Precise Event Sampling on x86 architectures and its uses in profiling tools:

KOÇ ÜNİVERSİTESİ

Assoc. Prof. Didem Unat (dunat@ku.edu.tr)

in collaboration with Aditya Sasongko, Milind Chabbi, Paul Kelly

NANDA Workshop 5-6 Sept 2022

Precise Event Sampling

- **Hardware feature** in commodity CPUs
  - Extends Performance Monitoring Units (PMUs)
  - PMUs consist hardware counters + model-specific registers (MSRs)
- **sampling hardware or software events** periodically based on user-defined sampling period
- **attributing** those samples **accurately** to the **instructions** that trigger them.
- Examples: Intel PEBS, AMD IBS, PowerPC MRK, ARM SPE

Profiling Tools based on Event Sampling

ComDetective

- **"ComDetective: A Lightweight Communication Detection Tool for Threads**", IEEE/ACM Supercomputing Conference, **Best Student Paper and Best Paper finalist for SC19**.

ReuseTracker

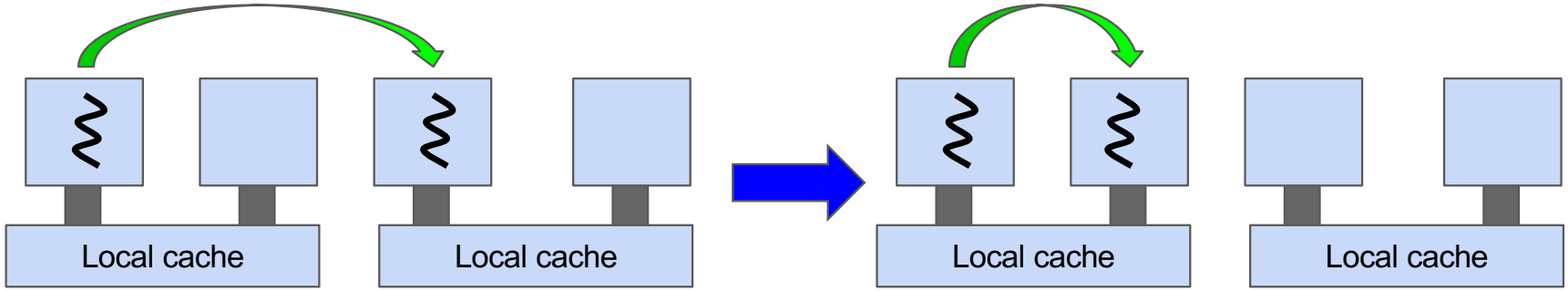
- "**ReuseTracker: Fast Yet Accurate Multicore Reuse Distance Analyzer**", ACM TACO and HiPEAC Conference, June 2022

AMD vs Intel

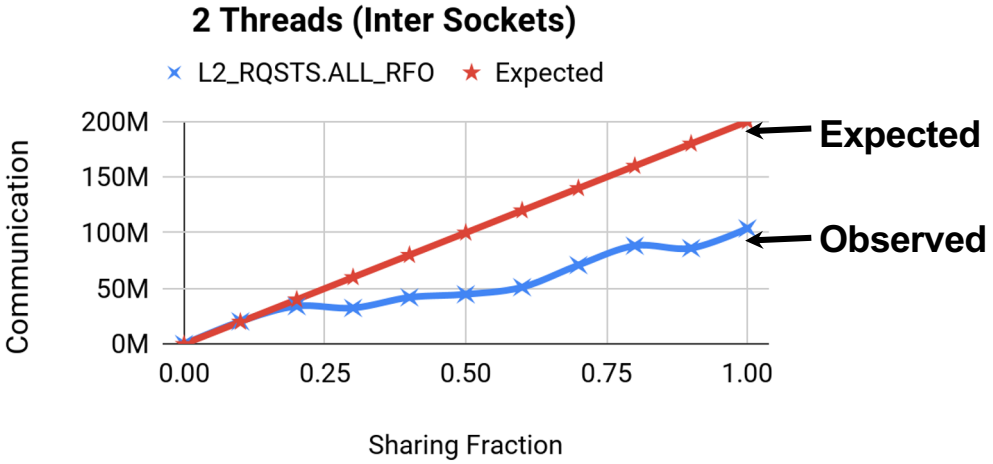
- *M. A. Sasongko, M. Chabbi, P. H. J. Kelly, D. Unat, under review at IEEE TPDS*

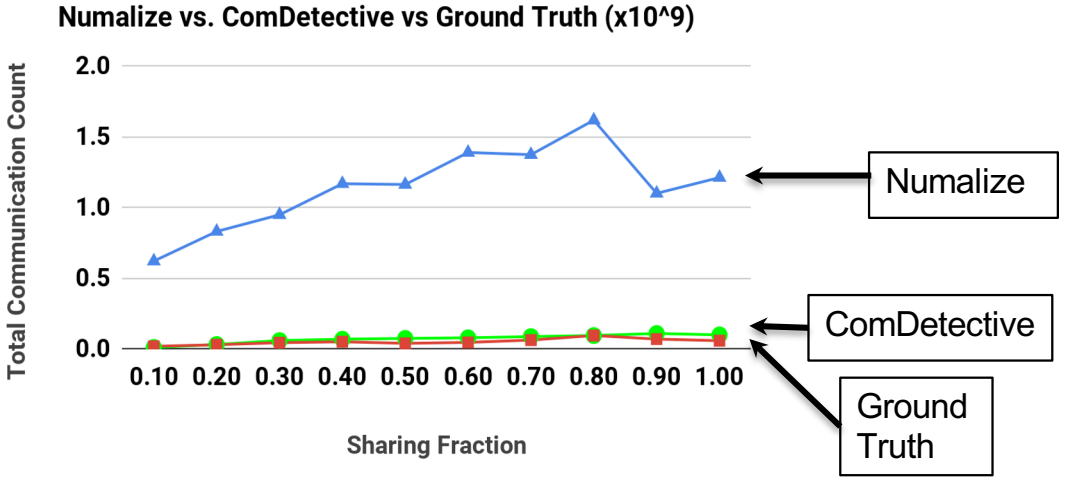
Why detect inter-thread communication?

- Identify possible sources of performance bottlenecks
- Help explain why one threading library is better than another
  - e.g. Intel OpenMP vs GNU OpenMP
- Guide performance optimizations such as
  - thread binding
  - data structure modification
  - false sharing elimination
- Hardware design: on-chip network design, cache coherence protocol



Problem with existing tools





Not accurate:

- Distortion of parallel schedule among threads by binary instrumentation (Pericas, et al. ICS 2014)

Too slow to use in practice

- Unable to capture actual communication pattern in real applications

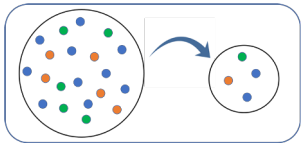
ComDetective

Accurate

Validated against several benchmarks and HPC applications

Lightweight

Space overhead (1.27x) and time (1.3x) overhead



Sampling based

Uses ready-available hardware events in commodity CPUs

Differentiates kind of communication

True sharing (necessary) vs. false sharing (unnecessary)

