KOÇ UNIVERSITY

Best Paper and Best Student Paper Nominee at SC19

ComDetective: A Lightweight Tool for Detecting Inter-Thread Communication

Muhammad Aditya Sasongko (Koç University), Milind Chabbi (Scalable Machines)
Palwisha Akhtar (Koç University), **Didem Unat (Koç University)**

parcorelab.com

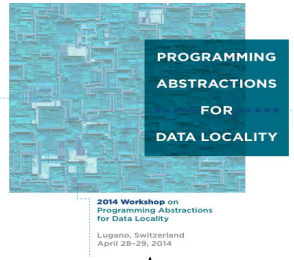
About me?

BİLİM AKADEMİSİ

MARIE CURIE ACTIONS

TEDx Istanbul
x = independently organized TED event

KOÇ UNIVERSITY
PARALLEL AND MULTICORE COMPUTING LABORATORY



Abstract Machine Models and Proxy Architectures for Exascale Computing

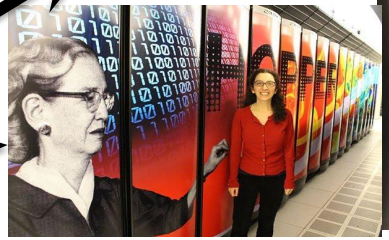
Rev 1.1

J.A. Ang[1], R.F. Barrett[1], R.E. Benner[1], D. Burke[2], C. Chan[2], D. Donofrio[2], S.D. Hammond[1], K.S. Hemmert[1], S.M. Kelly[1], H. Le[1], V.J. Leung[1], D.R. Resnick[1], A.F. Rodrigues[1], J. Shalf[2], D. Stark[1], D. Unat[2], N.J. Wright[2]

Sandia National Laboratories, NM[1]
Lawrence Berkeley National Laboratory, CA[2]

May, 16 2014

BERKELEY LAB
Lawrence Berkeley National Laboratory



Research Interests

- Address software challenges of emerging architectures
- Develop tools in collaboration with computational scientists
  - Programming models and runtime systems
    - Data locality is at the center
    - Focus on homogeneous and heterogeneous large-scale systems
    - Embrace asynchrony to scale on thousands of processors
  - Tools for performance monitoring and modeling
    - Design and develop tools for performance modeling and optimization on multicore and heterogeneous architectures

TiDA | Perilla | ExaSAT | TaskSanitizer | ComDetective

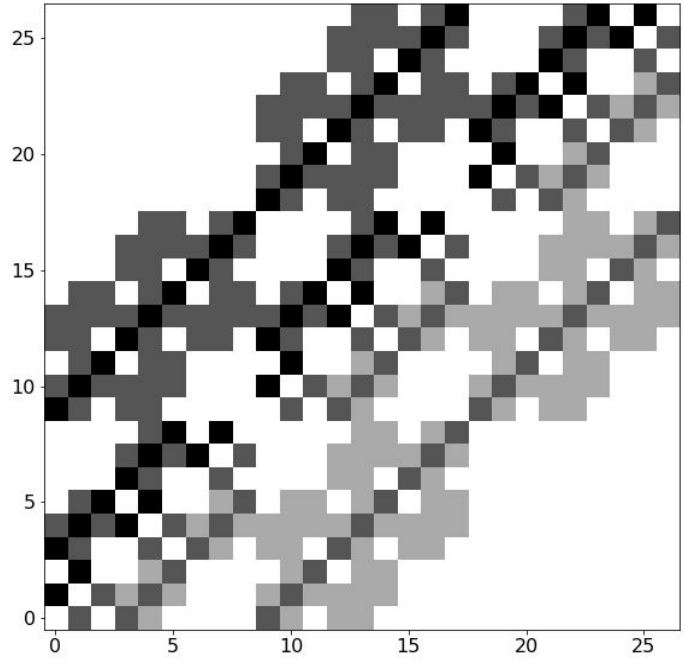
Modern HPC Applications

- Employ multi-socket, multicore, many-core CPUs within a node
- Use MPI+Threads
  - MPI for communication among nodes
  - OpenMP or other threading models for intra-node communication
- Do **explicit inter-process** communication
  - Managed via message passing (e.g., MPI) Send/Recv primitives
- Do **implicit inter-thread** communication
  - Hidden by standard load/store CPU instructions

Regardless of communication type, data transfer is dominant in performance and energy consumption.

Need Communication Detection Tools

MPI communication matrix for LULESH via EZTrace

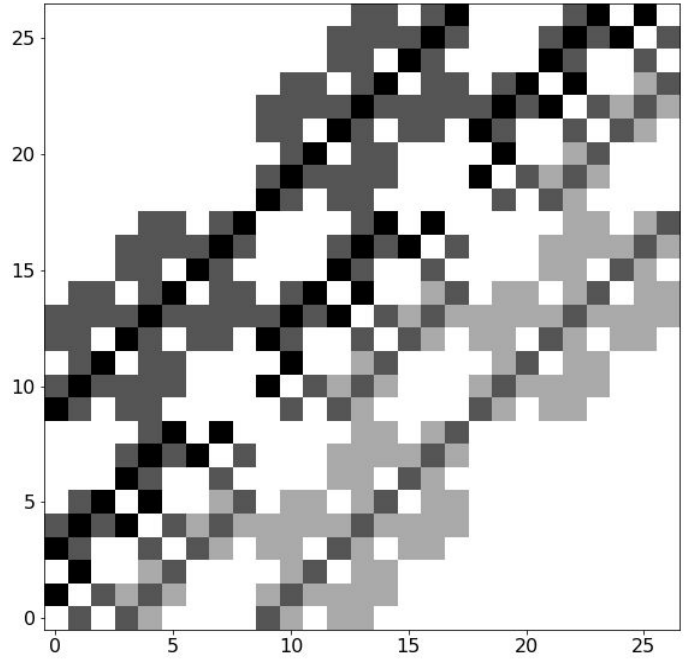


Inter-thread communication matrix

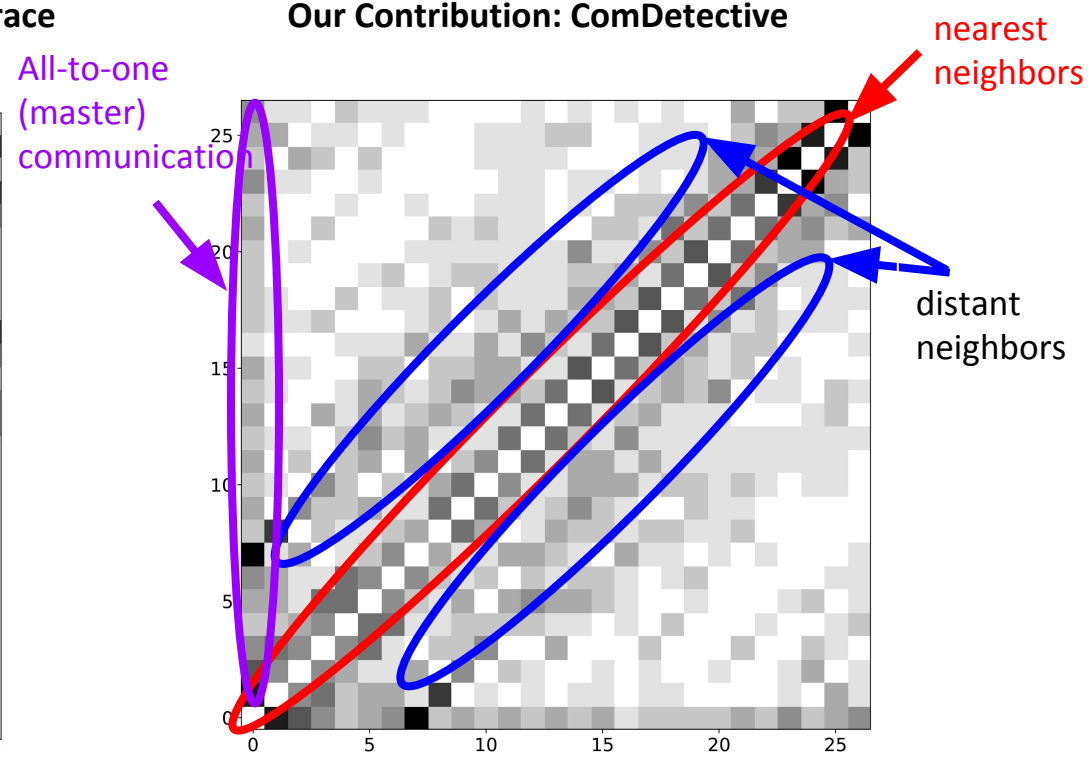
?

