

When Sampling Lies: Trustworthy Performance Profiling for Flat Workloads with *Blink* (Operational Systems)

Rishikesh Devsot, ChenXing Yang, Yi Fan Yu,
Prabhdeep Singh Soni, Afshin Arefi, Bryan Chan, Reza Azimi, Ding Yuan



UNIVERSITY OF TORONTO

YScope



Who Are We?

- Compiler engineers whose job is to optimize OpenHarmony OS

- Deployed in **1.2 billion devices worldwide**



A Compiler Engineer's Reality

Optimize workloads



Perf: Measure and validate impact



Deploy on OpenHarmony OS

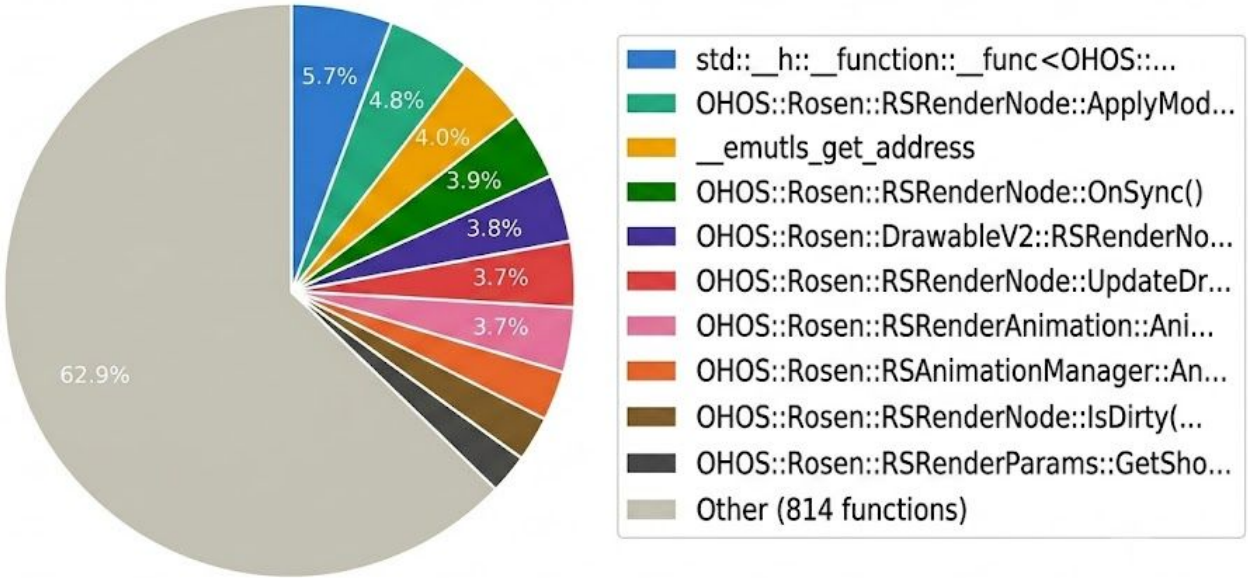
OpenHarmony



1.2 billion devices

Flat Profile Workloads

Function distribution of a rendering workload



A Compiler Engineer's Reality

Optimize workloads



🏆

1% improvement is a big win

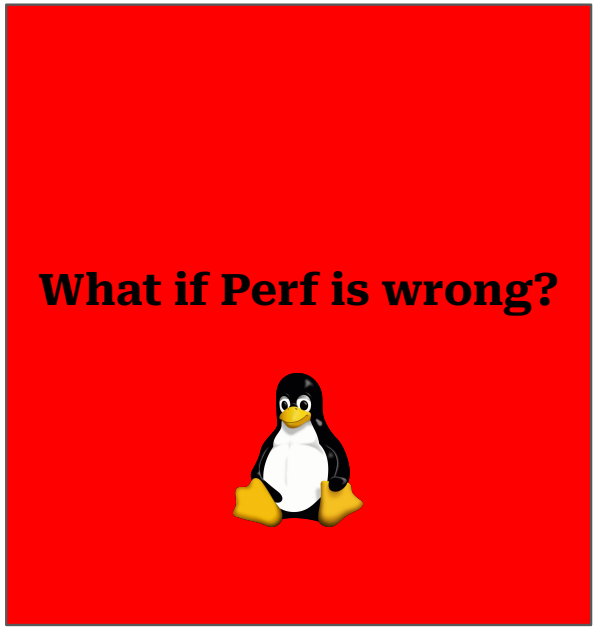
Perf: Measure and validate impact



Deploy on OpenHarmony OS

OpenHarmony

A Compiler Engineer’s Reality



Real World Example

- Large System Extension Optimization (LSEO)
- Optimization to improve performance of atomic operations
- Armv8.1-A architecture introduced new atomic instructions.
- Huawei used LSEO to improve atomic decrement in `librender`.

