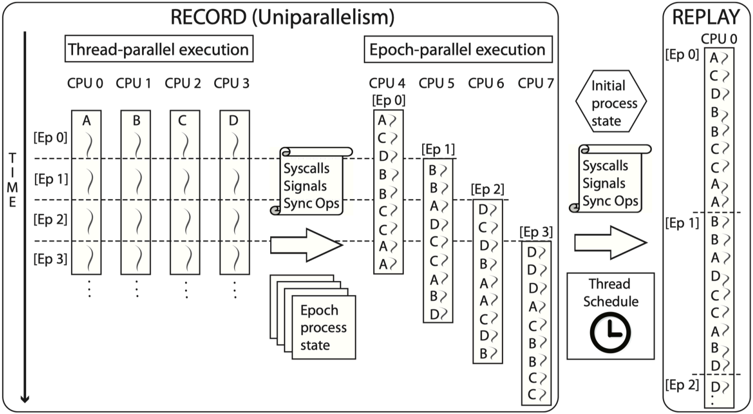
- shared memory (shared **source of truth**)
- order matters!

Common solution

turn multicore program into a **equivalent** unicore program.
(equivalent: two program beginning at same status end at same status.)

DoublePlay

K. Veeraraghavan et al., “DoublePlay: Parallelizing Sequential Logging and Replay,” ACM Trans. Comput. Syst., vol. 30, no. 1, pp. 1–24, Feb. 2012, doi: 10.1145/2110356.2110359.


RECORD (Uniparallelism)
Thread-parallel execution
Epoch-parallel execution
CPU 0
CPU 1
CPU 2
CPU 3
A
B
C
D
[Ep 0]
[Ep 1]
[Ep 2]
[Ep 3]
TIME
Syscalls
Signals
Sync Ops
Epoch process state
CPU 4
CPU 5
CPU 6
CPU 7
[Ep 0] A C D B B C C A A
[Ep 1] B B A D C C A B D
[Ep 2] D C D B A A C D B
[Ep 3] D D D A C B B C C
Initial process state
Syscalls
Signals
Sync Ops
Thread Schedule
REPLAY
CPU 0
[Ep 0] A C D B B C C A A
[Ep 1] B B A D C C A B D
[Ep 2] D

Details at “My notes on DoublePlay-Parallelizing-Sequential-Logging-and-Replay”

Paper Introduction

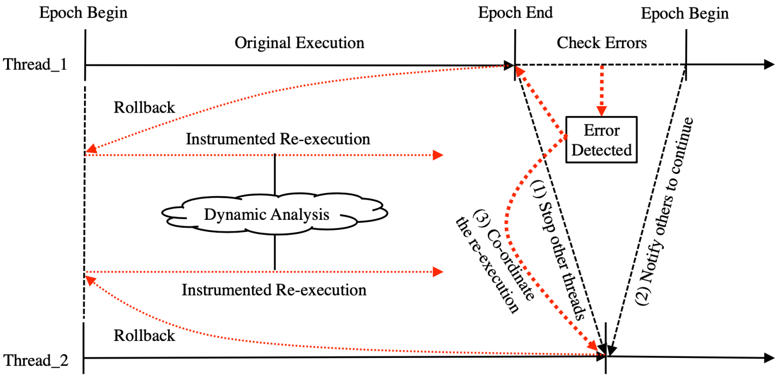
- Hongyu Liu, Tongping Liu et al. “iReplayer: In-situ and Identical Record-and-Replay for Multithreaded Applications”, PLDI’18
- University of Texas at San Antonio, Huawei US Lab
- Only replay the execution **if necessary**
- ~~This paper was rejected 7 times~~

Types of Replay

- *Lawful*: replay is re-execution (ReVirt)
- *Neutral*: capture/snapshot is also replay (TTD, REPT)
- *Chaotic*: rollback is also replay (iReplayer)

iReplayer

Desgin goal: in-situ, identical, efficient



Syscalls in Different Types

Category	Syscall Examples
Repeatable	`getpid`, `getcwd`
Recordable	`gettimeofday`, `mmap`, `open`
Revocable	file `read`/`write`
Deferrable	`close`, `munmap`, (thread exits)
Irrevocable	`fork`, `lseek`

About REPT: OSDI'18

- "Reverse Debugging of Failures in Deployed Software"
- They adapted and deployed in WinDbg (see: https://youtu.be/0VUy4mqA_Lk)
- An improvement of *Time Travle Debugging*

Next Week Plan

- For these categories of syscalls, to find some ways to record