Quantitative Evaluation of Intel PEBS Overhead for Online System-Noise Analysis

June 27, 2017, ROSS @ Washington, DC

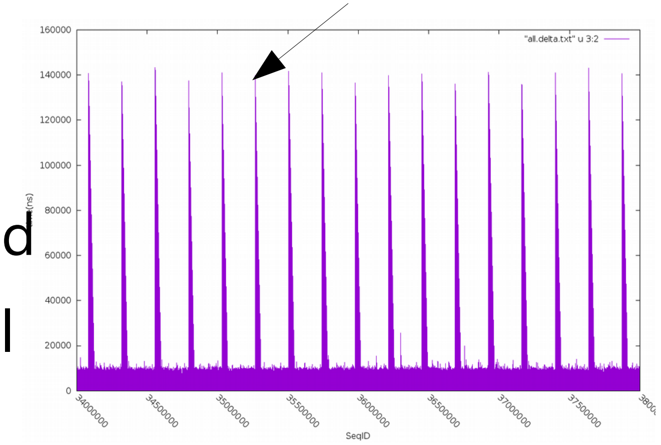
Soramichi Akiyama, Takahiro Hirofuchi

National Institute of Advanced Industrial Science and Technology (AIST), Japan

{s.akiyama, t.hirofuchi}@aist.go.jp

Perf Analysis of High Throughput Systems

- High throughput systems
  - Spark, RDBMS (Millions of transactions/s), …
  - Each data/message lives for a transient period
  - Performance fluctuation in the message-level



Latency Fluctuation

- Traditional performance analysis (e.g. gprof, vTune)
  - Function or code-block based (**func_A** takes most of the time)
  - Averaged profile across a whole run → Cannot catch fluctuations

- Message-level performance analysis needed
  - Profilers must distinguish each message (**message_X** takes longer time than other ones)

System Noise

- A factor of performance fluctuation stemming from the underlying system (HW, OS)
  - cache/TLB miss cost, context switching cost, scheduler, …
- Our focus:

Online analysis of system noise caused to high-throughput systems

- Examples
  - TCP packets with rare routes → extra cache/TLB misses (because the corresponding flow table rarely loaded)
  - Memory allocation from heap sometimes takes longer time than usual (because of fragmentation)

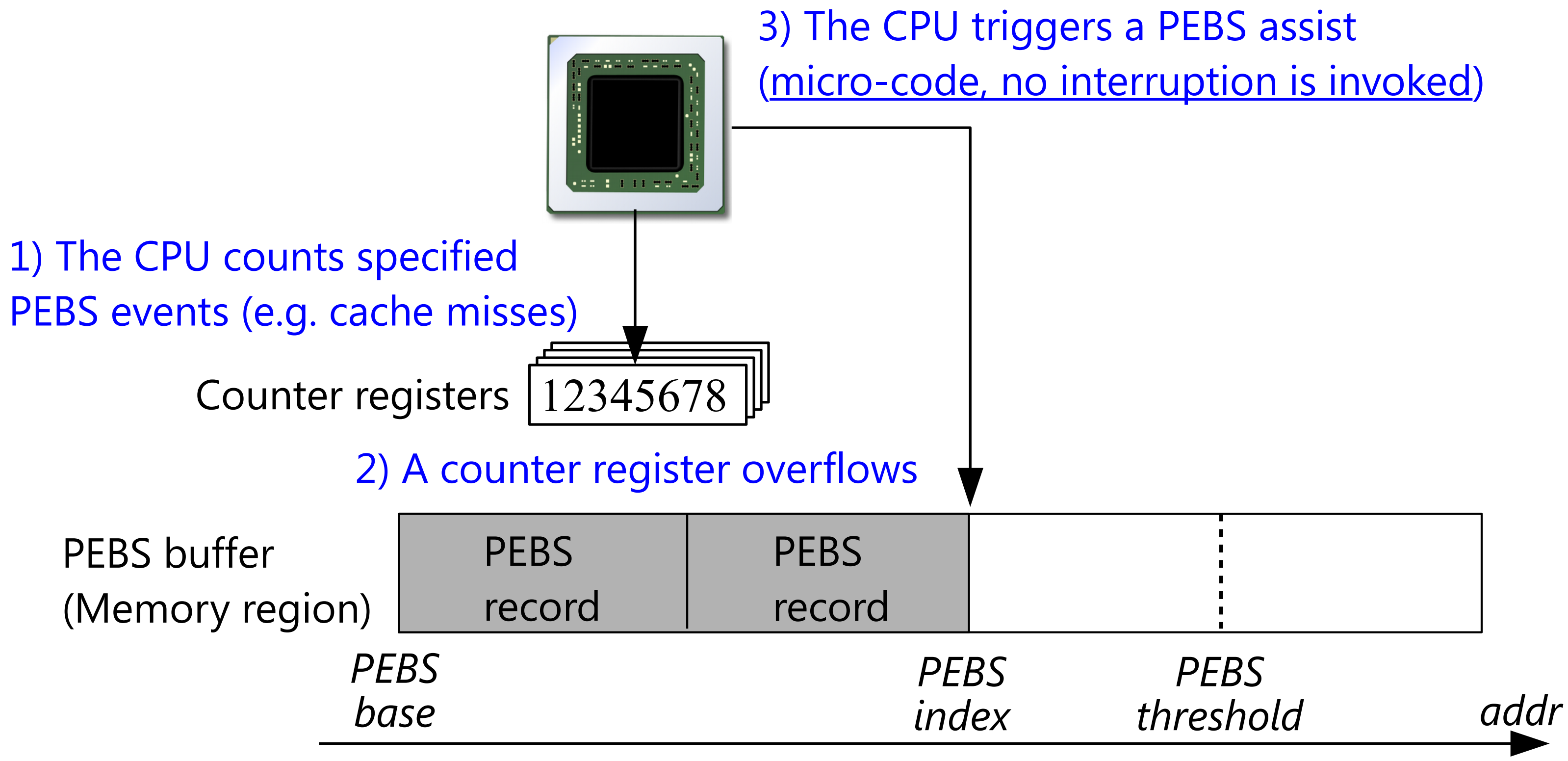
Existing Work for Message-level Profiling