LSEO: Reduce Instruction Count

Before LSEO

```
retry:
    ldaxr x9, [x8]
    sub x10, x9, #1
    stlxr w11, x10, [x8]
    cbnz w11, goto retry
```

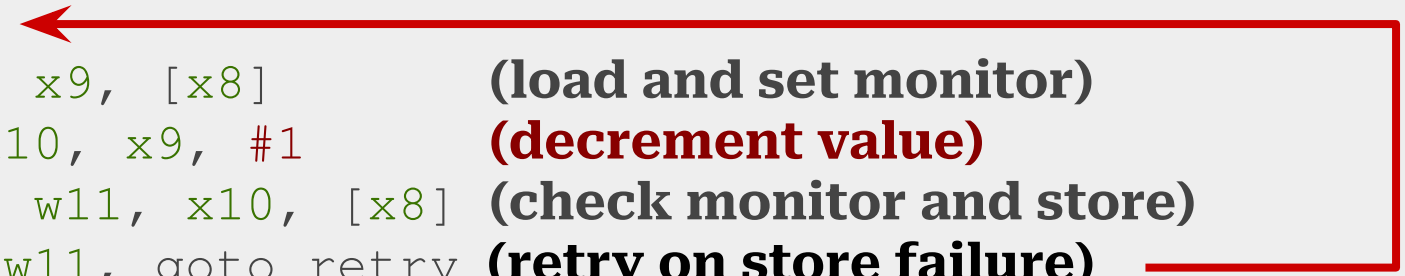
After LSEO

```
mov x9, #-1
ldaddal x9, x8, [x8]
```

LSEO Optimization: Improve Atomic Decrement

Atomic decrement **before LSEO**

```
retry:
    ldaxr x9, [x8]          (load and set monitor)
    sub x10, x9, #1         (decrement value)
    stlxr w11, x10, [x8]    (check monitor and store)
    cbnz w11, goto retry    (retry on store failure)
```



LSEO Optimization: Improve Atomic Decrement

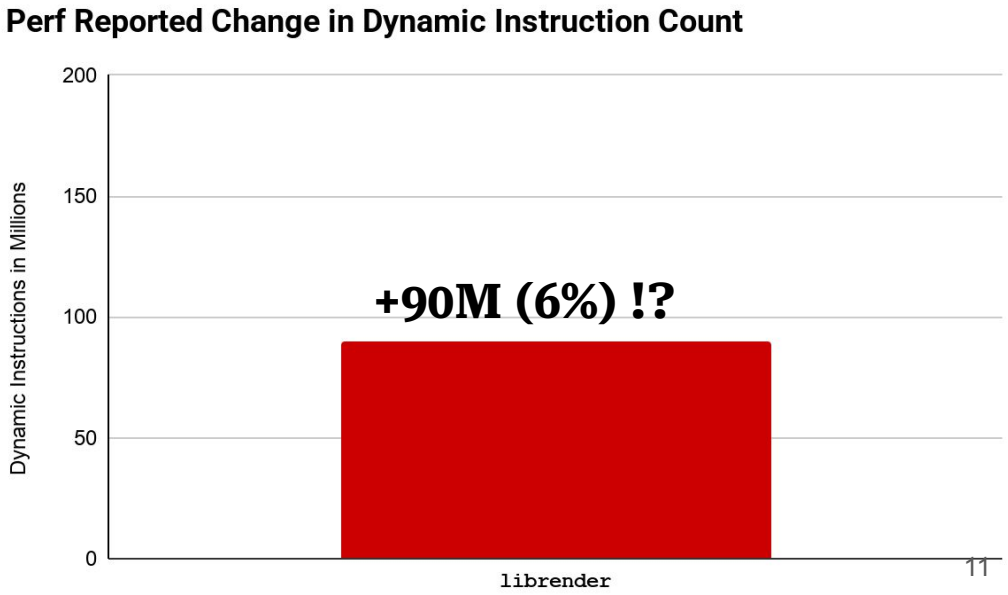
Atomic decrement **after LSEO**

```
mov x9, #-1              (store -1)
ldaddal x9, x8, [x8]     (hardware atomic add)
```

LSEO Optimization: The Mystery

- Used `perf record` to verify *decrease* in dynamic instruction count.
- However, observed an *increase* for librender

Contradiction:
LSEO *decreases* static instruction count but Perf reports an *increase* in dynamic instruction count.



