Hardware-assisted Tracing for Low Overhead Data Race Detection

Farzam Dorostkar with Pr. Michel Dagenais

May 16th 2022

Polytechnique Montreal

DORSAL Laboratory

Project Introduction

Research Topic: Low overhead memory bug detection using hardware tracing

Current Track:

Detecting data races in C/C++ programs that use POSIX pthreads

❖ Post-mortem data race detection using Intel Processor Trace (Intel PT)

Agenda

- Introduction
  - Data Race
  - Motivation
  - Intel Processor Trace (Intel PT)
- Methodology
  - Opportunities & Limitations
  - Algorithm
  - Hybrid Tracing
  - Preliminary Results
  - Tools
- Conclusion & Future work

Introduction

Introduction: Data Race

In multithread Programming:

- Shared variables allow threads to communicate quickly
- A bug when two+ threads access the same shared variable concurrently and at least one access is a write (Data Race!)

Introduction: Data Race

In multithread Programming:

- Shared variables allow threads to communicate quickly
- A bug when two+ threads access the same shared variable concurrently and at least one access is a write (Data Race!)

t1 R	t2 R	t1 R
t1 W	t2 W	t2 R
t2 R	t1 R	t1 W
t2 W	t1 W	t2 W
count = 2	count = 2	Count = 1

```
int count = 0; //Shared variable

void *routine_one(void *arg) {
   count++;
}

void *routine_two(void *arg) {
   count++;
}

int main() {
   pthread_t t1, t2;
   pthread_create(&t1, NULL, &routine_one, NULL);
   pthread_create(&t2, NULL, &routine_two, NULL);
   pthread_join(t1, NULL);
   pthread_join(t2, NULL);
   return 0;
}
```

Introduction: Motivation

Why a new data race detector?

- ❖ Current standard tools impose considerable overhead!
  - ThreadSanitizer (TSan):
    - Slowdown: 5×-15× & Memory overhead: 5×-10×
  - Helgrind:
    - Slowdown: 100× & Memory overhead: 20×
- ❖ Not usable in production + difficult to test applications under real-world loads

Reason: Sole reliance on heavy code instrumentation!

Research question: Can we reduce the need for code instrumentation in a data race detector by making use of hardware-assisted tracing?

Introduction: Intel Processor Trace (Intel PT)

- A hardware feature that logs information about software execution with minimal impact
- A non-intrusive means to trace control flow by generating a variety of packets
  - <5% performance overhead
- Decoder reconstructs the precise execution flow by combining PT packets with the binaries of the traced program
- Trace data is generated only for non-statically-known control flow changes
- Can store both cycle count and timestamp information
- No need to modify source code!
  - Run under Intel PT-enabled debug and profiling tools (like Linux perf & GDB)

Introduction: Intel Processor Trace (Intel PT)

Control Flow Tracing

- TNT (Taken Not-Taken) : direct conditional branches (generates only a 1-bit indication)
- TIP (Target IP) : target address of indirect branches, exception, and interrupts

Instructions	Encoding
push	
mov	
cmp	
je .L1	TNT (Taken Not Taken)
mov	
add	
L1:	
Call (edx) // virtual function	TIP (Target IP)

PT trace
T
0x4012a4

Introduction: Intel Processor Trace (Intel PT)

Reconstructing the Control Flow

Decoder can determine the exact execution flow from trace log

Instructions
push
mov
cmp
je .L1
mov
add
L1:
Call (edx) // virtual function

+

PT trace
T
0x4012a4

decode

Reconstructed Control Flow
push
mov
cmp
je .L1
mov
add
L1:
Call (0x4012a4)

Methodology

Methodology: Opportunities & Limitations

What are the opportunities and limitations of using HW tracing for detecting data races?

↘

What information is required for detecting data races?

↘

How data races are detected?

Data race detection algorithms

- Happens-before
- Lockset

Methodology: Algorithm

Happens-before Algorithm

- Determines partial ordering between program events
- Inspired by Lamport's Happens-before relation [1]
- For two events *a* and *b*:

  i. If *a* and *b* in the same thread & *a* comes before *b* : $a \rightarrow b$

  ii. If (*a, b*) is a synchronization-pair (like lock/unlock the same mutex) : $a \rightarrow b$

  iii. If $a \rightarrow b$ & $b \rightarrow c$: $a \rightarrow c$ (Transitivity)

  If $a ! \rightarrow b$ & $b ! \rightarrow a$: *a* and b are concurrent!

- Used by many tools including TSan, Helgrind, and GO's built-in data race detector

How data races are detected? ✓

[1] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," Commun. ACM, 21(7):558–565, 1978.

Methodology: Algorithm

Events of interest:

- ❖ Memory accesses:
  - reads and writes
- ❖ Synchronization:
  - Unlocking and locking the same mutex
  - Signaling a condition and waiting on the same condition
  - Broadcasting a condition and waiting on the same condition

Other required information:

- ❖ Control flow related information (like order of events)

What information is required for detecting data races? ✓

Methodology: Hybrid Tracing

An Example Decoded PT Trace

Two threads updating the global object "count" while holding the mutual lock "mutex"

TID	IP	Addr	Sym+off	Insn
20295	0	4011dc	routine_two+0xc	mov $0x404060, %rdi
20295	401080	4011e6	routine_two+0x16	callq 0xfffffffffffffe9a
20295	0	4011eb	routine_two+0x1b	movl 0x40405c, %ecx
20295	0	4011f5	routine_two+0x25	movl %ecx, 0x40405c
20295	0	4011fc	routine_two+0x2c	mov $0x404060, %rdi
20295	401050	401209	routine_two+0x39	callq 0xfffffffffffffe47
20294	0	40118c	routine_one+0xc	mov $0x404060, %rdi
20294	401080	401196	routine_one+0x16	callq 0xfffffffffffffeea
20294	0	40119b	routine_one+0x1b	movl 0x40405c, %ecx
20294	0	4011a5	routine_one+0x25	movl %ecx, 0x40405c
20294	0	4011ac	routine_one+0x2c	mov $0x404060, %rdi
20294	401050	4011b9	routine_one+0x39	callq 0xfffffffffffffe97