Data objects

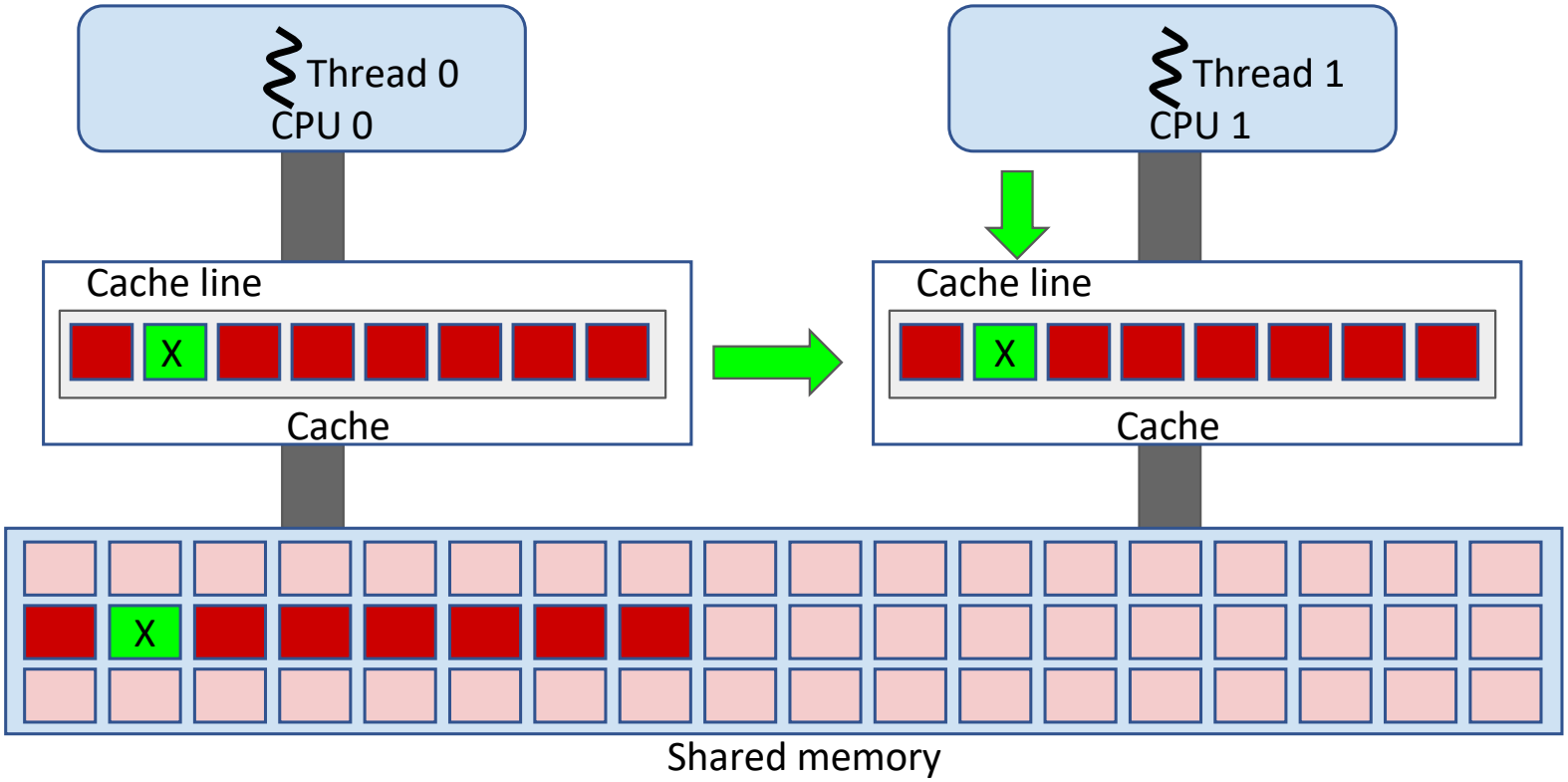
Attributes communication to program data objects

Open source (supports Intel and AMD x86 archs)