LSEO Optimization: When Sampling Lies

- We trusted perf's result and spent a whole month trying to debug any side effects of the optimization *but to no avail*.

Conclusion:
Perf's results are ***wrong.***

Hypothesis that fits the data: ***Skid and Shadow effect***

How Perf Records Dynamic Instruction Counts

- Sampling based tools record a sample after *N* hardware *e*vents.
- After N events, the executing instruction is assigned the sample.
- The # samples in a function decide it's final event count.



Perf primes PMU

Perf sets a sampling period *N* on the PMU counter

Program Execution

PMU counter is incremented when an instruction retires.

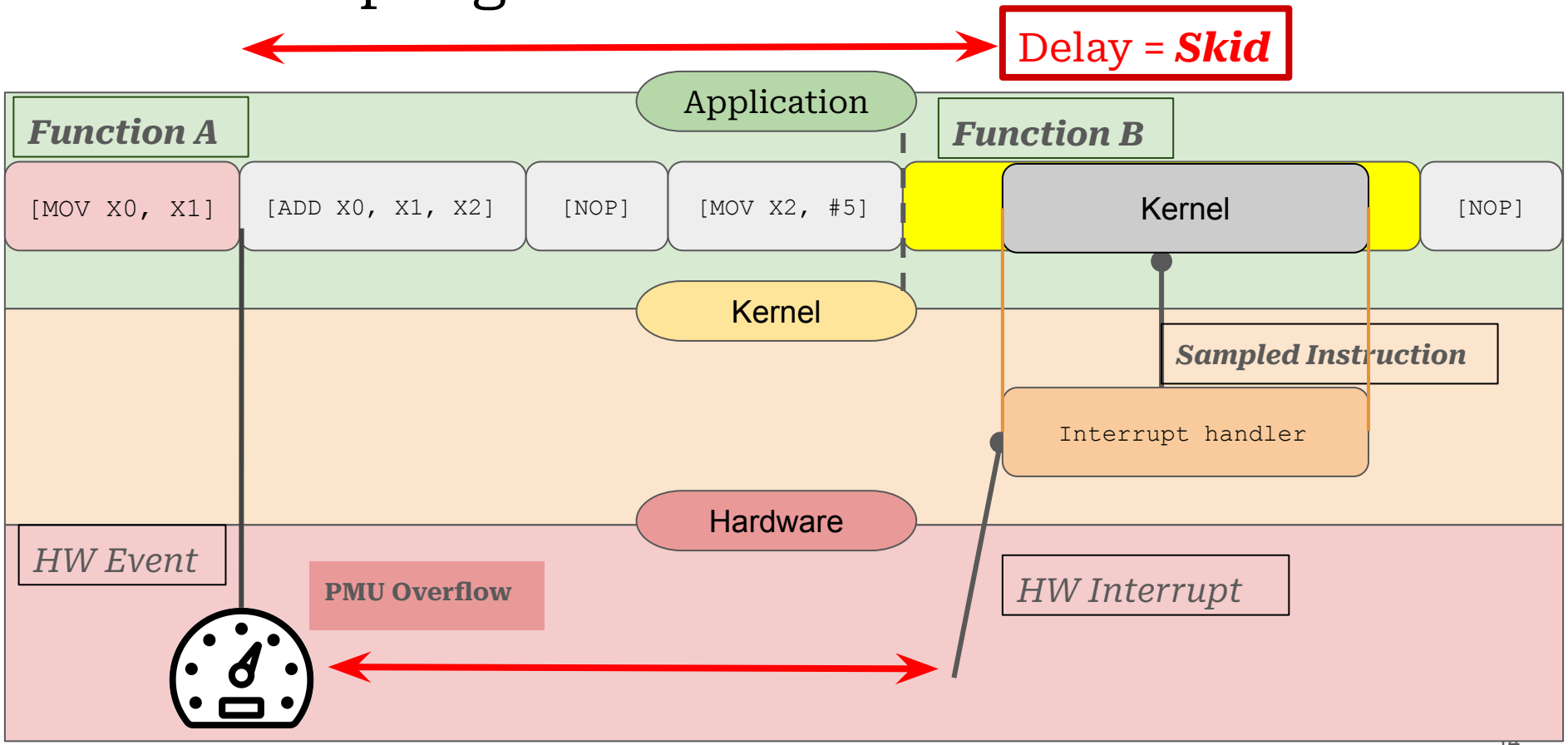
PMU Interrupt

When counter == N, PMU sends an interrupt and resets

Sample

Interrupt handler samples executing instruction.