Need Communication Detection Tools

MPI communication matrix for LULESH via EZTrace

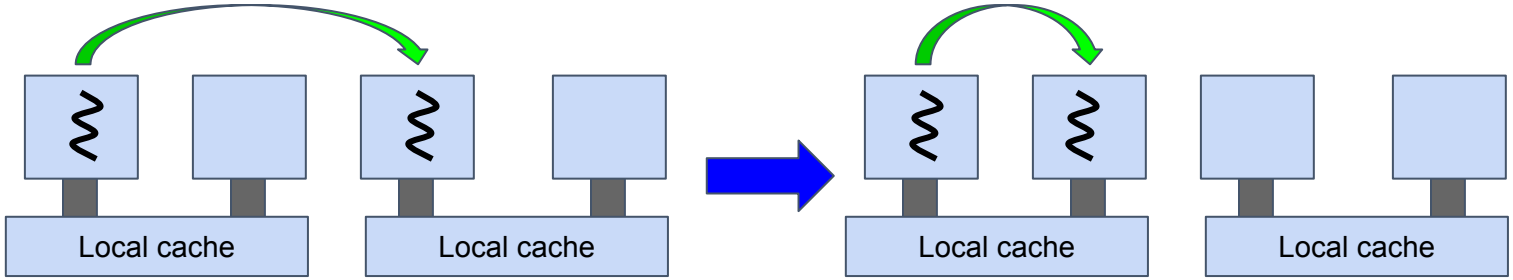


Our Contribution: ComDetective



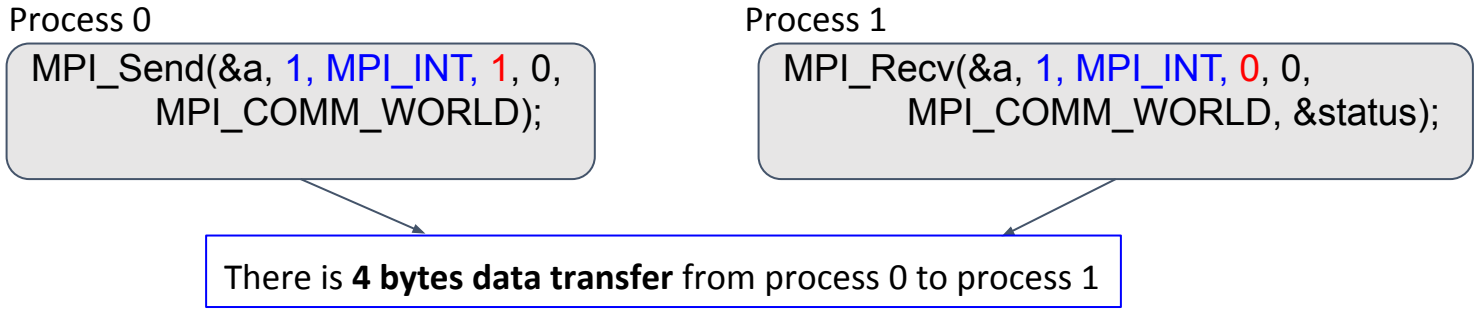
Why Detect Inter-Thread Communication?

- 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



Challenges

- Inter-process communication detection in MPI is relatively straightforward



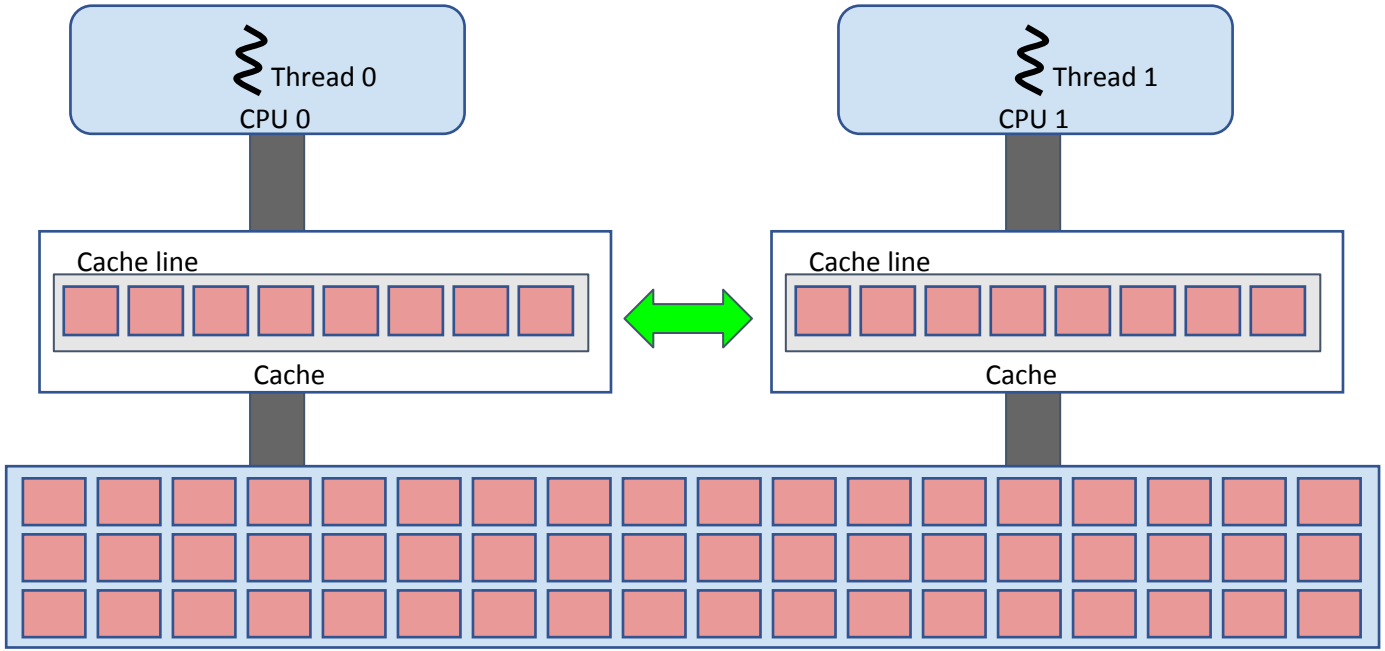
- Exact inter-thread communication detection poses some challenges
  - requires **interception of load and store** operations
  - incurs **huge space and time overheads** if all load and store operations are intercepted
  - **dilates execution** and changes program behavior
  - **scales poorly** with increasing number of threads

ComDetective: Salient Features

- **Accurate**
  - Validated against several benchmarks and HPC applications
- **Lightweight**
  - Space overhead (1.3x) and time (1.3x) overhead
- **Sampling-based**
  - Uses hardware performance monitoring units
- **Differentiates the kind of communication**
  - True sharing (necessary) vs. false sharing (unnecessary)
- **Data objects**
  - Attributes communication to program data objects
- Open source: https://github.com/comdetective-tools