https://github.com/ParCoreLab/ParCoreTools

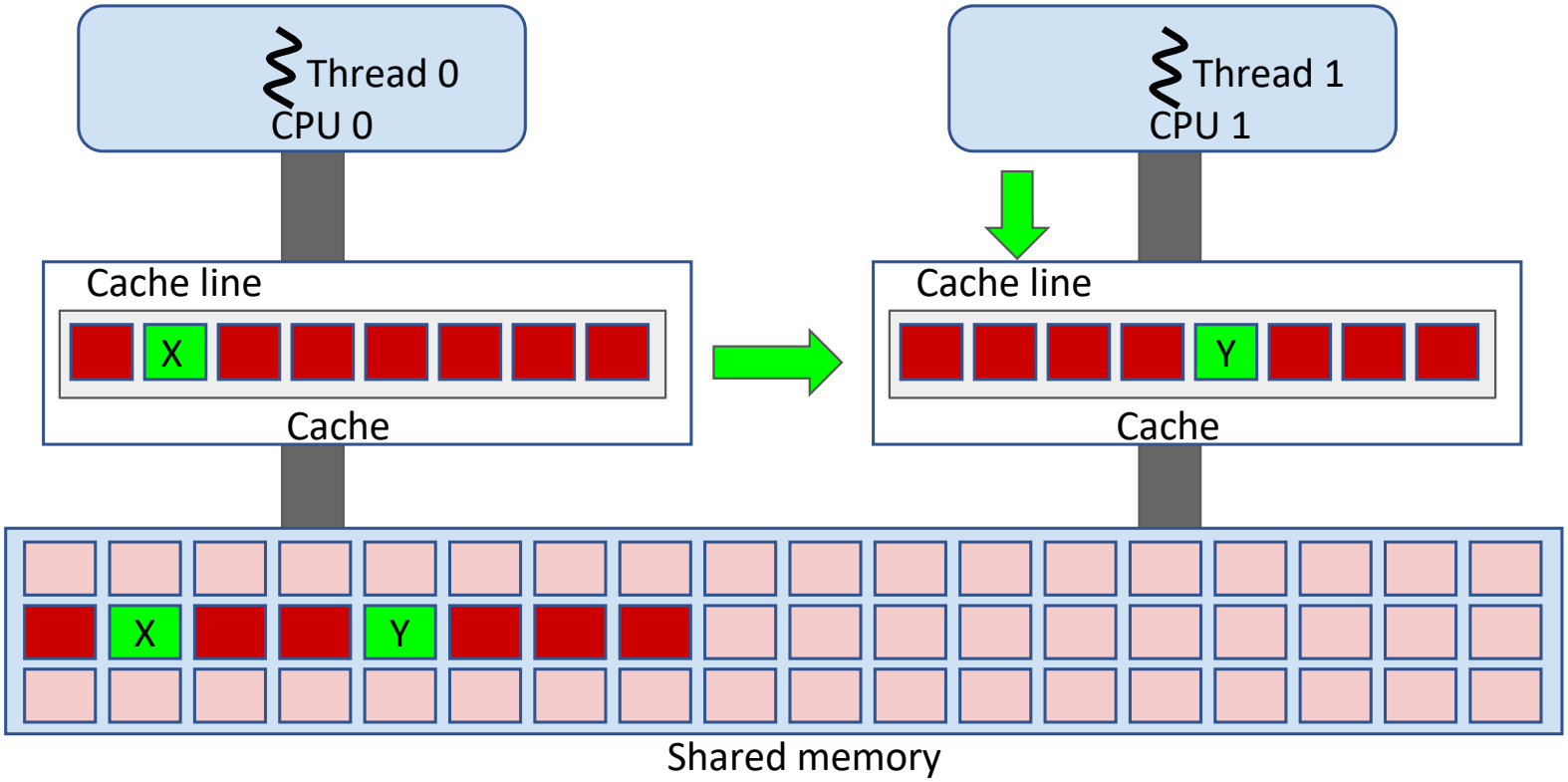
True Sharing

- This type of communication is called **true sharing**

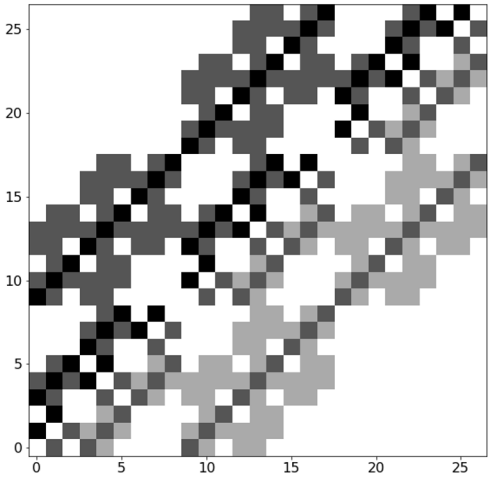


False Sharing

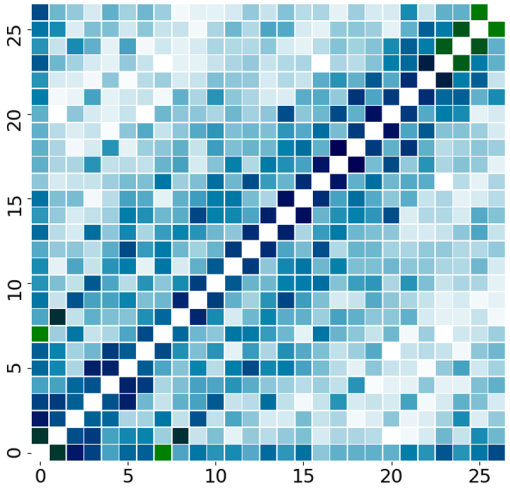
- Another possible type is **false sharing**
- Threads 0 and 1 access different memory regions in the same cache line