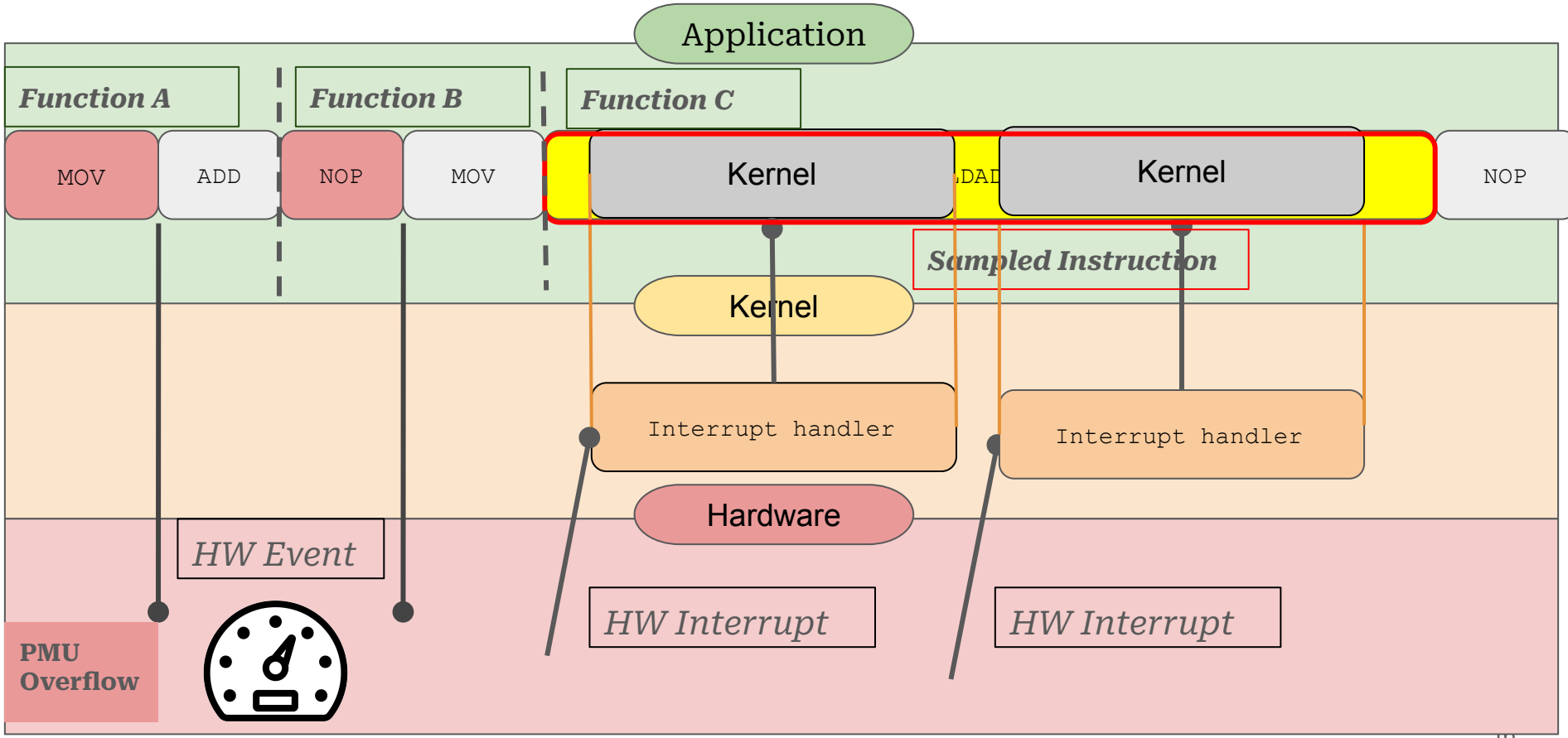
When Sampling Lies: Skid



When Sampling Lies: Skid

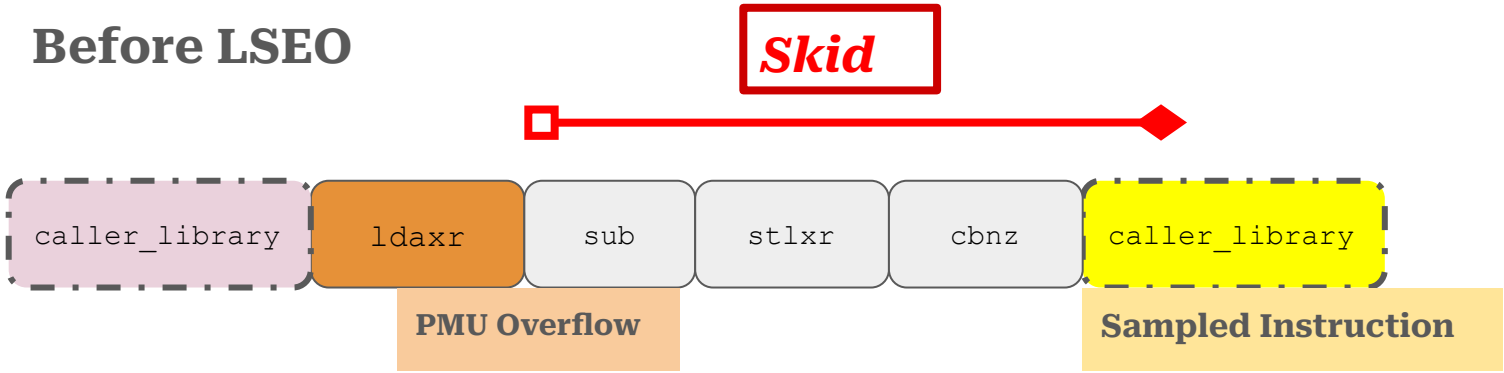
- Hardware delay is random noise
- Can be counteracted by measuring a workload multiple times.
- **Skid combined with Shadow Effect** leads to **consistent inaccuracy**