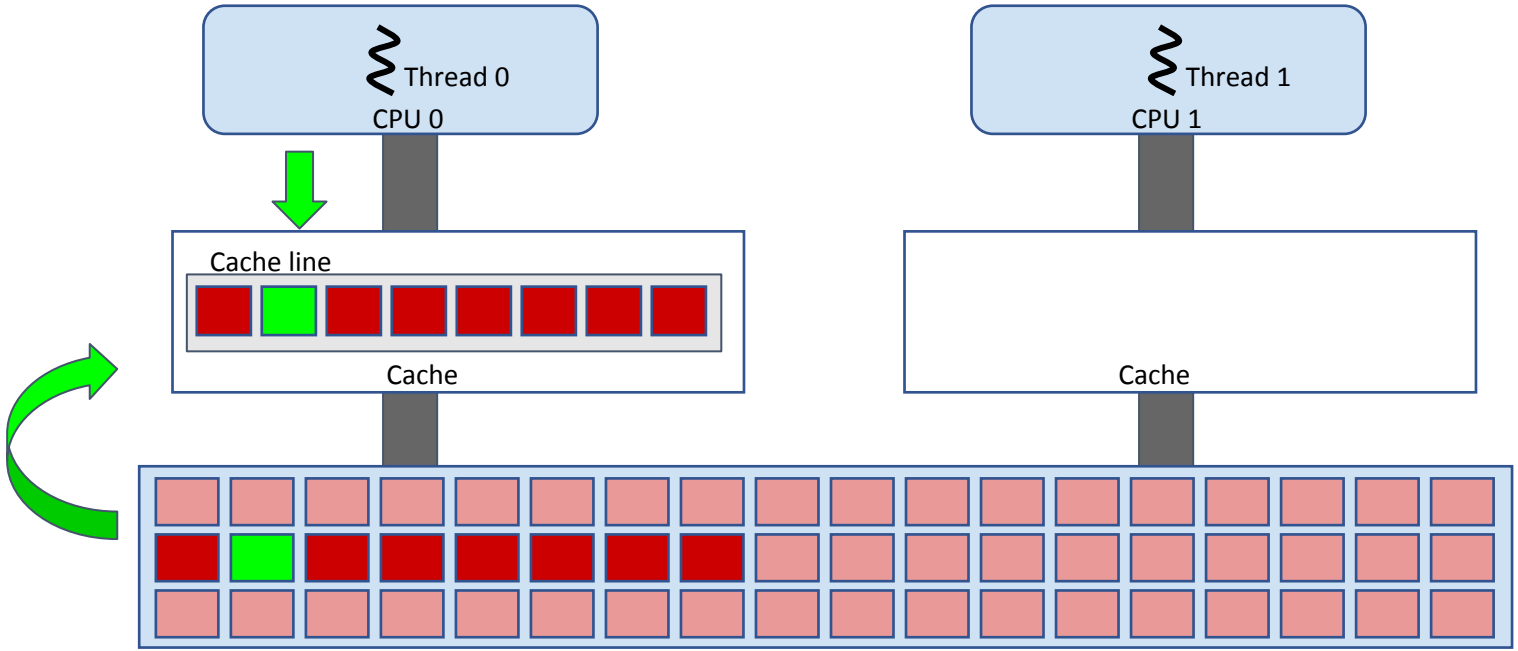
Inter-Thread Communication

- Occurs in multi-threaded programs or hybrid programs (e.g. MPI+OpenMP hybrid)
- Occurs at CPU cache line granularities