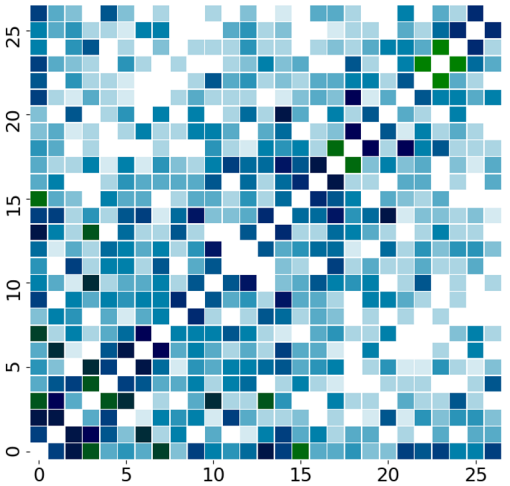
Communication Matrix



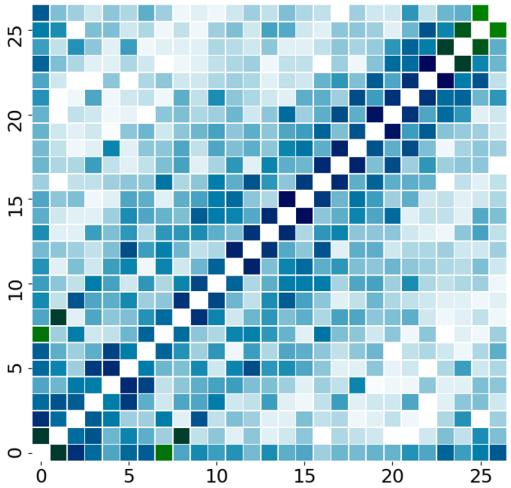
MPI communication matrix



communication matrix



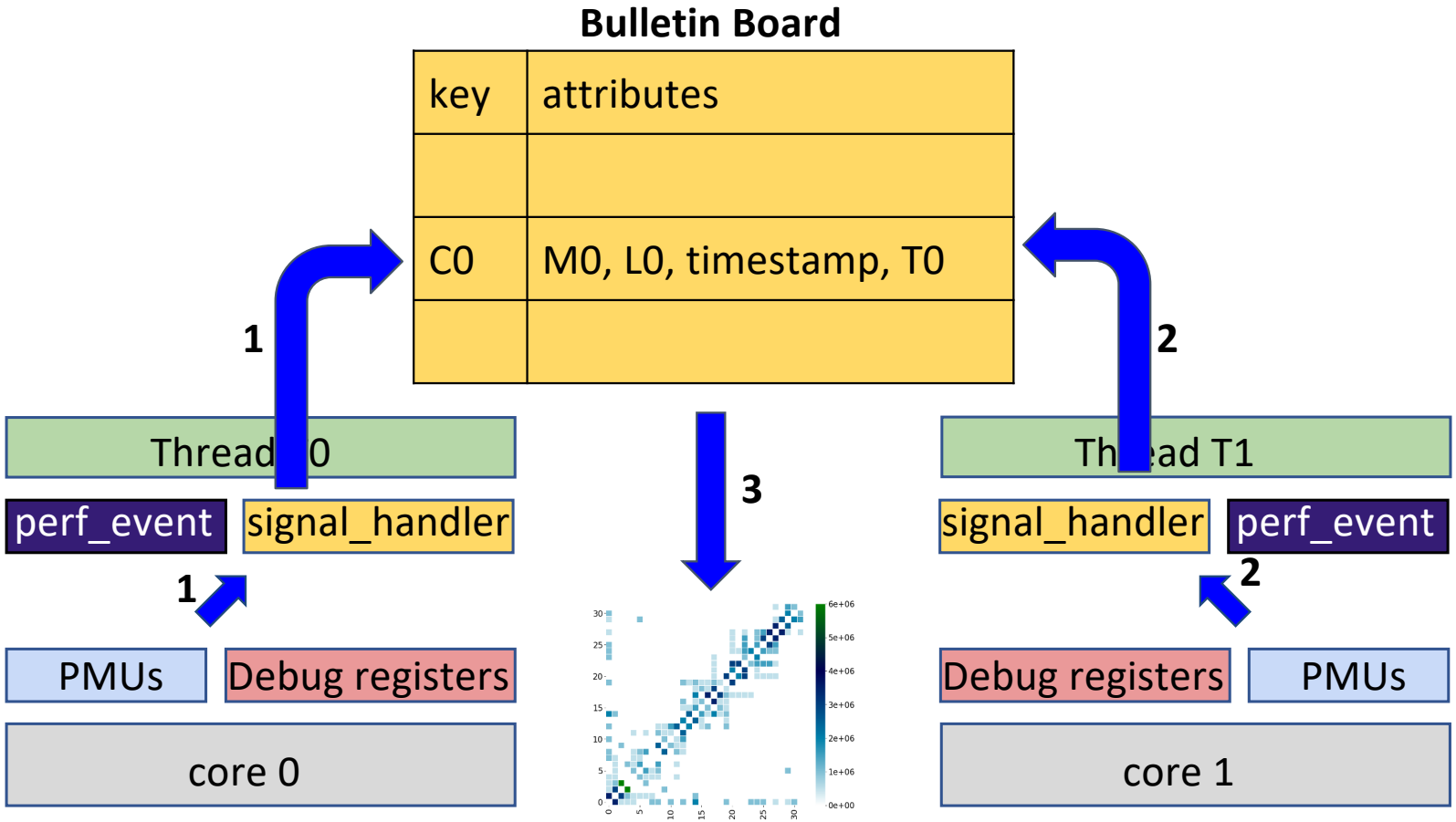
true sharing matrix



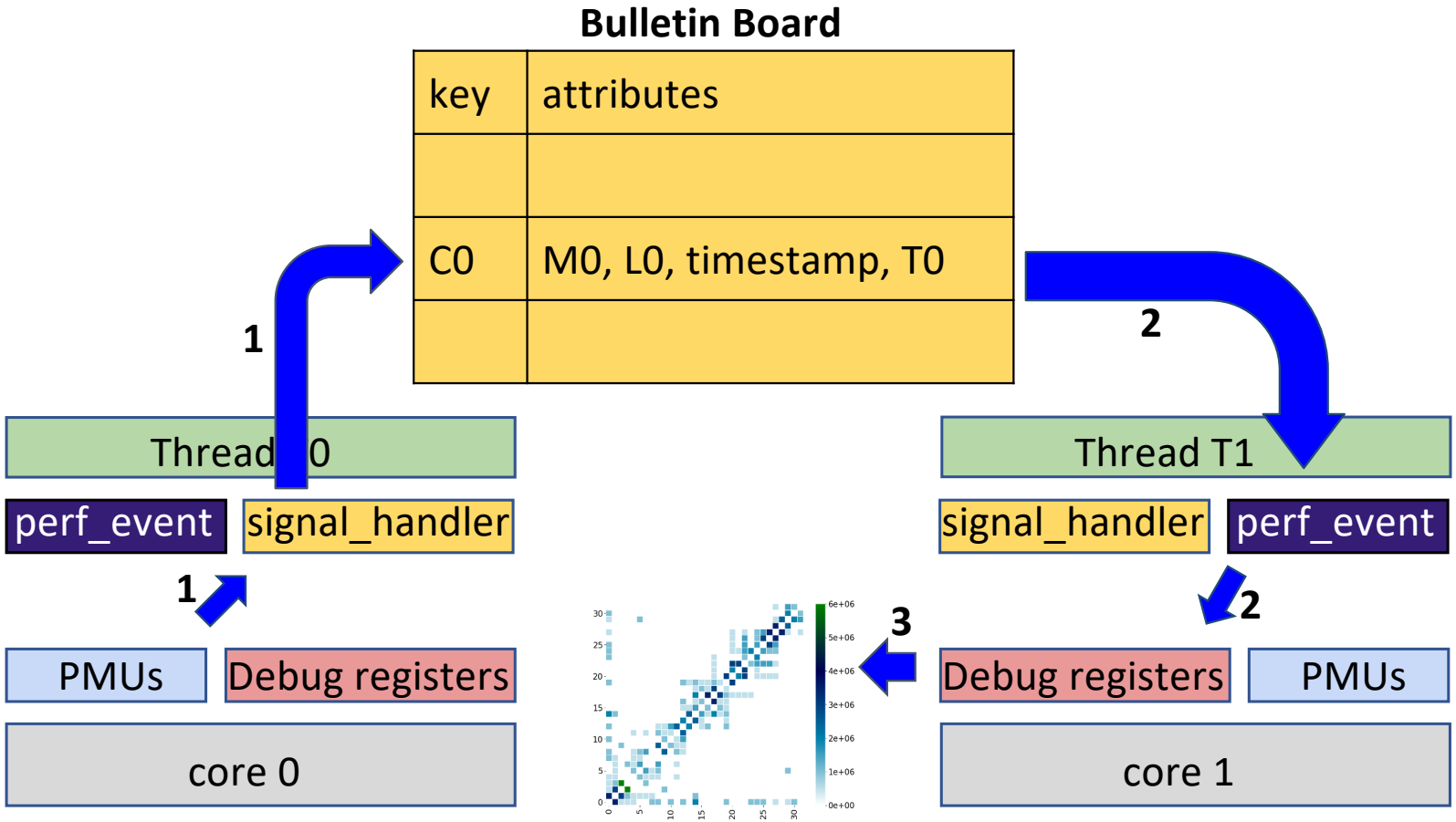
false sharing matrix

- In addition to all communication matrix, ComDetective also produces true sharing and false sharing matrices

Communication Detection

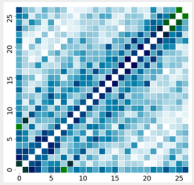


Communication Detection

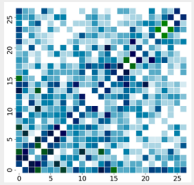


ComDetective

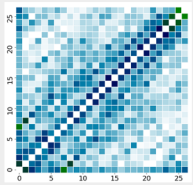
communication matrix



true sharing matrix



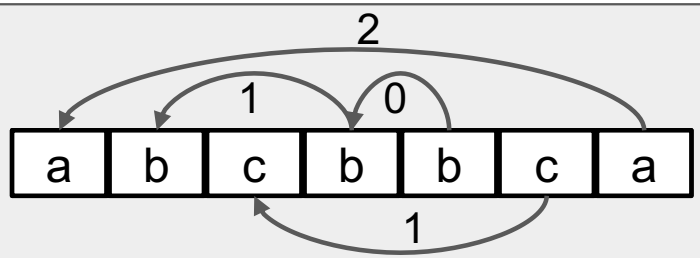
false sharing matrix



Inter-thread communication analyzer.
Published at Supercomputing'19.

The first lightweight tool that accurately detects inter-thread communications.

ReuseTracker



Reuse distance analyzer for multithreaded code.
Published at ACM TACO in 2022.

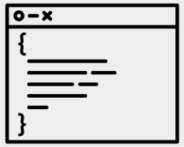
The first lightweight tool that accurately measures reuse distance in multicore.

Lightweight

Low runtime and memory overhead

Accurate

Validated against several benchmarks



Easy to Use

Attributes profiling data to source code lines

https://github.com/ParCoreLab/ParCoreTools

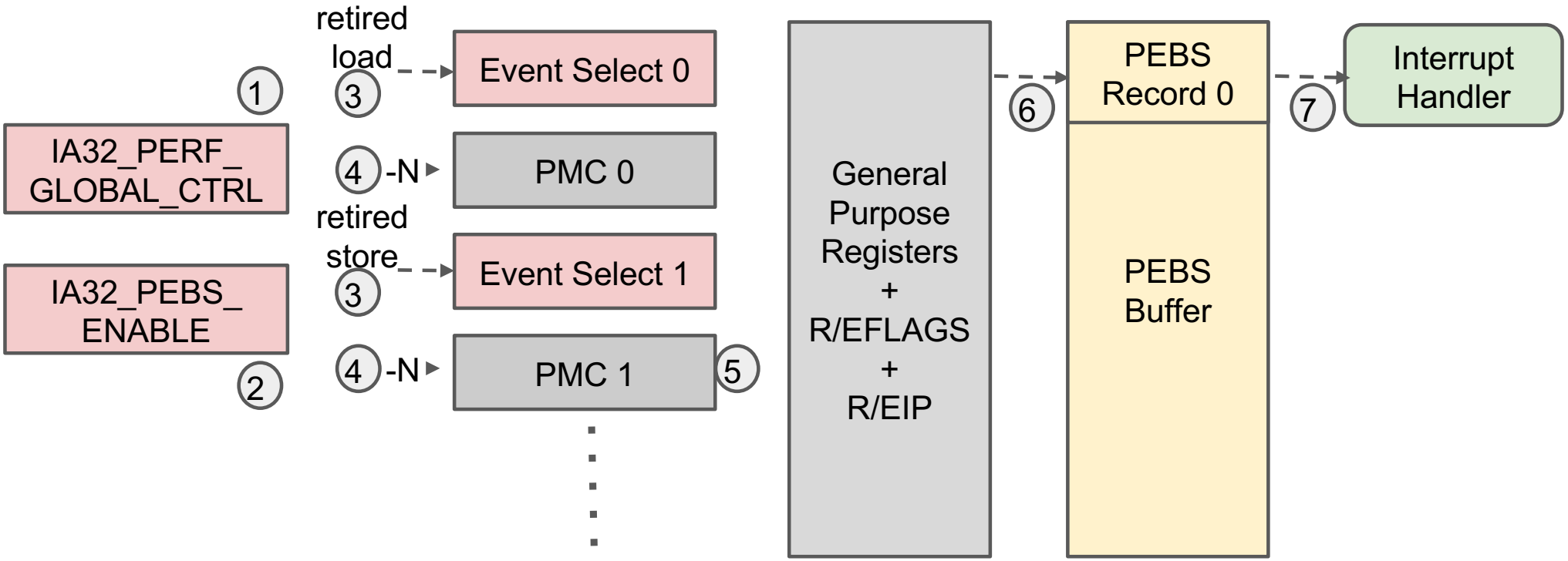
Why analyze precise-event sampling?