- lprof [Zhao et al., OSDI'14]
  - Non-intrusive message level profiler from logs
  - **For each message**, outputs lists of timestamps of the message's arrival/retirement to/from methods
  - Cannot capture hardware events / kernel space activity
- Blocked time analysis [Ousterhout et al., NSDI'15]
  - Instrumentation-based perf analysis for Spark
  - **For each query**, analyze how long time the query is blocked
  - Cannot capture hardware events / kernel space activity

Need help of perf counters to capture HW events and kernel activities in the message-level

PEBS: How it works

- Precise Event-Based Sampling (PEBS): An extension of performance counters by Intel


3) The CPU triggers a PEBS assist
(micro-code, no interruption is invoked)
1) The CPU counts specified
PEBS events (e.g. cache misses)
Counter registers
12345678
2) A counter register overflows
PEBS buffer
(Memory region)
PEBS
record
PEBS
record
PEBS
base
PEBS
index
PEBS
threshold
addr

A PEBS record includes:
{ **General purpose registers (eax, ebx, ..., r14, r15), Instruction Pointer (IP), HW timestamp (tsc)**, Data LA, Load Latency, TX abort reason flag }

PEBS vs. Normal Perf Counters

Normal Counters

Count by hardware, sample by software
(Ex. # of cache misses reaches to 100K → OS receives an interruption to collect a sample)

- Frequent sampling → many interruptions

- Non-negligible time gap between an event occurrence and the corresponding sample (sampled IP may be biased)

PEBS (Precise Event Based Sampling)

Count and sample by hardware
(Ex. # of cache misses reaches to 100K → CPU automatically saves a sample)