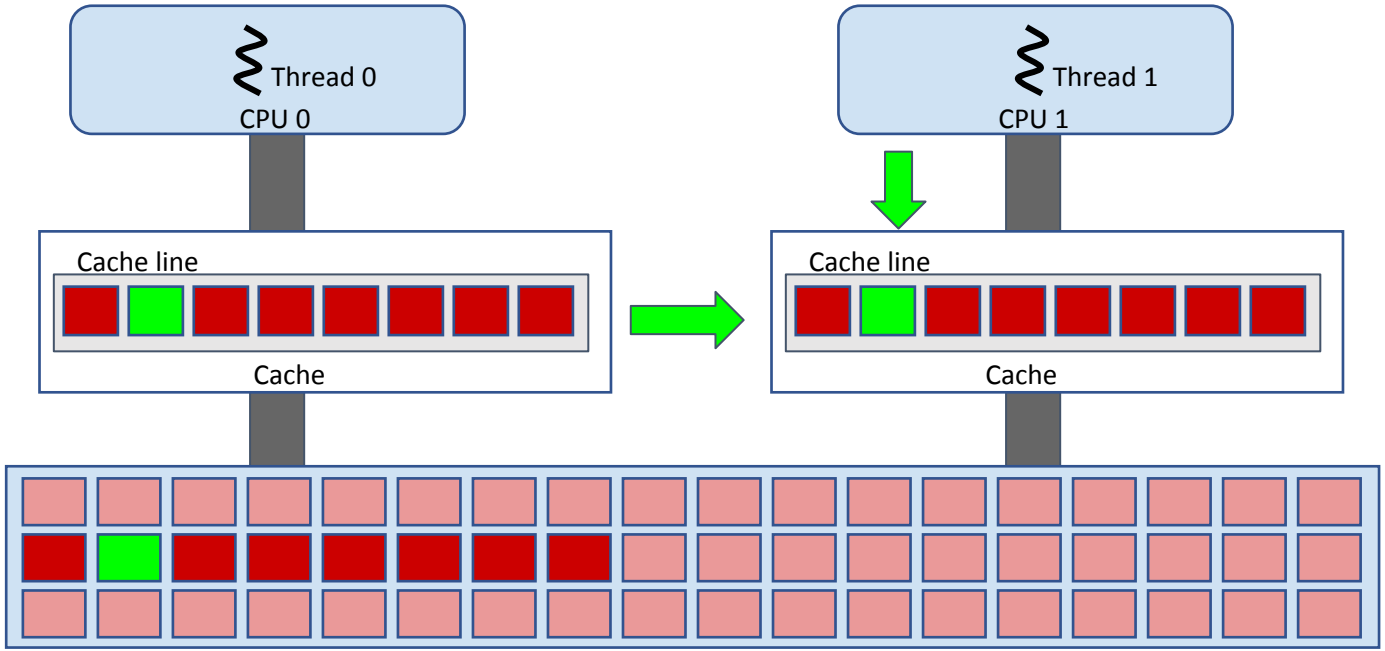
Inter-Thread Communication

- Memory access by CPU 0



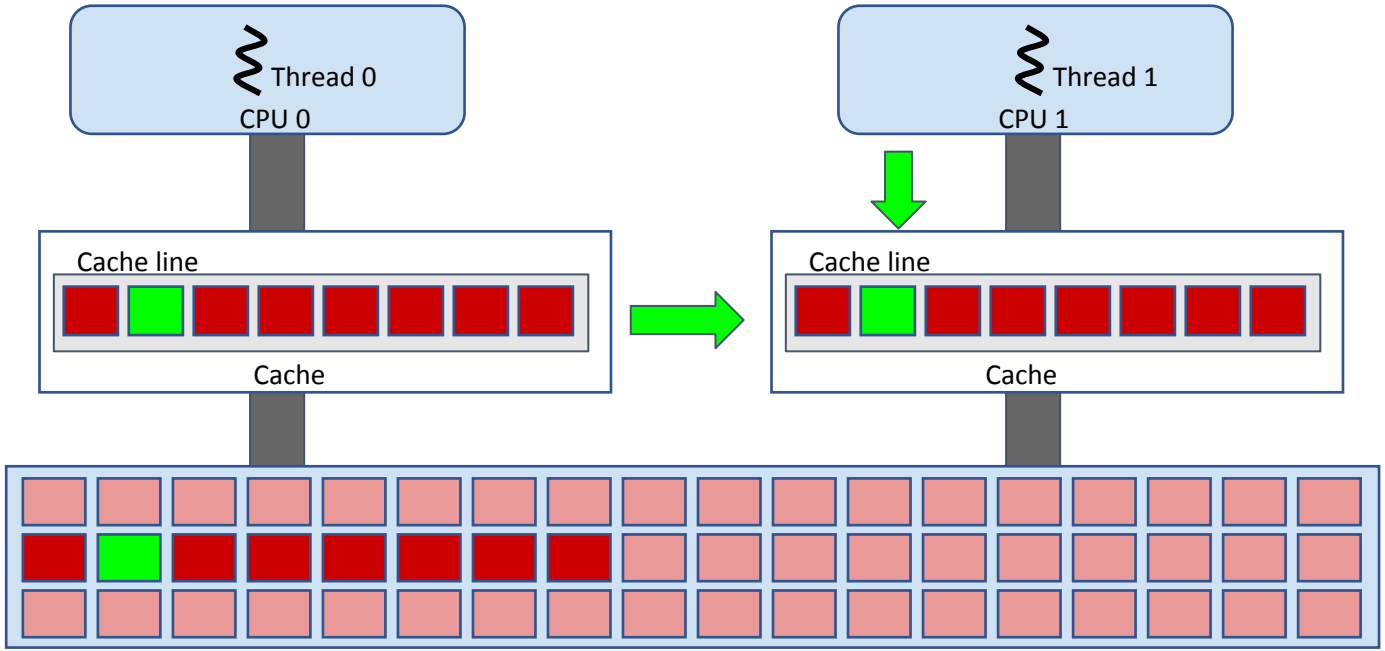
Inter-Thread Communication

- Memory access by CPU 1



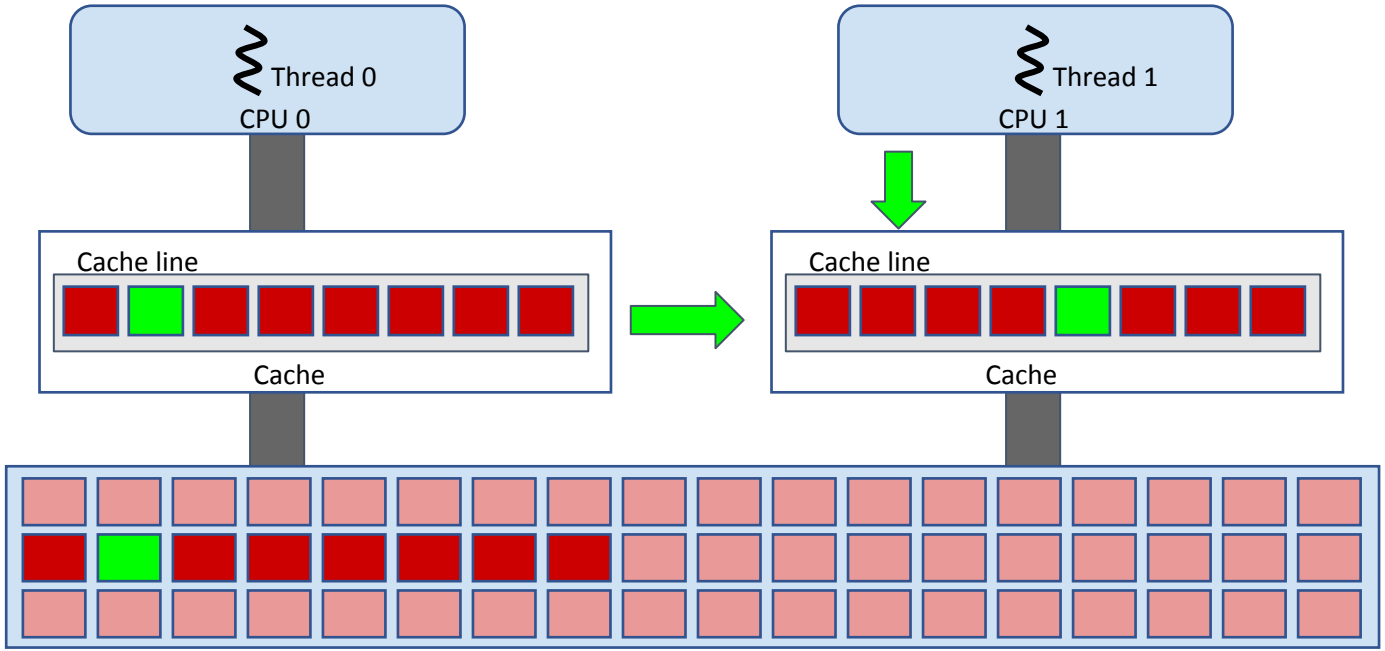
Inter-Thread Communication: Necessary

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

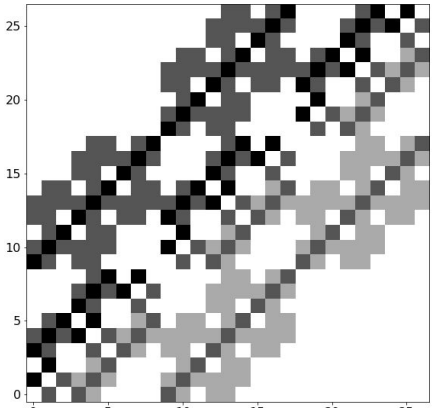


Inter-Thread Communication: Unnecessary

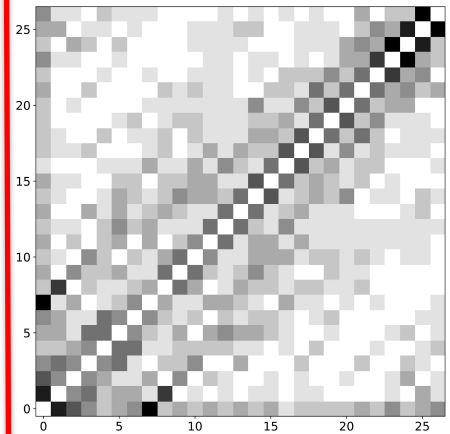
- Another possible type is **false sharing**
- Threads 0 and 1 access different memory regions in the same cache line



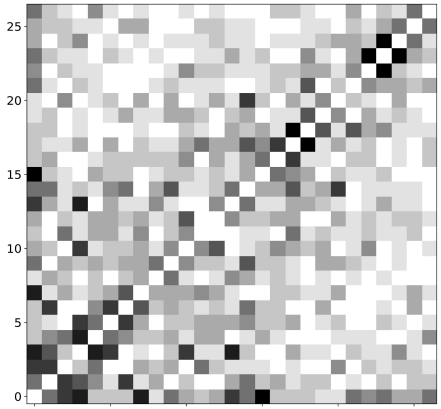
An Example Output from LULESH



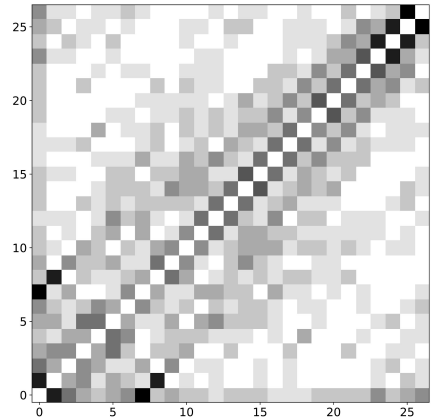
MPI communication matrix



communication matrix



true sharing matrix



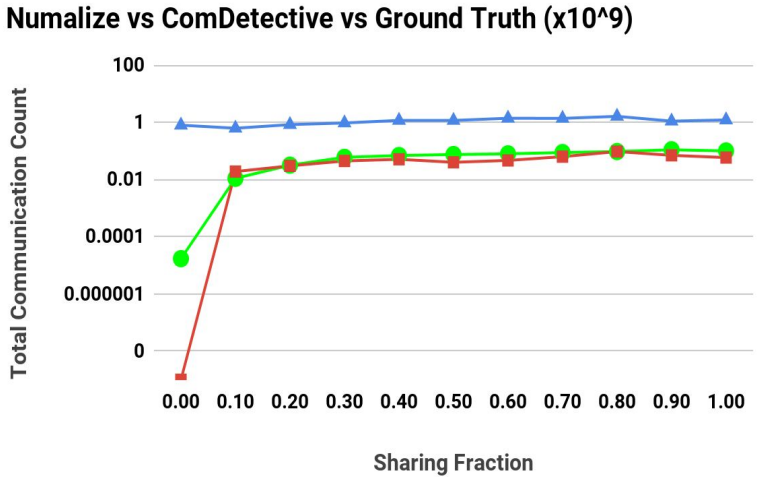
false sharing matrix

- In addition to communication matrix, ComDetective also produces true sharing and false sharing matrices
- It took only **1.28x** performance and **1.11x** memory footprint overhead to generate these matrices with ComDetective