When Sampling Lies: Shadow Effect

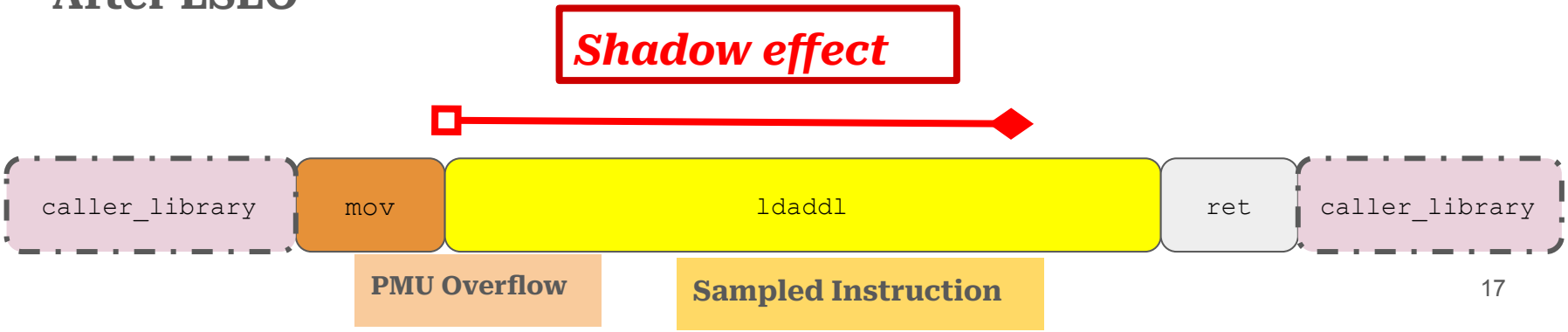


LSEO: Skid And Shadow Effect

Before LSEO

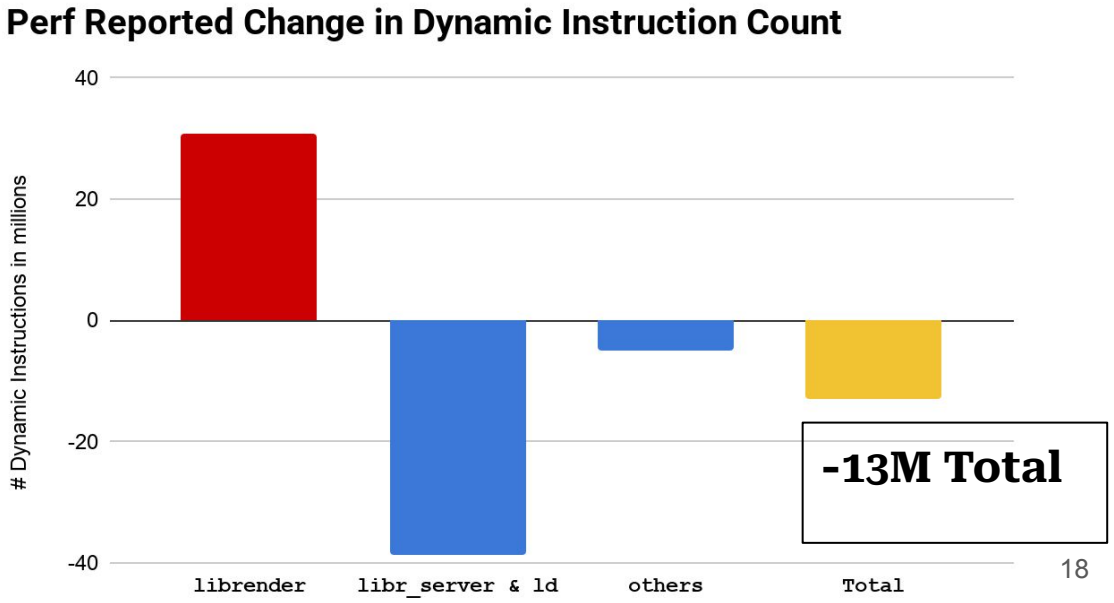