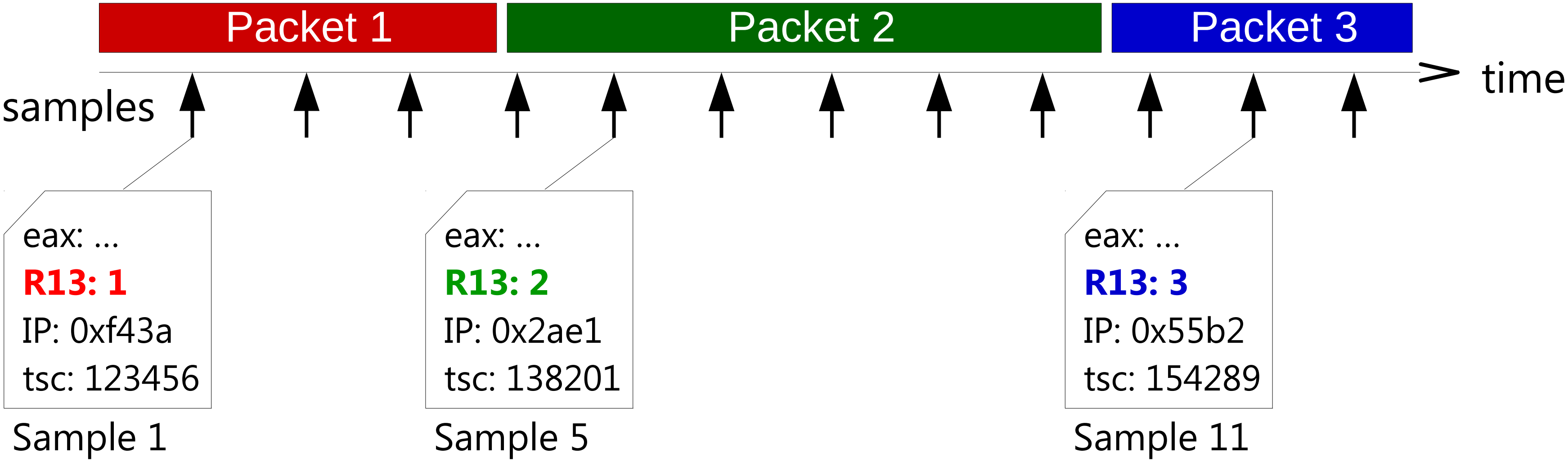
- Orders of magnitude smaller # of interruptions → smaller overhead

- Much more precise than normal performance counters

PEBS (small overhead, precise timing) is promising for message-level system noise analysis

System Noise Analysis w/ PEBS (1/2)

- Example for a DPDK-based network latency injector (*)
- Reserve one general purpose register (e.g. **r13**)
  - `gcc -ffixed-r13` → the code compiles without using **r13**
- Store packet ID to **r13** and sample general purpose regs, instruction pointer, tsc w/ PEBS



Sample 1:

```
eax: ...
R13: 1
IP: 0xf43a
tsc: 123456
```

Sample 5:

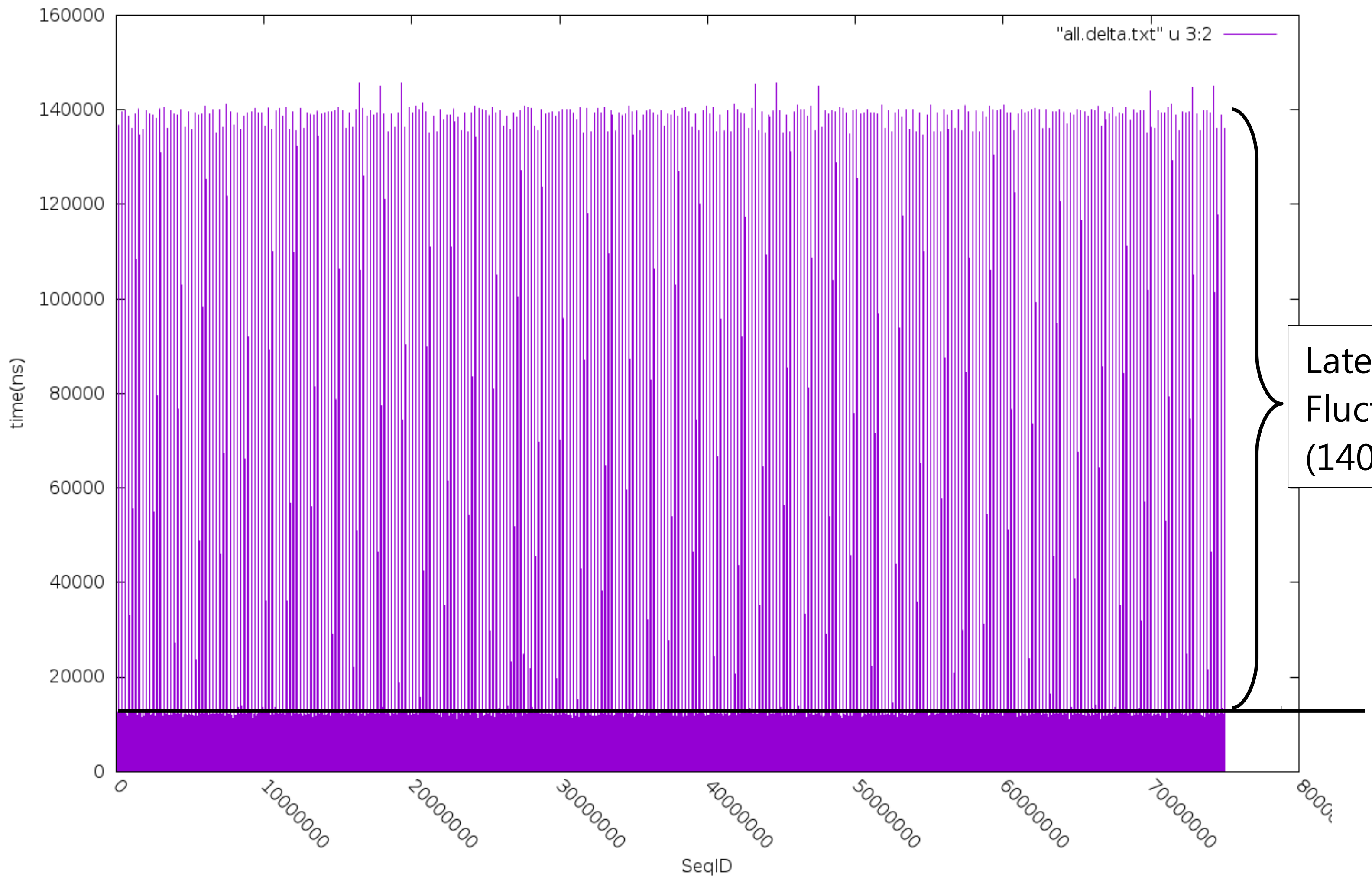
```
eax: ...
R13: 2
IP: 0x2ae1
tsc: 138201
```