- To understand the behavior of precise event sampling-based profiling tools
- To gain insights useful for:
  - developers of new precise event sampling features, e.g. for RISC-V architecture
  - major vendors such as Intel, AMD, and ARM
- Questions unanswered by previous research:
  - Sampling bias, memory overhead, accuracy of instruction attribution
  - Stability of accuracy, accuracy in multiple events monitoring
  - Almost no study on AMD precise event sampling
  - The coverage of studies on Intel PEBS has been very limited.

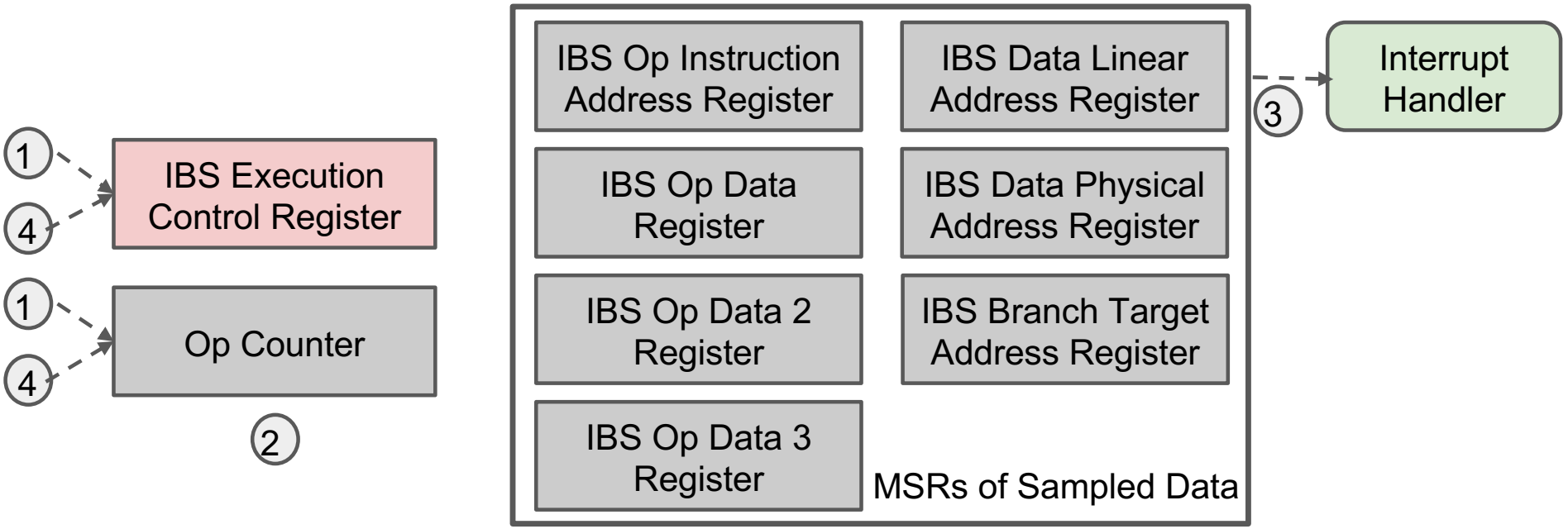
Contributions

- **In-depth analysis on precise event sampling** features of Intel and AMD x86 architectures
  - Quantitative and Qualitative comparison
- **Benchmark suites** that evaluate various aspects of precise event sampling facilities
  - Experiment 1: accuracy
  - Experiment 2: sensitivity to sampling interval and stability
  - Experiment 3: sampling bias and instruction attribution
  - Experiment 4: time overhead
  - Experiment 5: memory overhead
  - Experiment 6: multiple event monitoring
  - Experiment 7: kernel mode vs user mode detection

Intel PEBS


retired load
1
3
IA32_PERF_ GLOBAL_CTRL
Event Select 0
4 -N
PMC 0
retired store
IA32_PEBS_ ENABLE
3
Event Select 1
2
4 -N
PMC 1
5
General Purpose Registers + R/EFLAGS + R/EIP
6
PEBS Record 0
PEBS Buffer
7
Interrupt Handler

AMD IBS


1
4
IBS Execution Control Register
1
4
Op Counter
2
IBS Op Instruction Address Register
IBS Data Linear Address Register
IBS Op Data Register
IBS Data Physical Address Register
IBS Op Data 2 Register
IBS Branch Target Address Register
IBS Op Data 3 Register
MSRs of Sampled Data
3
Interrupt Handler

ARM SPE

- The facility in ARM, i.e. Statistical Profiling Extension (SPE), has the same sampling approach as IBS.
  - Its counter counts dispatched micro-operations, and therefore, facilitates sampling from this event.
- Our preliminary study on a high-end state-of-the art ARM processor
  - showed that its precise event sampling support is immature and cannot be directly comparable against PEBS or IBS.

Qualitative Differences

	Aspect	Intel PEBS	AMD IBS
Observation 1	Hardware counter	Shares the same counters with other non-precise events, might lead to event multiplexing	Has its own counters
Observation 2	Hardware counter	Counter is not randomized after each sample	If micro-op is monitored, counter is randomized after each sample
Observation 3	Sampled event	Has many different events to select from	Can only monitor 2 events: instruction fetch and micro-op
Observation 4	Sampled data	Might have to monitor multiple events to reach the same level of info as IBS	Offers richer set of data in each sample
Observation 5	Execution mode	Can count event in user mode, in kernel mode, or in any mode	Can only count event without discriminating execution mode

Quantitative Experiments

- **Experiment 1: accuracy**
- Experiment 2: sensitivity to sampling interval and stability
- Experiment 3: sampling bias and instruction attribution
- Experiment 4: time overhead
- Experiment 5: memory overhead
- Experiment 6: multiple event monitoring
- Experiment 7: kernel mo

Specification	AMD EPYC	Intel Xeon Gold
CPU Model	7352 CPU	6258R CPU
Microarch Family	Zen 2 (17h)	Cascade Lake
#Cores x Socket	24 x 2	28 x 2
Cache Sizes	L1i: 32KB, L1d: 32KB, L2: 512KB, L3: 16MB	L1i: 32KB, L1d: 32KB, L2: 1MB, L3: 39MB
Linux Kernel Version	Linux 5.11.0-36	Linux 5.11.0-36

Accuracy Evaluation

Underlying observations:

- Observation 2: The counter of IBS op is randomized after interrupt handling.
- Observation 3: Intel PEBS can target more specific hardware events, while AMD IBS can do only fetch or op sampling.