After LSEO



LSEO: Sampling Lies

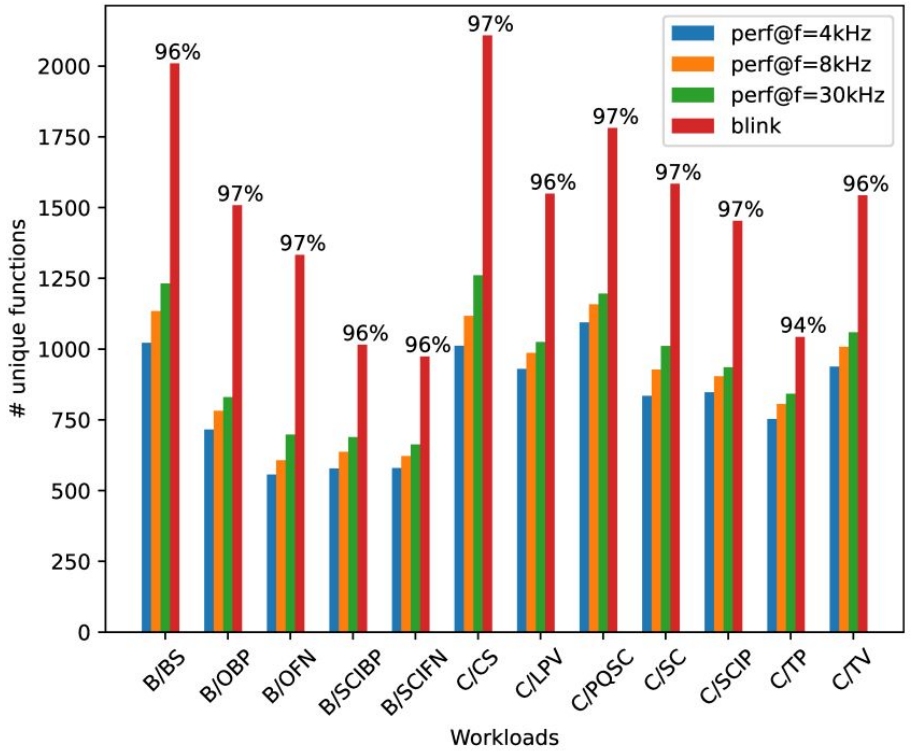
- Optimization applied to `librender` **only**
- Other libraries are showing a difference.