Sample 11:

```
eax: ...
R13: 3
IP: 0x55b2
tsc: 154289
```

(*) Aketa et al., “DEMU: A DPDK-based Network Latency Emulator”, LANMAN 2017

System Noise Analysis w/ PEBS (2/2)

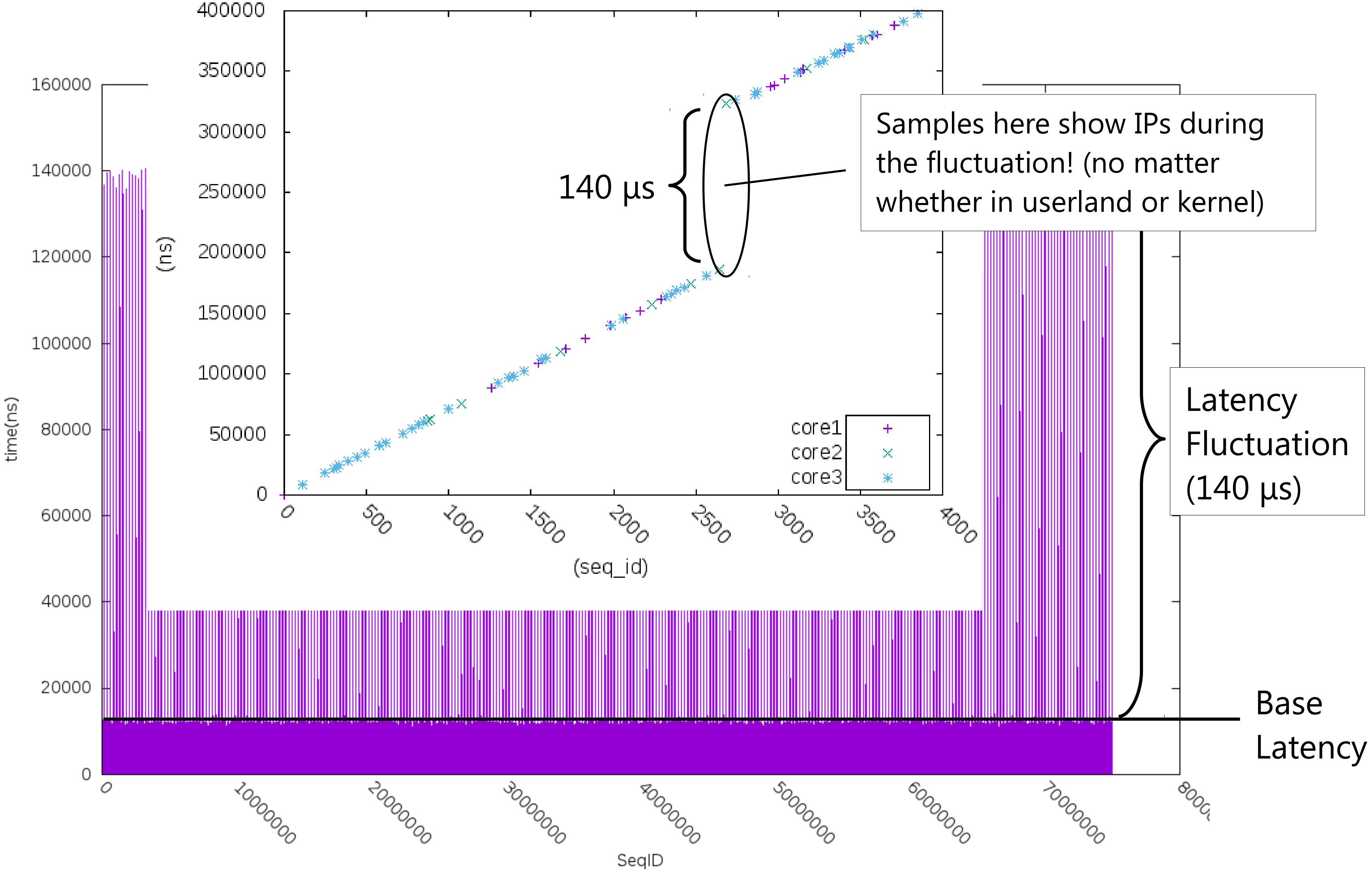


Latency Fluctuation (140 μs)

Base Latency

System Noise Analysis w/ PEBS (2/2)

Packet ID (r13) vs. tsc (converted to wallclock) sampled by PEBS



Samples here show IPs during the fluctuation! (no matter whether in userland or kernel)

Latency Fluctuation (140 μs)

Base Latency

Overhead of PEBS: Why we care

- Sampling rate should be very higher than normal usage
  - To distinguish each data/packet/message
  - No study has never been done for this high sampling rate

- Performance anomalies are difficult to reproduce offline
  - We need to apply PEBS to real running systems