Existing Tools

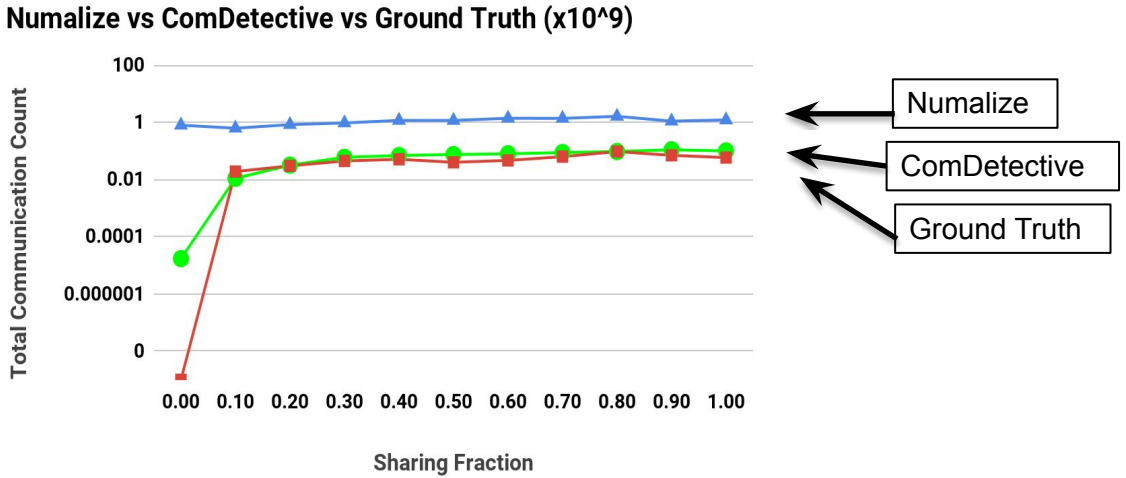
- Prior works on identifying inter-thread communication employed **hardware simulators** or **binary instrumentation**
  - Suffered from inaccuracy or high overhead
- Simulator-based tools [Barrow-Williams, et al, IISWC 2009] [Molina da Cruz, et al, IPDPSW 2011] [Diener, et al, PDP 2016]
  - Incurring huge memory footprints and very slow
  - Requires offline profiling
  - Not running on real hardware, so execution behavior can change.
- Performance monitoring units (PMUs)-based tools [Azimi, et al. ACM SIGOPS Operating Systems Review 2009][Tam, et al. EuroSys 2007].
  - Can be intrusive as it requires modification of kernel source code

Existing Tools



- Code instrumentation: binary [Diener, et al. Performance Evaluation 2015][Diener, et al, PDP 2016] (Numalize) and compiler-assisted [Mazaheri, et al. ICPP 2015][Mazaheri, et al. ICPP 2018]
  - Can suffer from large slowdown and memory overhead
  - We have found that Numalize is not accurate.

Existing Tools



- Code instrumentation: binary [Diener, et al. Performance Evaluation 2015][Diener, et al, PDP 2016] (Numalize) and compiler-assisted [Mazaheri, et al. ICPP 2015][Mazaheri, et al. ICPP 2018]
  - Can suffer from large slowdown and memory overhead
  - We have found that Numalize is not accurate.

ComDetective

- We develop a tool to detect inter-thread communication called **ComDetective**
- ComDetective is
  - **Fast --** uses **available hardware features**; PMUs and debug registers
  - **Accurate** -- has **been validated** in terms of correctness of **total communication volume** and correctness of **point-to-point communication ratio**
- ComDetective also
  - Differentiates **true sharing and false sharing** communications -- by detecting if **memory regions** accessed by communicating threads **overlap or not**
  - Associates communication matrices not only to the whole program but **to program objects** -- for global, stack, and heap objects

Outline

- Background Information on Inter-Thread Communication
- Motivation for Detecting Inter-Thread Communication
- Prior Arts
- Introduction to ComDetective
- Design Components
- Workflow
- Detailed Evaluation

Big Picture

- Inter-thread communication occurs between two threads if,

  Two threads access an address residing on the same cache line in a short interval

- Question: How to detect cache line communication?

Big Picture

- Inter-thread communication occurs between two threads if,

  Two threads access an address residing on the same cache line in a short interval

- Question: How to detect cache line communication?
  - A thread can **sample** its memory accesses via hardware performance counters (address sampling)
  - No load/store instrumentation ⇒ super low overhead

Big Picture

- Question: how can another thread know if it is accessing the same address **without instrumenting its loads and stores**?

Big Picture

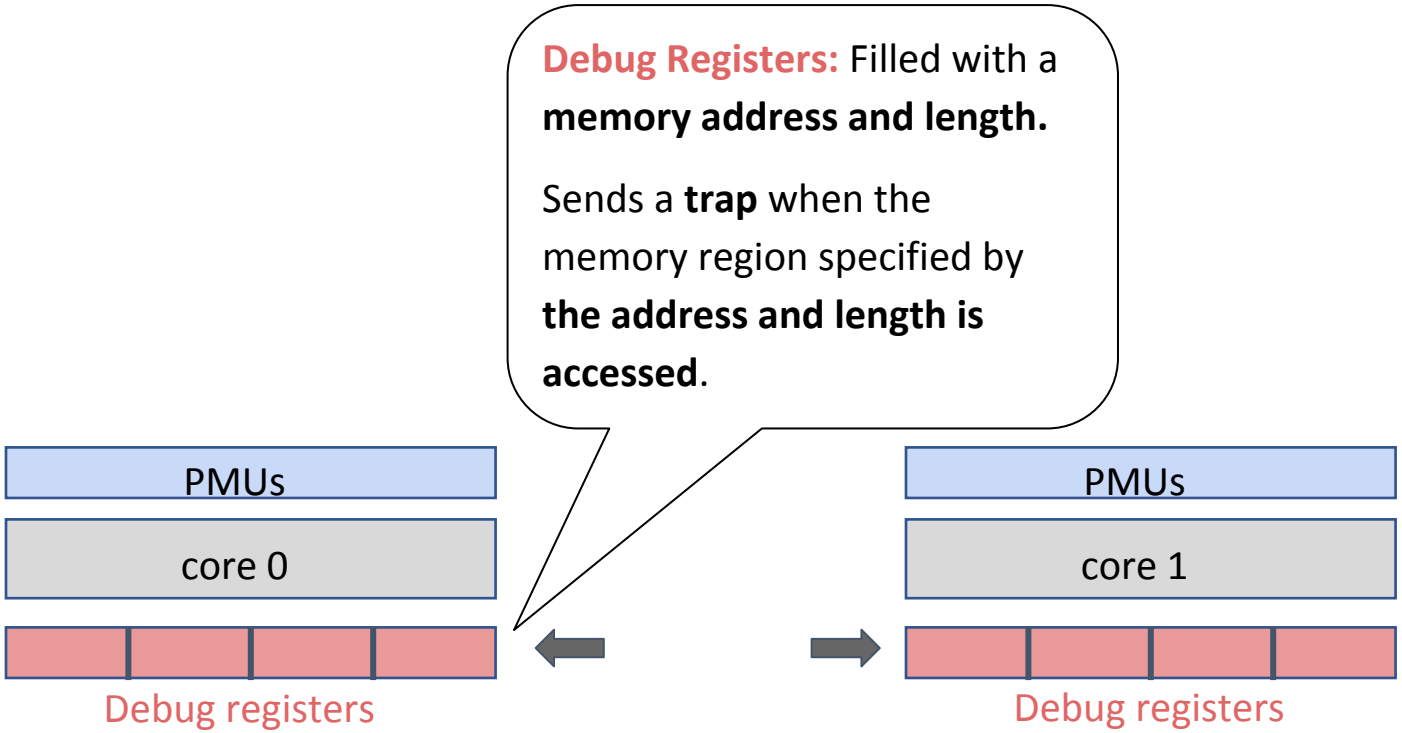
- Question: how can another thread know if it is accessing the same address **without instrumenting its loads and stores**?
- Answer:
  - The first thread
    - **publishes its sampled address** to a globally visible location
  - The second thread
    - **compares its sampled address** with the globally published addresses and if there is a match ⇒ inter-thread communication, or
    - uses **hardware-debug registers** (aka watchpoints) to monitor a globally published address
  - Watchpoint traps when the second thread accesses the same address ⇒ inter-thread communication

Design Components: PMU

PMU: Special registers that **count low-level events**, such as loads or stores.