Conclusion:
Sampling misattribution is the reason for incorrect results.

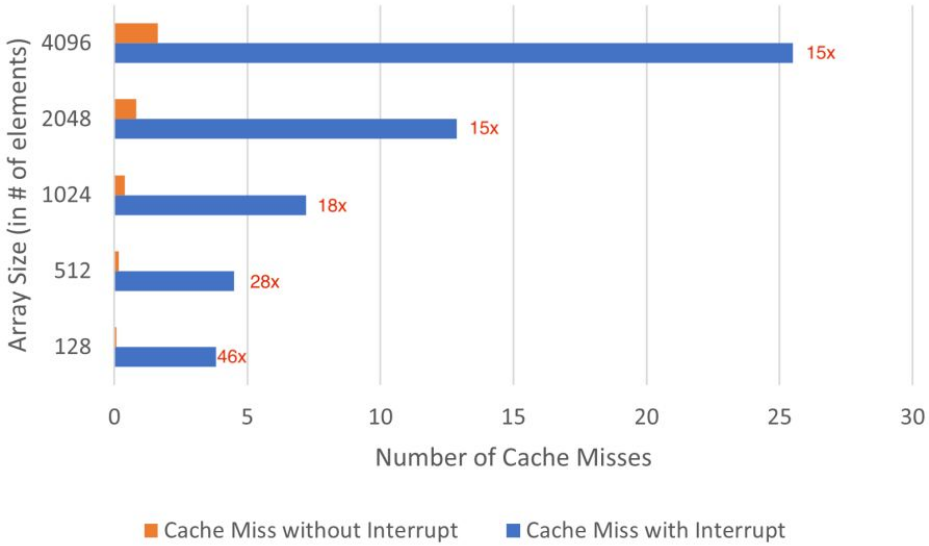
More Sampling Lies: Low Coverage

- Rendering workload has a ***flat profile*** – thousands of short lived functions
- Sampling is biased to measuring long latency functions.
- Even at max frequency (30kHz), Perf only covers 62% of executed functions



More Sampling Lies: Perturbation

- Kernel interrupt handler handles PMU interrupts
- Interrupt handler pollutes the cache leading to *cache misses*.
- Perturbs the behaviour of the executed program.
- ***We can't trust the sampled behaviour.***



A Compiler Engineer’s Reality

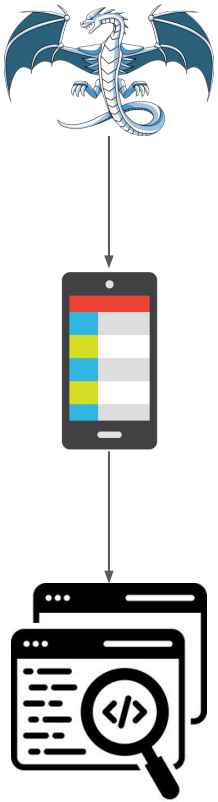


Solution: Blink

Blink: A Trustworthy Tracing Tool

- Achieves an *accuracy of* ***99.999%***
- *Covers* ***96%*** of all functions executed
- Maintains *less than* ***1%*** *overhead*

Blink: How it works



Compile time instrumentation: Instrument the entry and exit of every function

Trace collection at runtime: Blink collects and stores the traces in thread-local buffers.

Trace analysis: Generated trace files are analyzed to produce per-function summaries.

Blink: Precise Tracing

- **Maximum Coverage:** Blink instruments every function, so we trace every compiled function.
- **No perturbation:** Blink reads PMU counter directly. No interrupt based perturbations.
- **No Sampling Inaccuracy:**
  Exit PMU - Entry PMU = Precise Delta

```
target_fn() {
    blink_trace()
        // target_fn() logic
        …
    blink_trace()
}
```

Blink: Lightweight Tracing

Tracing every function adds significant overhead

- Solution: **Self-patching**
- Trace a function upto a threshold (e.g. 30 traces)
- Once hit, tracing logic ***overwrites its own binary***
- Overhead of a disabled trace is one jump.

Enabled

```
blink_trace:
1.  nop // Entry point
2.  stp x0, x1, [sp,-16]! // Save args
3.  <tracing_logic>
4.  mrs x0, pmccntr_el0  // Read PMU
```

Disabled

```
blink_trace:
1.  ~~nop~~ -> b +13 // Jump
2.  stp x0, x1, [sp,-16]! // Skipped
3.  <tracing_logic>       // Skipped
4.  mrs x0, pmccntr_el0  // Skipped
```

Blink: Lightweight Tracing

- ***Hand optimized assembly*** to minimize the number of registers that need to be saved.