  - Need to predict how much overhead PEBS incurs

We thoroughly investigate PEBS overhead in this paper

Overhead of PEBS: Overview

- A wide-spread myth:

  “PEBS incurs no overhead because it is hardware-based”

- The reality:
  - **Non-negligible CPU overhead and cache pollution**
  - Because PEBS is a micro-code, executed on the same resources (e.g. retirement ports) as normal operations

- This paper answers two question:
  - How much is the overhead?
  - How to configure PEBS to cope with the overhead?

PEBS Configuration vs. Overhead

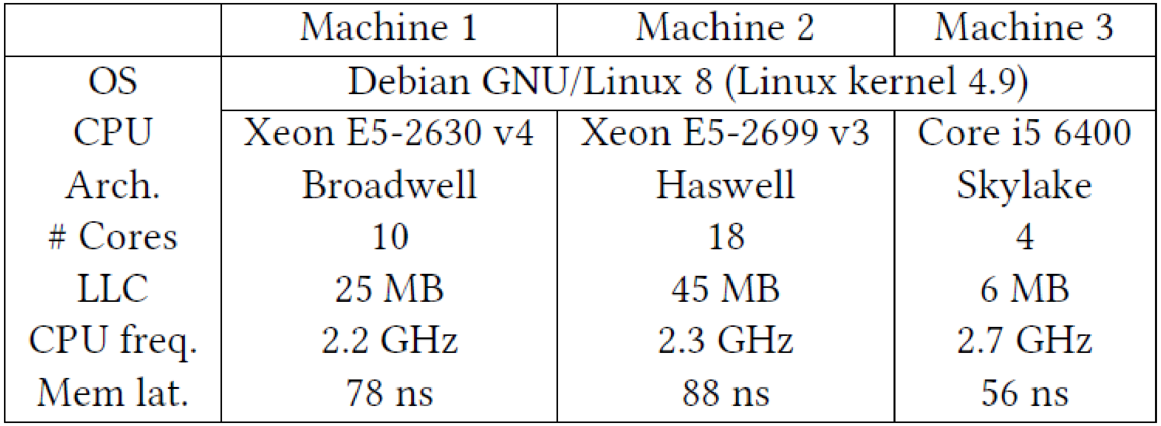
- Reset Value (*R*, a.k.a. Sample After Value)
  - A PEBS record is taken every *R* events → Decides the sampling rate
  - Ex. {*R* == 100, event == cache_misses} → A PEBS record is taken every 100 cache misses