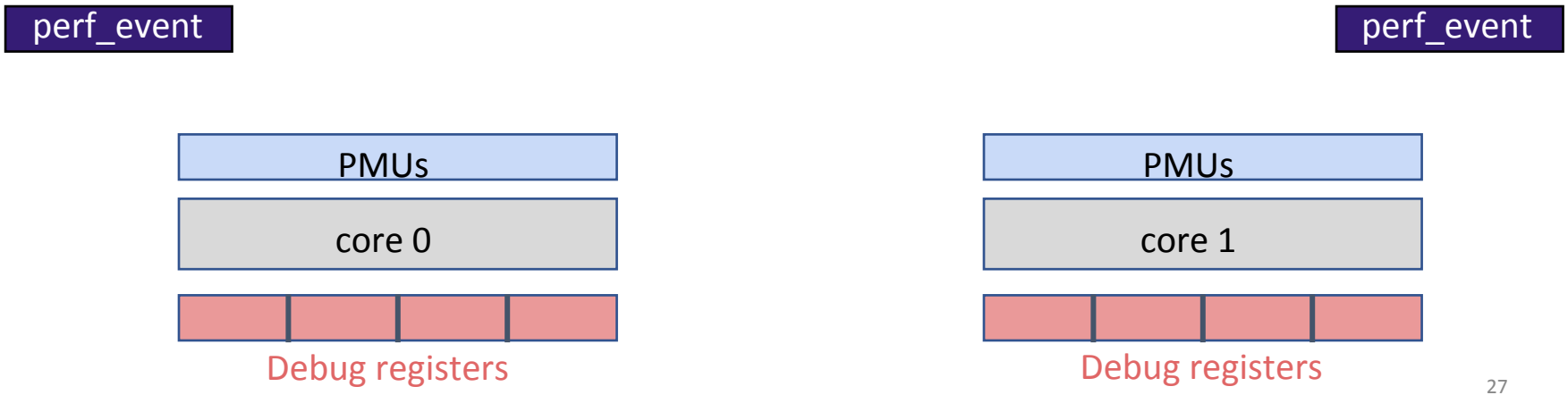
Sampling: PMUs can be configured to trigger interrupt f**or every N events.**