Hypothesis:

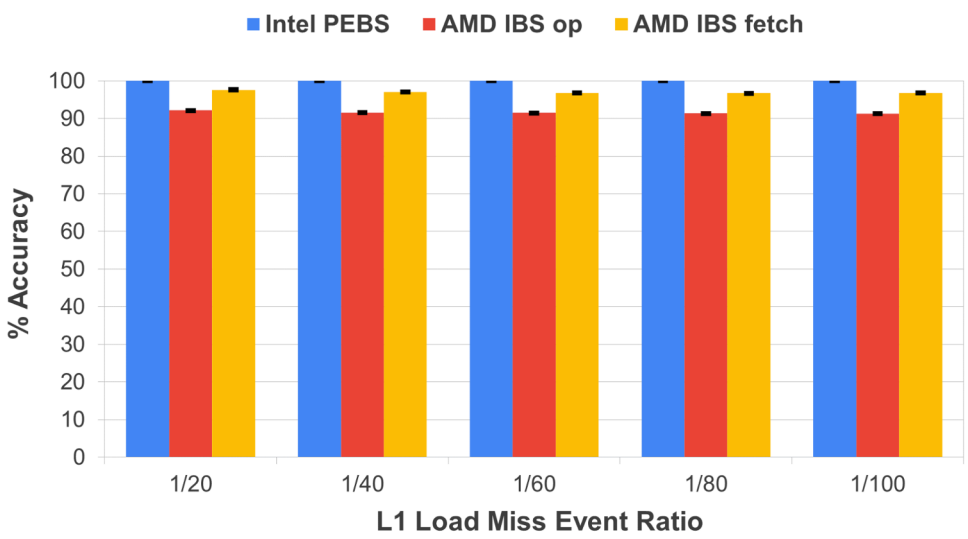
- Based on Observations 2 and 3, we expect PEBS to have better accuracy than both sampling flavors of IBS in capturing samples from any hardware event.

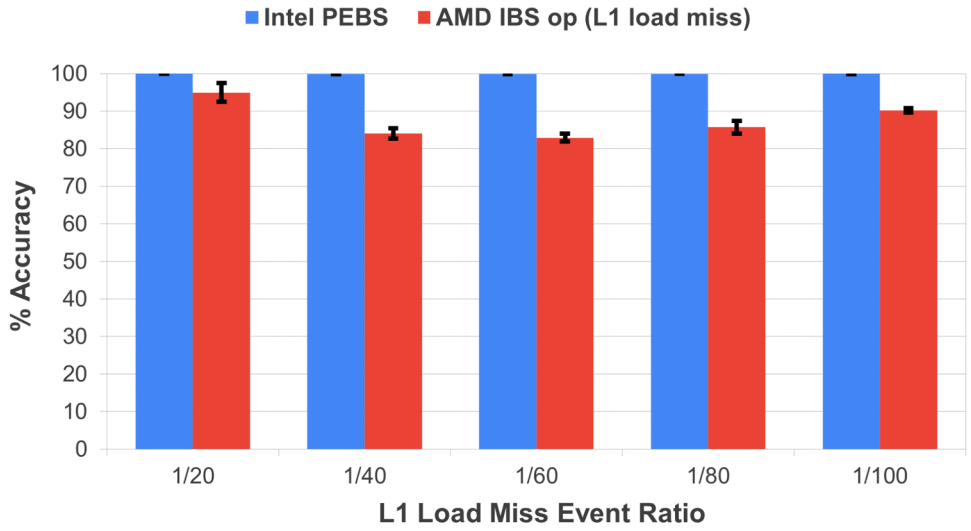
Accuracy Evaluation

Methodology of verification:

- Using a microbenchmark, called *Accuracy-Bench* benchmark, that has known L1 data cache miss count and configurable count of other instructions
  - Ground truth of instruction sampling: #instruction samples = #instruction count/sampling interval
  - Ground truth of data cache miss sampling: #L1 data cache miss samples = #L1 data cache miss count/sampling interval
- Programming PEBS and IBS to monitor retired instructions, instruction fetches, micro-operations, and retired load that misses in L1 data cache of the microbenchmark

Accuracy Results





Findings:

- PEBS shows high accuracy in sampling any event.
- IBS shows high accuracy only in sampling its most general events, i.e. instruction fetch and executed micro-operation.

Quantitative Experiments

- Experiment 1: accuracy
- **Experiment 2: sensitivity to sampling interval and stability**
- Experiment 3: sampling bias and instruction attribution
- Experiment 4: time overhead
- Experiment 5: memory overhead
- Experiment 6: multiple event monitoring
- Experiment 7: kernel mode vs user mode detection

Sensitivity to Sampling Interval and Stability

Underlying observations:

- Observation 2: The counter of IBS op is randomized after interrupt handling.
- Observation 3: Intel PEBS can target more specific hardware events, while AMD IBS can do only fetch or op sampling.

Hypothesis:

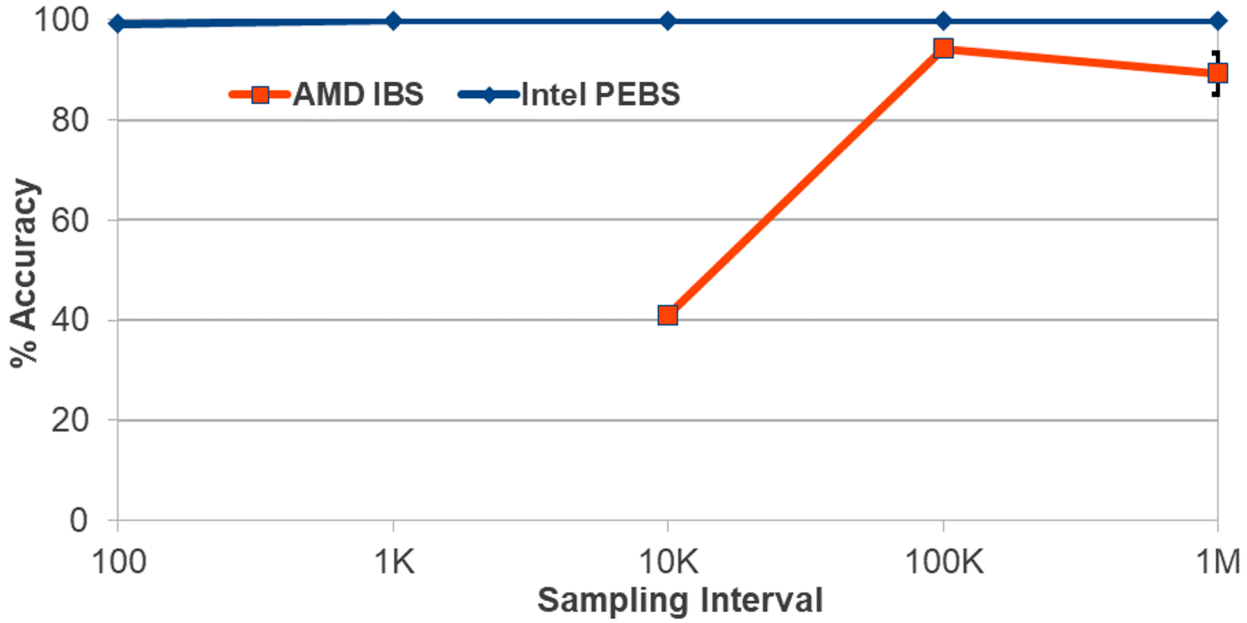
- Based on Observations 2 and 3, we expect PEBS to have higher accuracy than IBS under any sampling interval.

Sensitivity to Sampling Interval and Stability

Methodology of verification:

- Running the *Accuracy-Bench* benchmark under different sampling intervals
- Using the standard deviation of sample counts detected across multiple runs of *Accuracy-Bench* to measure stability

Sensitivity to Sampling Interval and Stability



Findings:

- PEBS shows high accuracy under any sampling interval, while IBS displays high accuracy only under large sampling intervals.
- PEBS shows high stability in sampling all tested events across multiple runs, while IBS displays lower stability in sampling subset events, e.g. L1 load miss.