- PEBS buffer size
  - Larger buffer incurs smaller number of interruptions
  - Larger buffer incurs more sever cache pollution
    - PEBS records written via CPU cache, not directly to the memory

  → Trade-off between # of interruptions and cache pollution

Evaluation Setup

- A simple kernel module
  - Configures PEBS (event, reset value, PEBS buffer size)
  - Counts # of PEBS records at every interruption and discards them
- Why build a new module?
  - Existing tools (e.g. perf): too rich → non-negligible overhead (*)
- Evaluation Environment



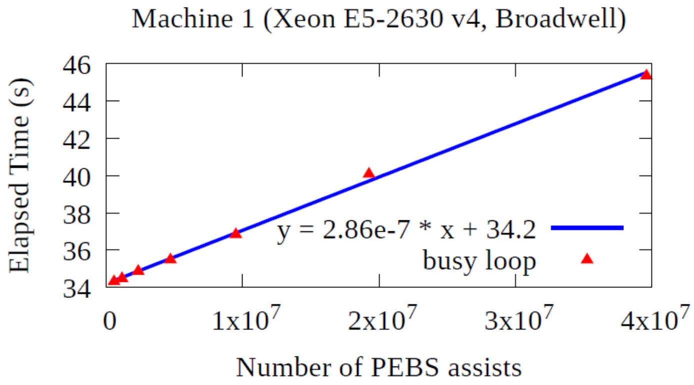
	Machine 1	Machine 2	Machine 3
OS	Debian GNU/Linux 8 (Linux kernel 4.9)		
CPU	Xeon E5-2630 v4	Xeon E5-2699 v3	Core i5 6400
Arch.	Broadwell	Haswell	Skylake
# Cores	10	18	4
LLC	25 MB	45 MB	6 MB
CPU freq.	2.2 GHz	2.3 GHz	2.7 GHz
Mem lat.	78 ns	88 ns	56 ns

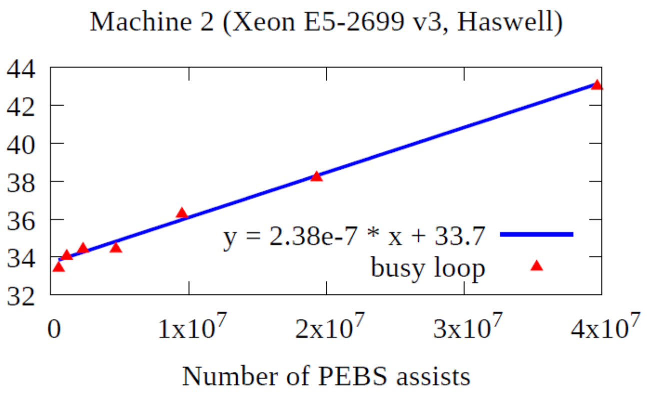
(*) Weaver, “Self-monitoring Overhead of the Linux perf_event performance counter interface”, ISPASS’15

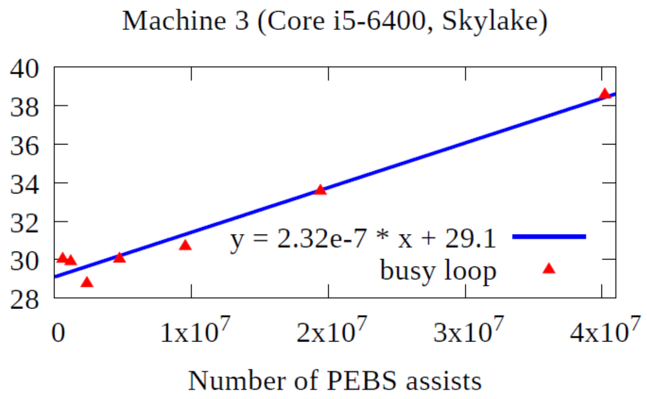
CPU Overhead per PEBS Assist

- **[Q1]** How much overhead does **one PEBS assist** have?
  - Compare elapsed time of pre-defined number of busy loops
  - For *R* = {2K, 4K, 8K, ..., 128K}, plot # of PEBS assists vs. elapsed time
  - PEBS event: UOPS_RETIRED.ALL (“All micro ops”), PEBS buffer: 4MB