- Thread Local Storage (TLS) buffer to ***eliminate contention***.

Blink: LSEO

- This endeavour started with validating the LSEO optimization
- **Blink validates the developer's hypothesis**:

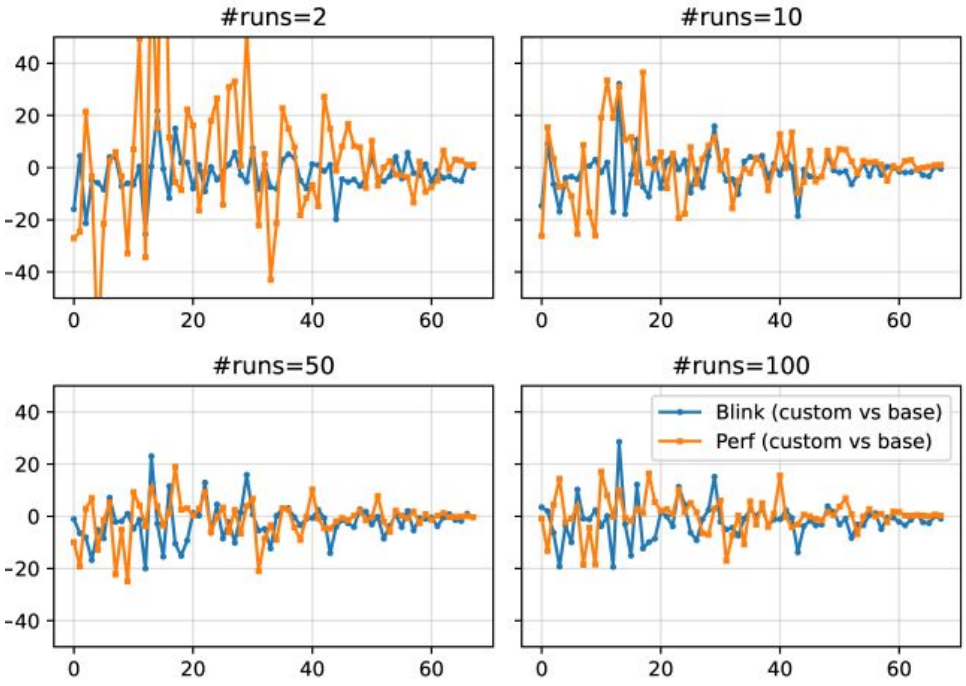
  LSEO optimization does reduce the dynamic instruction count

	Average Dynamic Instructions
Before LSEO	352.8
After LSEO	349.8

Average Dynamic Instruction Count of `update` function in `librender`

Blink: Auto-tuning

- Aim to train a ML model to automate compiler flag tuning decisions.
- Perf results have too much variance across runs/executions, making training difficult.
- Blink produces stable results after just 2 runs. Whereas, perf requires 50.



Related Work

1. Skid and shadow effects have been extensively studied
   - Chen et al. CGO'10; Dimakopoulou et al. SC'16; Banerjee et al. ASPLOS'21 etc.
   - But we describe real world significance of these issues.

2. XRay (*Berris et al.*) tracks execution time using compiler instrumentation.
   - Doesn't track all PMU events like Blink, adds 20%-40% overhead

3. Hubble (*Luo et al. OSDI'22*) traces Java/Kotlin function execution time in Android applications
   - Can't trace native functions, doesn't track all PMU events.
   - Always on, no self-patching

4. Project5, MagPie, X-Trace, Dapper, Fay, ÜberTrace, and Pivot Tracing profile distributed clusters rather than smartphone workloads.

Conclusion

- Perf cannot be trusted to measure flat-profile workloads.

- Blink is a lightweight and trustworthy alternative
  - Provides accurate and maximal coverage
  - Adds minimal overhead.

- Blink is now integrated in Huawei's testing suite and is used by compiler engineers for their performance optimization efforts.

Thank you for listening!

Blink is open sourced: https://github.com/Huawei-CPLLab/Blink

More details in our paper

- Investigating frame drops using Blink
- Integrating Blink into Huawei test suite.
- Measuring accuracy and overhead of Blink.

Questions?