Quantitative Experiments

- Experiment 1: accuracy
- Experiment 2: sensitivity to sampling interval and stability
- **Experiment 3: sampling bias and instruction attribution**
- Experiment 4: time overhead
- Experiment 5: memory overhead
- Experiment 6: multiple event monitoring
- Experiment 7: kernel mode vs user mode detection

Sampling Bias and Instruction Attribution

Hypothesis:

- We expect PEBS and IBS to have no bias in sampling from multiple different instructions that perform the same monitored event.
- We also expect PEBS and IBS to attribute the sampled events to the actual instructions that trigger those events.

Sampling Bias and Instruction Attribution

Methodology of verification:

- Using a microbenchmark, called *Bias-Bench* benchmark, that has 4 load instructions in each loop iteration
  - Ground truth of sampling bias: 25% of load samples are attributed to each load instruction
  - Ground truth of instruction attribution: none of the load samples is attributed to non-load instructions

Bias-Bench Microbenchmark

- Bias-Bench with 4 load instructions in each loop iteration:

```
movq $10000000000, %rcx
movl $1, %ebx
loop0:
movl (%rax), %ebx // load 1
movl (%rax), %ebx // load 2
movl (%rax), %ebx // load 3
movl (%rax), %ebx // load 4
subq $1, %rcx // subq
cmpq $0, %rcx // cmpq
jne loop0 // jne
```

Sampling Bias and Instruction Attribution

- Distribution of samples:

Instruction	Expected	Intel PEBS load	AMD IBS op
load 1	25%	58.6%	0%
load 2	25%	9.04%	3.58%
load 3	25%	18.34%	62.14%
load 4	25%	14.02%	4.64%
subq	0%	0%	29.62%

Findings:

- Both PEBS and IBS are equally biased in sampling from multiple instructions.
- PEBS records the addresses of the instructions that trigger the sampled events, while IBS op records the addresses of the instructions that execute after the triggering instructions.

Quantitative Experiments

- Experiment 1: accuracy
- Experiment 2: sensitivity to sampling interval and stability
- Experiment 3: sampling bias and instruction attribution
- **Experiment 4: time overhead**
- Experiment 5: memory overhead
- Experiment 6: multiple event monitoring
- Experiment 7: kernel mode vs user mode detection

Time Overhead

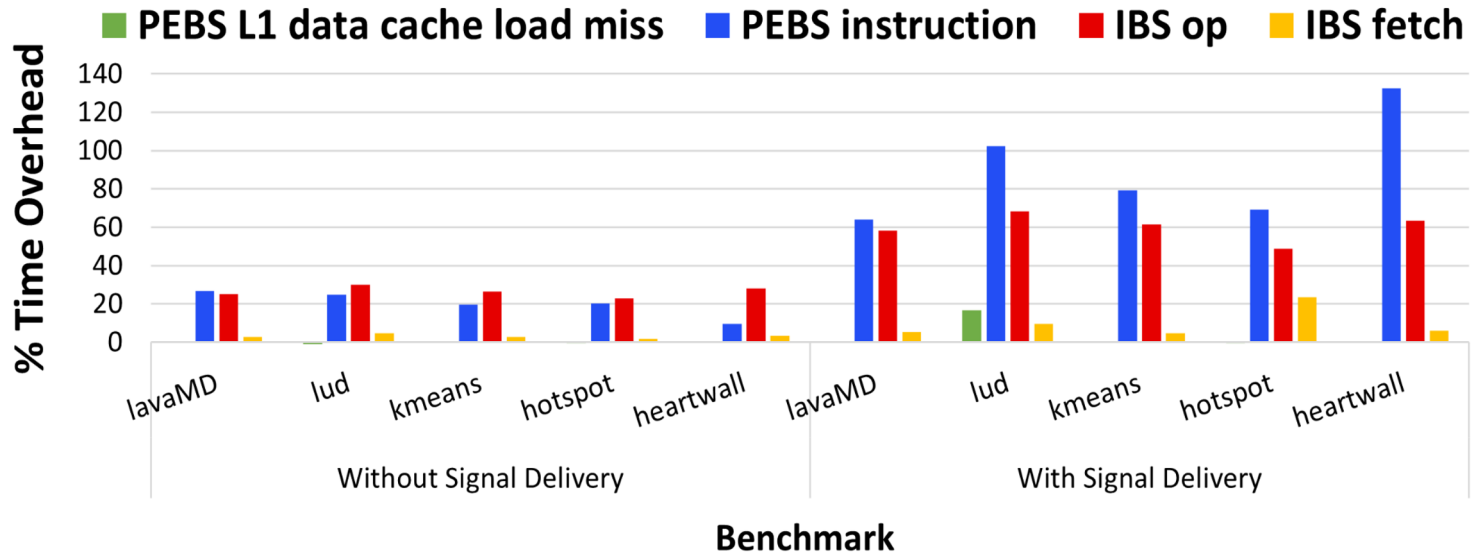
Underlying observations:

- Observation 3: Intel PEBS can target more specific hardware events, while AMD IBS can do only fetch or op sampling.

Hypothesis:

- Based on Observation 3, we expect PEBS that monitors a specific hardware event, e.g., L1 data cache load miss, to incur lower overhead than IBS as it will most likely encounter fewer sampling interrupts than IBS.

Time Overhead



Findings:

- PEBS incurs slightly lower time overhead on Rodinia benchmarks without OS signal delivery. -> PEBS hardware and microcode work more efficiently in recording data
- Both PEBS and IBS have significantly higher time overhead when OS signal delivery is enabled.
  - Time overhead of PEBS is much higher when OS signal is enabled.

Quantitative Experiments

- Experiment 1: accuracy
- Experiment 2: sensitivity to sampling interval and stability
- Experiment 3: sampling bias and instruction attribution
- Experiment 4: time overhead
- **Experiment 5: memory overhead**
- Experiment 6: multiple event monitoring
- Experiment 7: kernel mode vs user mode detection

Memory Overhead

Hypothesis:

- We expect both PEBS and IBS to have low memory overheads as we do not process sampled data in any signal handler.

Methodology of verification:

- Measuring memory overhead in terms of maximum resident set size in main memory with and without OS signal delivery enabled
- Collecting the memory consumption data together with the runtime data from *Accuracy-Bench* and Rodinia benchmarks

Findings:

- The measured memory overheads are all less than 0.3%.

Quantitative Experiments

- Experiment 1: accuracy
- Experiment 2: sensitivity to sampling interval and stability
- Experiment 3: sampling bias and instruction attribution
- Experiment 4: time overhead
- Experiment 5: memory overhead
- **Experiment 6: multiple event monitoring**
- Experiment 7: kernel mode vs user mode detection

Multiple Event Monitoring

Motivation:

- Observation 4: PEBS might have to monitor multiple events simultaneously to produce the same level of info as IBS.

Underlying observations:

- Observation 1: PEBS shares the same counters with other non-precise PMU events, while IBS has its own counters.
- Observation 2: The counter of IBS op is randomized after interrupt handling.
- Observation 3: Intel PEBS can target more specific hardware events, while AMD IBS can do only fetch or op sampling.