- Results


Machine 1 (Xeon E5-2630 v4, Broadwell)


Machine 2 (Xeon E5-2699 v3, Haswell)

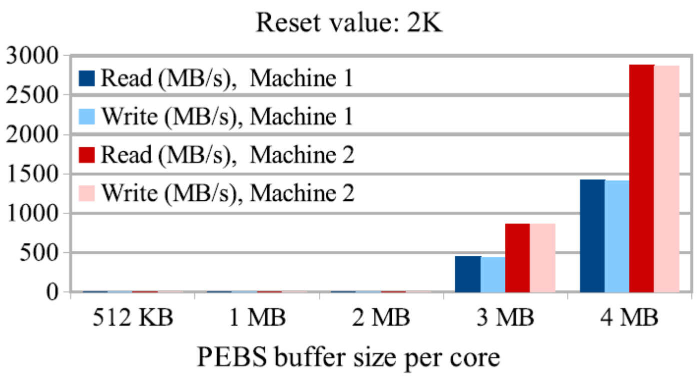

Machine 3 (Core i5-6400, Skylake)

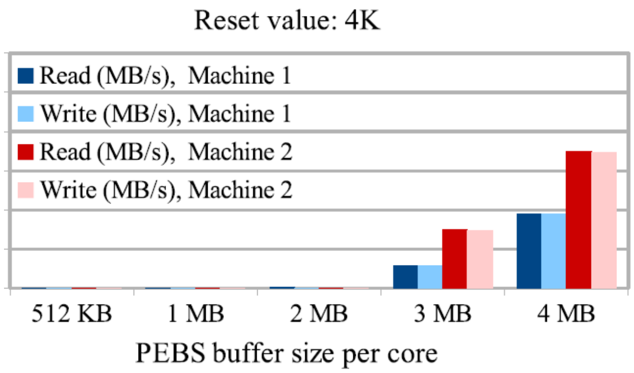
  - **Elapsed time grows linearly w.r.t # of PEBS asists**
  - Overhead per PEBS assits: **286ns**, **238ns**, **232ns** (slopes of the blue lines)

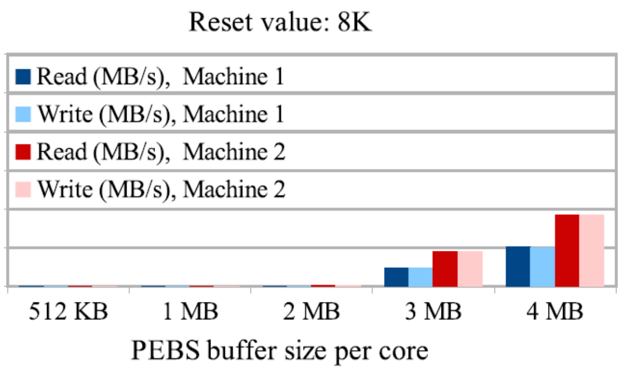
Memory IO by PEBS

- [Q2] How much memory IO does PEBS have?
  - Measure memory IO when PEBS is applied to busy loops
  - Memory IO measured from the memory controllers
  - Plot PEBS buffer size per core vs. Measured memory IO

- Results (Note: The counter available only in Xeon processors)