Design Components: Debug Registers

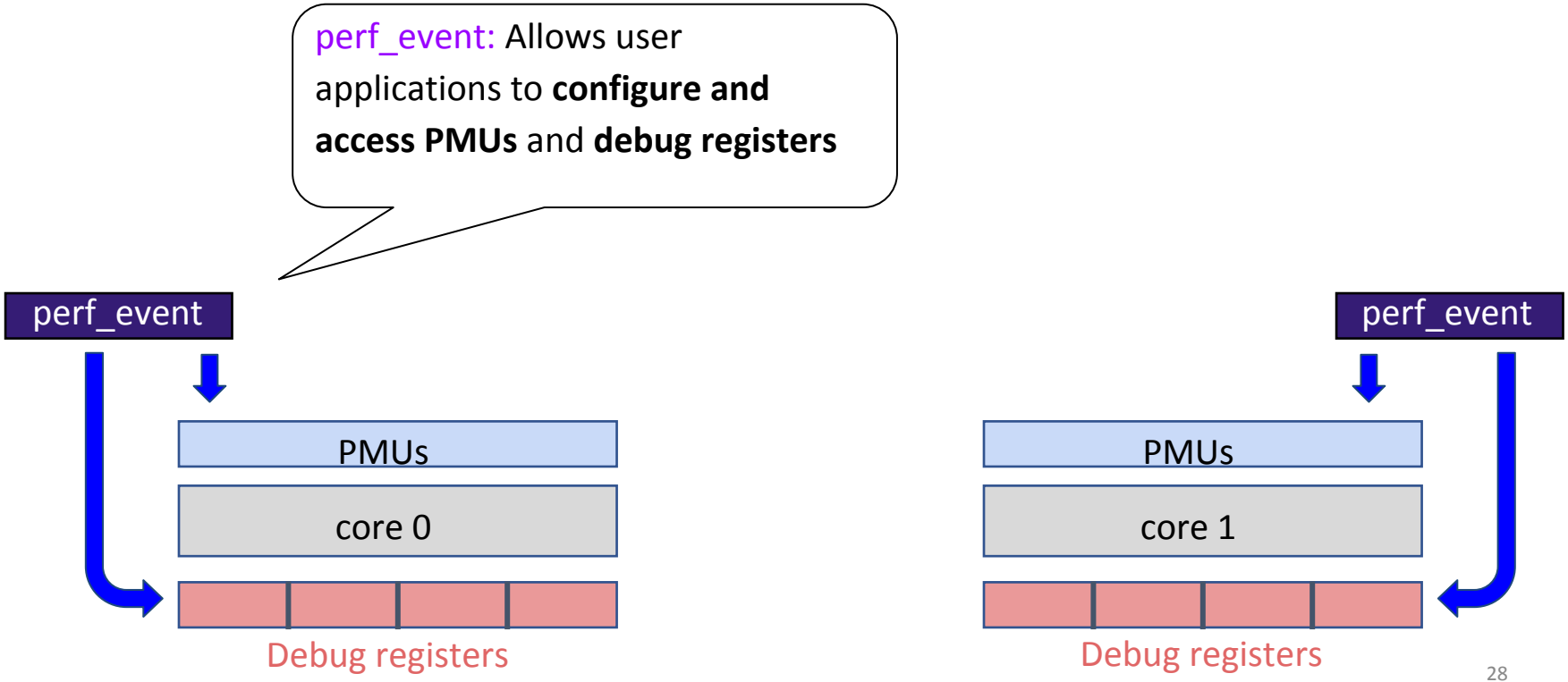


Design Components: perf_event

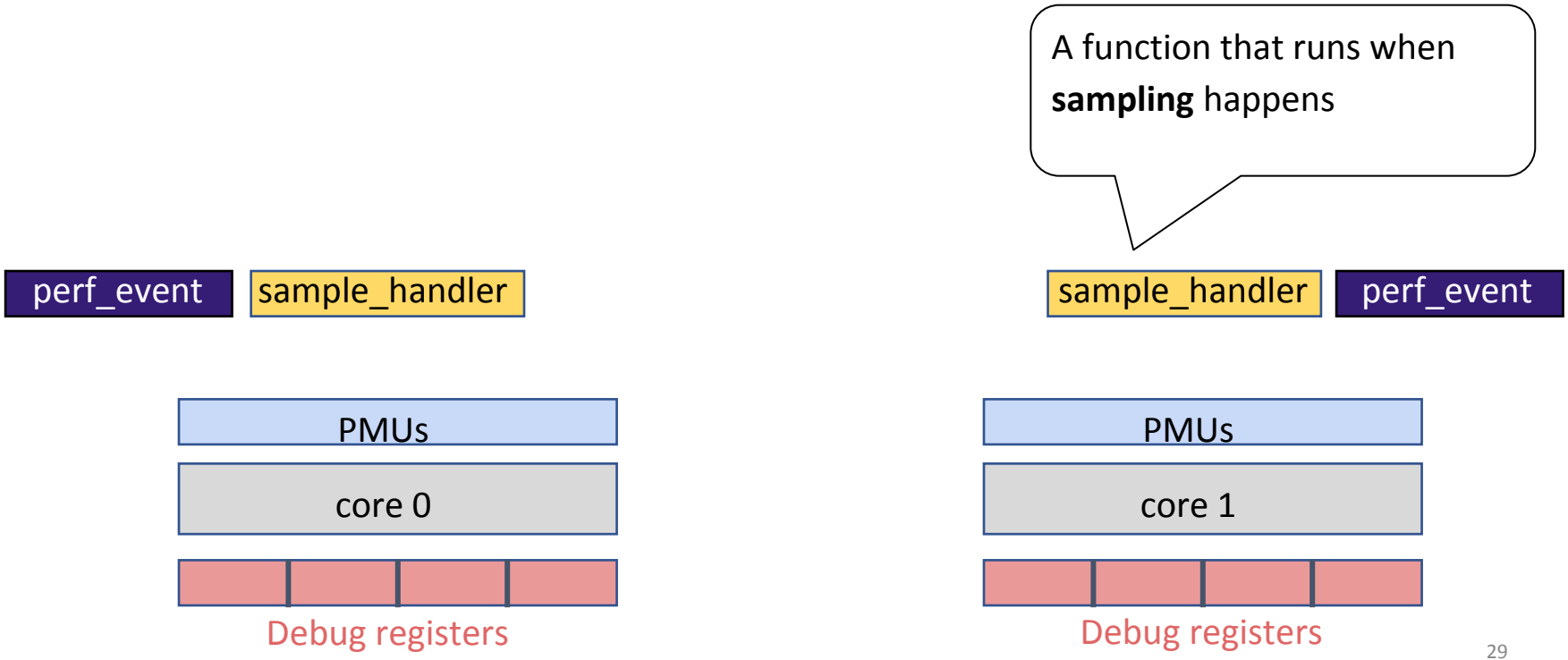
perf_event: Allows user applications to **configure and access PMUs** and **debug registers**