Symbol	Addr
count	0x40405c
mutex	0x404060
pthread_mutex_lock	0x401080
pthread_mutex_unlock	0x401050

- Control flow information ✓
- Calls to synchronization functions ✓
- Accesses to the global variable ✓

No instrumentation required for this example

Methodology: Hybrid Tracing

In General:

❖ Included in PT Trace:

- Control flow related information
- Calls to synchronization functions of pthreads
- Accesses to global objects

$\}$ Hardware-assisted Tracing

❖ Not Included in PT Trace:

- Accesses to heap objects
- Accesses to stack objects
- Indirect pointer-based memory accesses

$\}$ Software Tracing (i.e. Instrumentation)

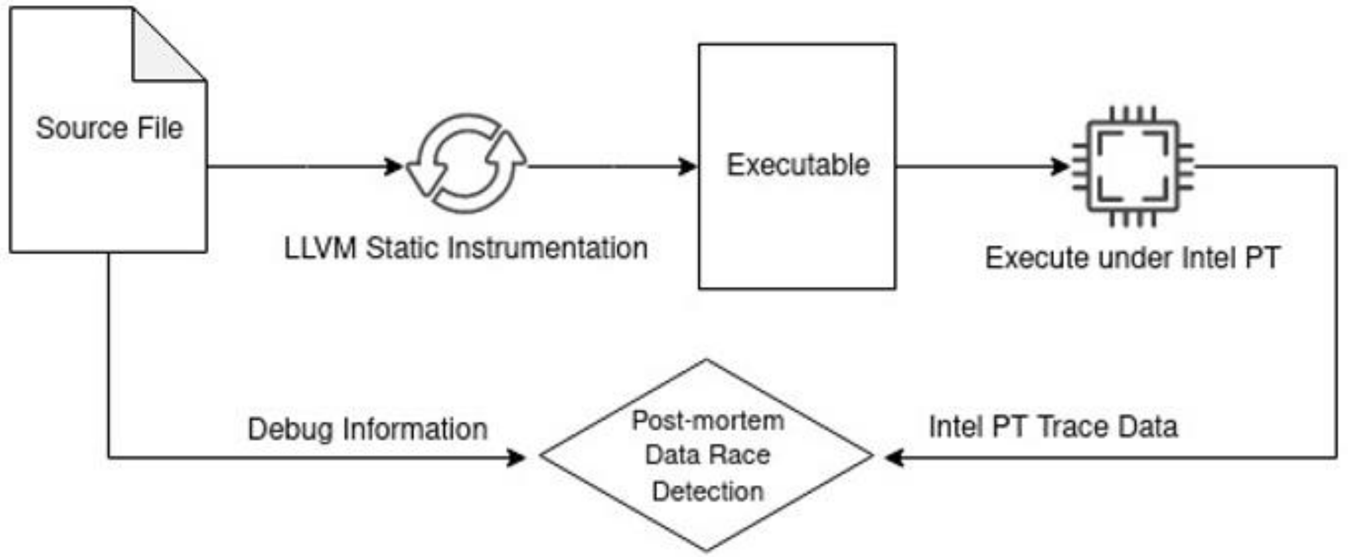
What are the opportunities and limitations of using HW tracing for detecting data races? ✓

Methodology: Hybrid Tracing

Instrumenting Most Memory Accesses

At the current stage:

- Static instrumentation
- Using the LLVM infrastructure
- Instrumenting **`load`** and **`store`** instructions at IR level with dummy function calls



When the execution is over, and possibly on a different machine, the Intel PT trace data is decoded and analysed for possible data races.

Methodology: Preliminary Results

Benchmark	Benchmark Characteristic			Runtime Overhead			Memory Overhead		
	#Threads	R/RF	Object Type	Helgrind	TSan	Proposed	Helgrind	TSan	Proposed
#1	3	Race-free	Global	99.4×	4.9×	1.8×	23.4×	7.6×	3.1×
#2	3	Data Race	Global	20.2×	231.1×	41.6×	22.3×	7.4×	2.9×
#3	6	Data Race	Heap	76.7×	38.8×	3.5×	22.2×	10.5×	2.8×
#4	3	Data Race	Stack	327.3×	747.4×	68.7×	22.2×	8.0×	2.8×
#5	6	Race-free	Heap	62.4×	6.4×	3.8×	22.1×	9.6×	2.7×

- Five C micro benchmarks
- No false positive or false negative report
- Proposed approach:
  - Noticeably less memory overhead, for all the five benchmarks
  - The least runtime overhead for four benchmarks
  - Performing the post-mortem analysis took between 3 minutes (for benchmark #4 / 11 MB) to 16 minutes (for benchmark #5 / 347 MB)

Conclusion & Future Work

Conclusion & Future Work

❖ Conclusion:

- A hybrid data race detection tool that benefits from hardware-assisted tracing to minimize the need for code instrumentation
- Only instrumenting memory accesses. No need to instrument synchronization events.
- Promising preliminary results in comparison with Helgrind and TSan

❖ Future Work:

- More efficient instrumentation + Dynamic Instrumentation
- Exploit the potential of static code analysis to specify memory accesses that do not affect the correctness of data race detection if not instrumented
- Investigating the potential of the PTWRITE instruction to further reduce the need for instrumentation

Questions?

farzam.dorostkar@polymtl.ca
https://github.com/FarzamDorostkar