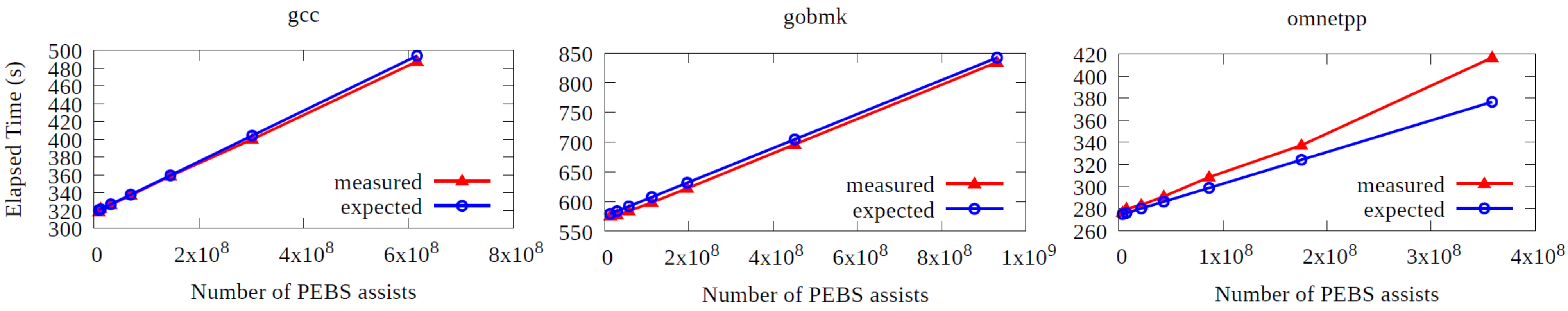


  - Prominent memory IO when PEBS buffer > 3MB/core (Recall: CPU cache of our machines is 2.5MB per core) → Reason: cache spill
  - PEBS data written via cache, which may degrade app performance

CPU Overhead on Real Workloads

- [Q3] Overhead per PEBS assist applicable to real workloads?
  - Predict the overhead caused to SPEC CPU 2006 benchmarks
  - Compare expected elapsed time and measured elapsed time

- Results (more on the paper)



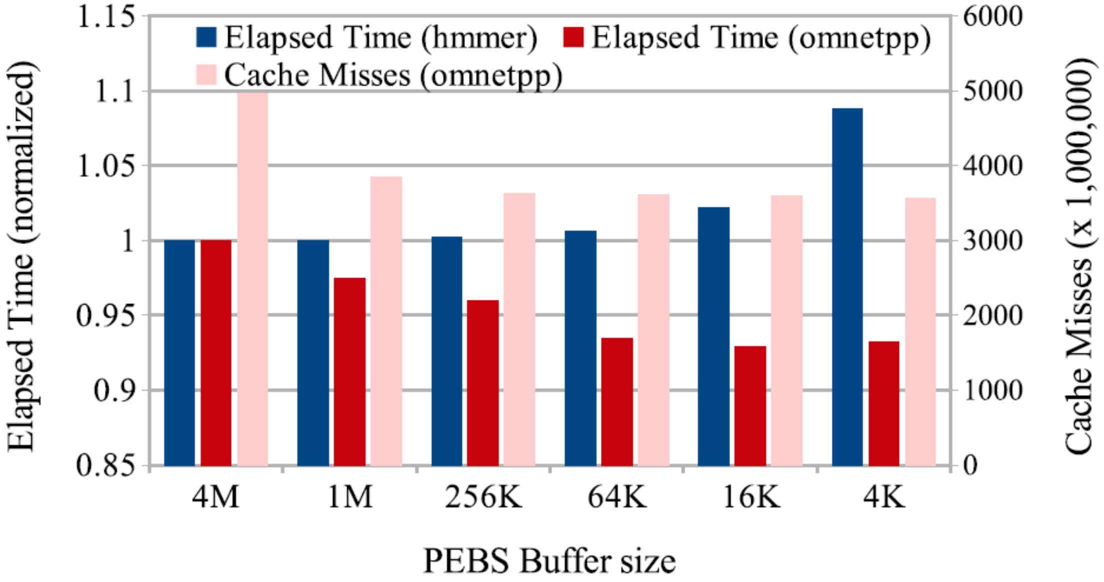
  - Expected time match measured time in 11 benchmarks (out of 12)

    → Overhead / PEBS assist applicable to predict elapsed time of real workloads with PEBS enabled (except some special cases)

PEBS buffer size vs. Cache pollution

- [Q4] How much the cache pollution affect the application performance?
  - Measure the effect of PEBS buffer size for omnetpp (cache-sensitive) and hmmer (cache-oblivious)
  - Larger PEBS buffer → Less interruptions, severer cache pollution

- Results



omnetpp: **Faster** as PEBS buffer gets smaller (thanks to less cache pollution)

hmmer: **Slower** as PEBS buffer gets smaller (due to more interruptions)

→ PEBS buffer size should be decided based on the workload characteristics

Lessons Learned and Future Work

- Overhead per sampling: 230~280ns (hopefully suffices for our ongoing analysis work)
  - Works well even for complex workloads
- PEBS buffer size must be carefully decided
  - Should always be less than the CPU cache size
  - Large PEBS buffer may degrade workload performance due to cache conflicts (e.g. omnetpp from SPEC CPU 2006)
- Future Work
  - Further investigation of the cache pollution
  - Real system noise analysis using PEBS