Multiple Event Monitoring

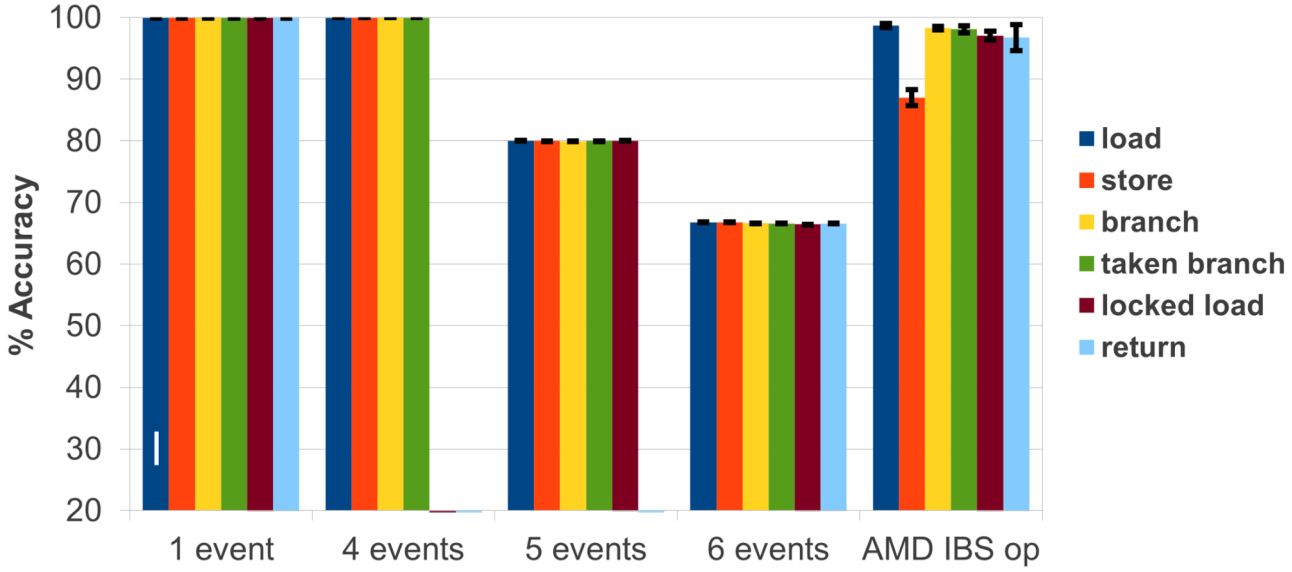
Hypothesis:

- Based on Observations 1, 2, and 3, we expect PEBS to be more accurate than IBS as long as the number of monitored events is less than or equal to the number of general-purpose counters.
- If #events > #counters, each event will lose a fraction of samples due to event multiplexing

Methodology of verification:

- Developing a microbenchmark that has known numbers of load, store, branch, taken branch, return, and locked load instructions.
- Monitoring one, four, five and six of these events in separate runs using PEBS
- Monitoring micro-operation using IBS,which already captures all these events

Multiple Event Monitoring



Findings:

- If the number of events that are simultaneously monitored is higher than the general purpose counters in PEBS, accuracy of PEBS drops.
- (Not shown) Time overhead is affected by the number of sampling interrupts rather than the number of events monitored because of event multiplexing.

Quantitative Experiments

- Experiment 1: accuracy
- Experiment 2: sensitivity to sampling interval and stability
- Experiment 3: sampling bias and instruction attribution
- Experiment 4: time overhead
- Experiment 5: memory overhead
- Experiment 6: multiple event monitoring
- **Experiment 7: kernel mode vs user mode detection**

Kernel Mode vs User Mode Detection

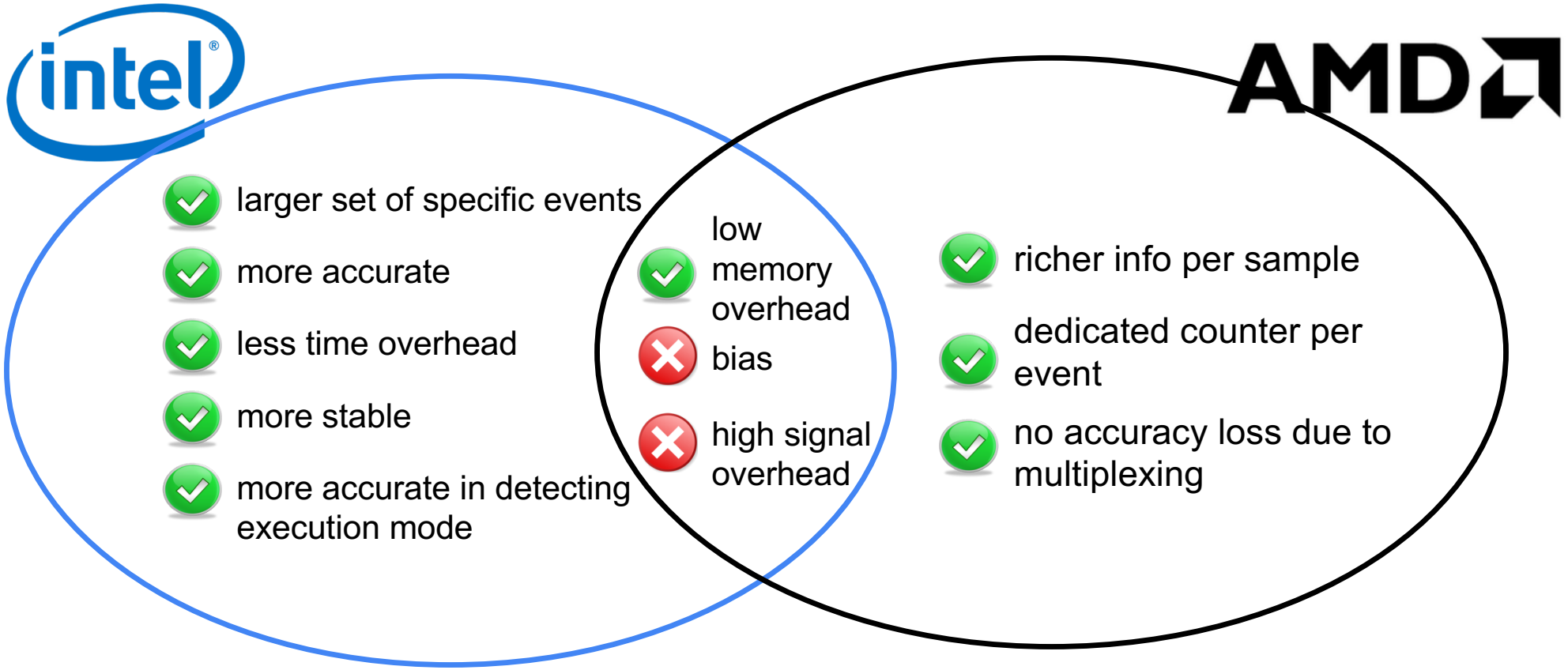
Methodology of verification:

- Using a microbenchmark, called *Exec-Mode* benchmark, that executes 1 billion locked load operations in kernel space and has no such operation in user space.
  - Ground truth of sampling bias: no locked load sample detected in user space

Findings:

- From 5 runs of the benchmark, PEBS always shows no detection of locked load sample in user mode, while IBS shows detects 41 locked load samples on average out of 10K expected samples in user space.

Summary



Submitted to TPDS and is under review

Publicly available code and benchmark repository: https://github.com/ParCoreLab/ParCoreTools

Acknowledgement

- The work is supported by the Scientific and Technological Research Council of Turkey (TUBITAK), Grant no. 120E492.
- Dr. Didem Unat is supported by the Royal Society-Newton Advanced Fellowship.
- This project has received funding from the European High-Performance Computing Joint Undertaking under grant agreement No. 956213

EuroHPC
Joint Undertaking