Design Components: perf_event



Design Components: ComDetective



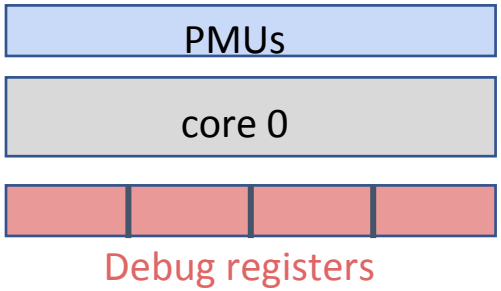
Design Components: ComDetective

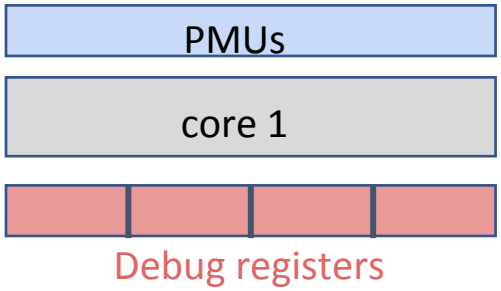
key	attributes

Bulletin Board: A **hash table** that publishes some of the **sampled data**

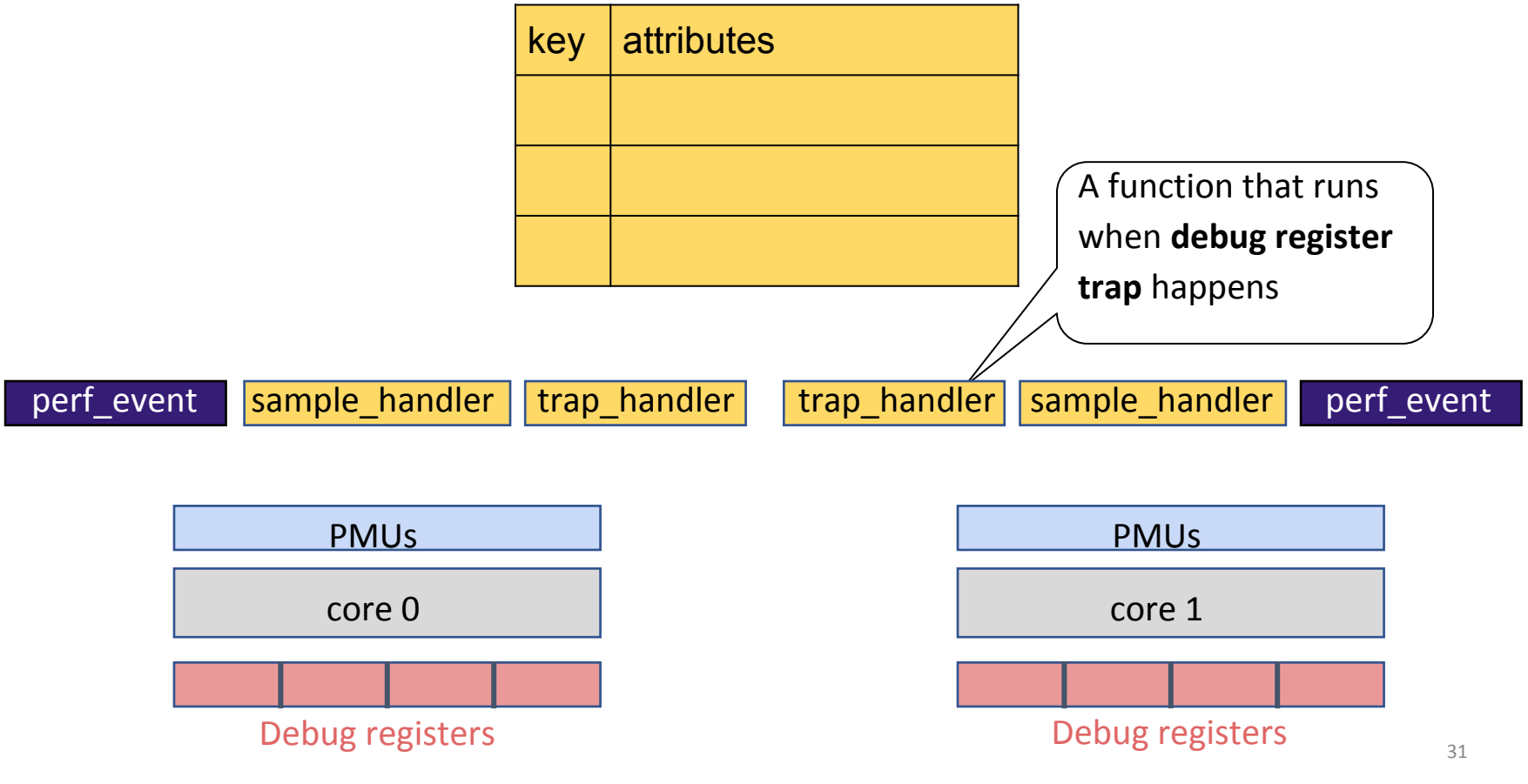
perf_event | sample_handler

sample_handler | perf_event

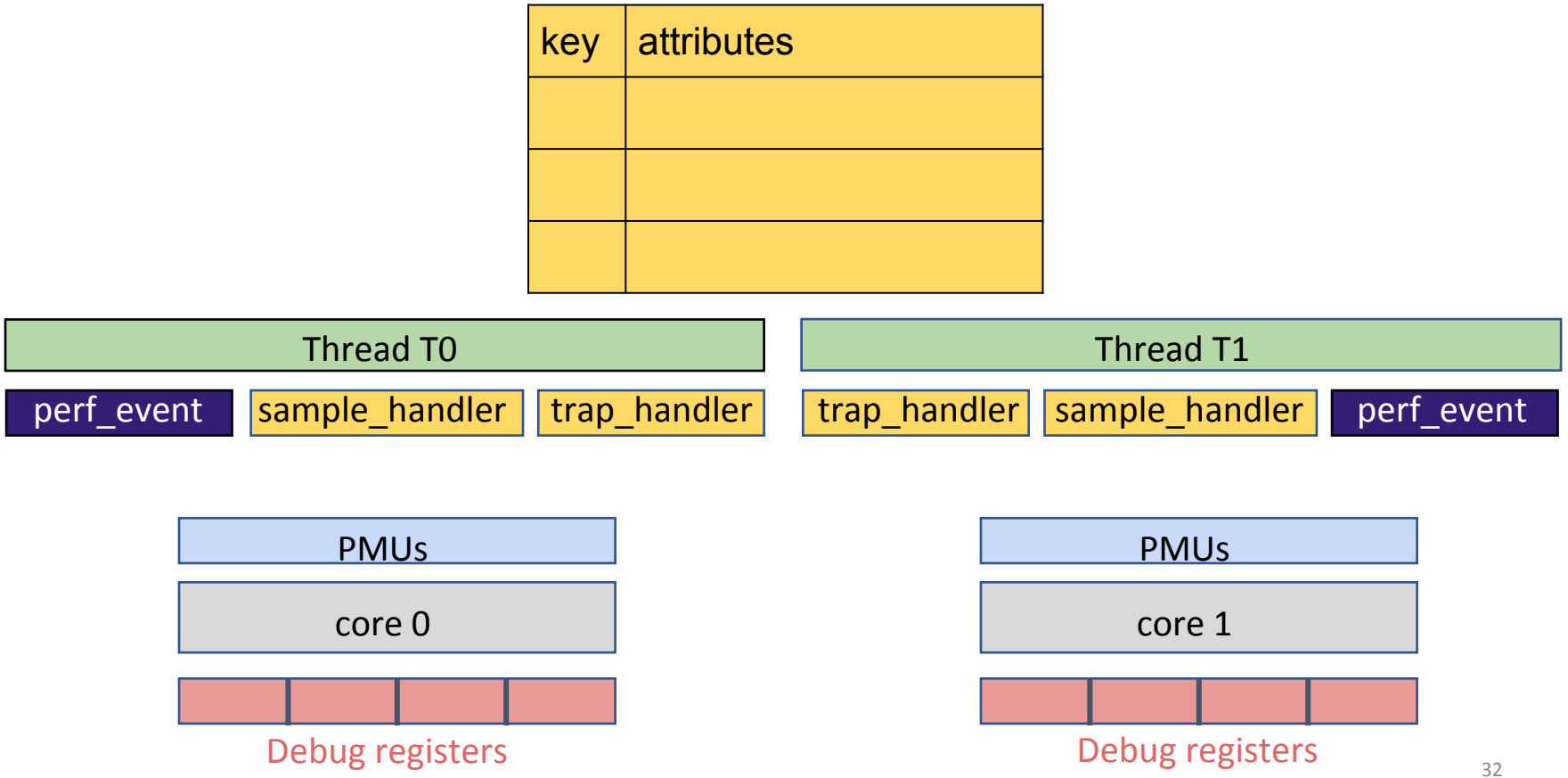




Design Components: ComDetective



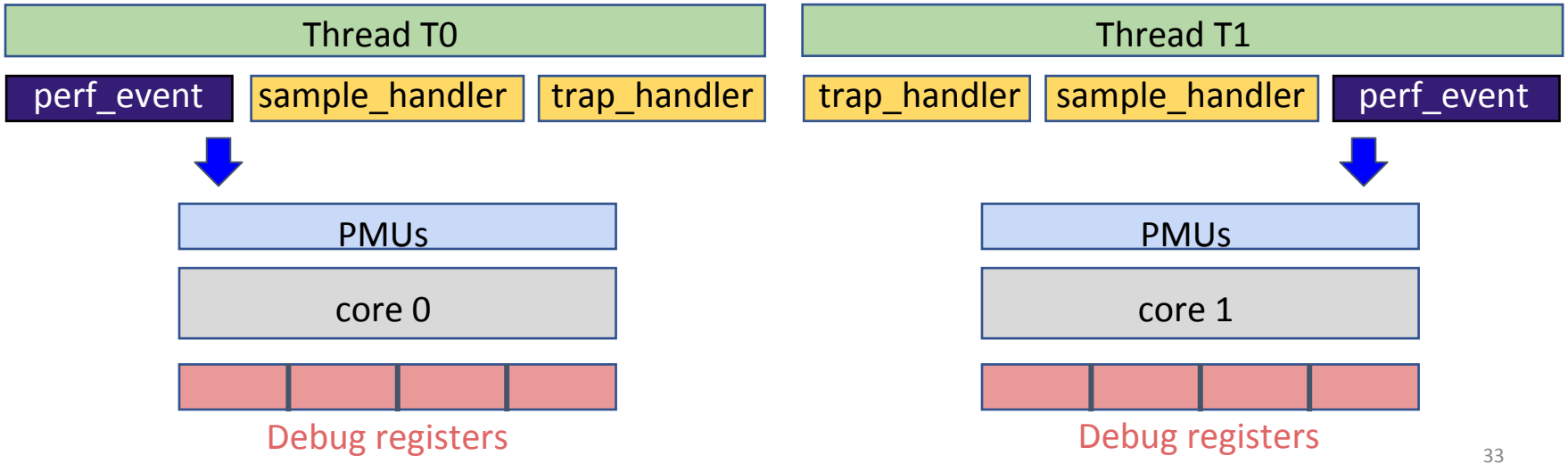
Design Components



An Example Workflow

perf_event is configured so that **loads** and **stores** to be sampled for every **N events**.

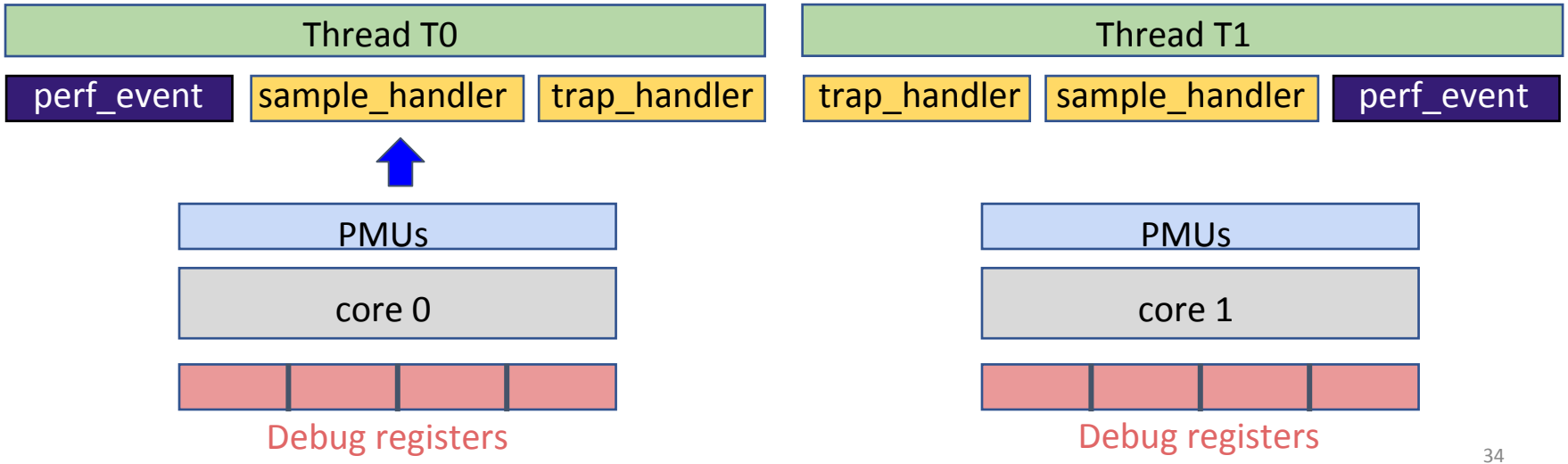
key	attributes
-1	
-1	
-1	



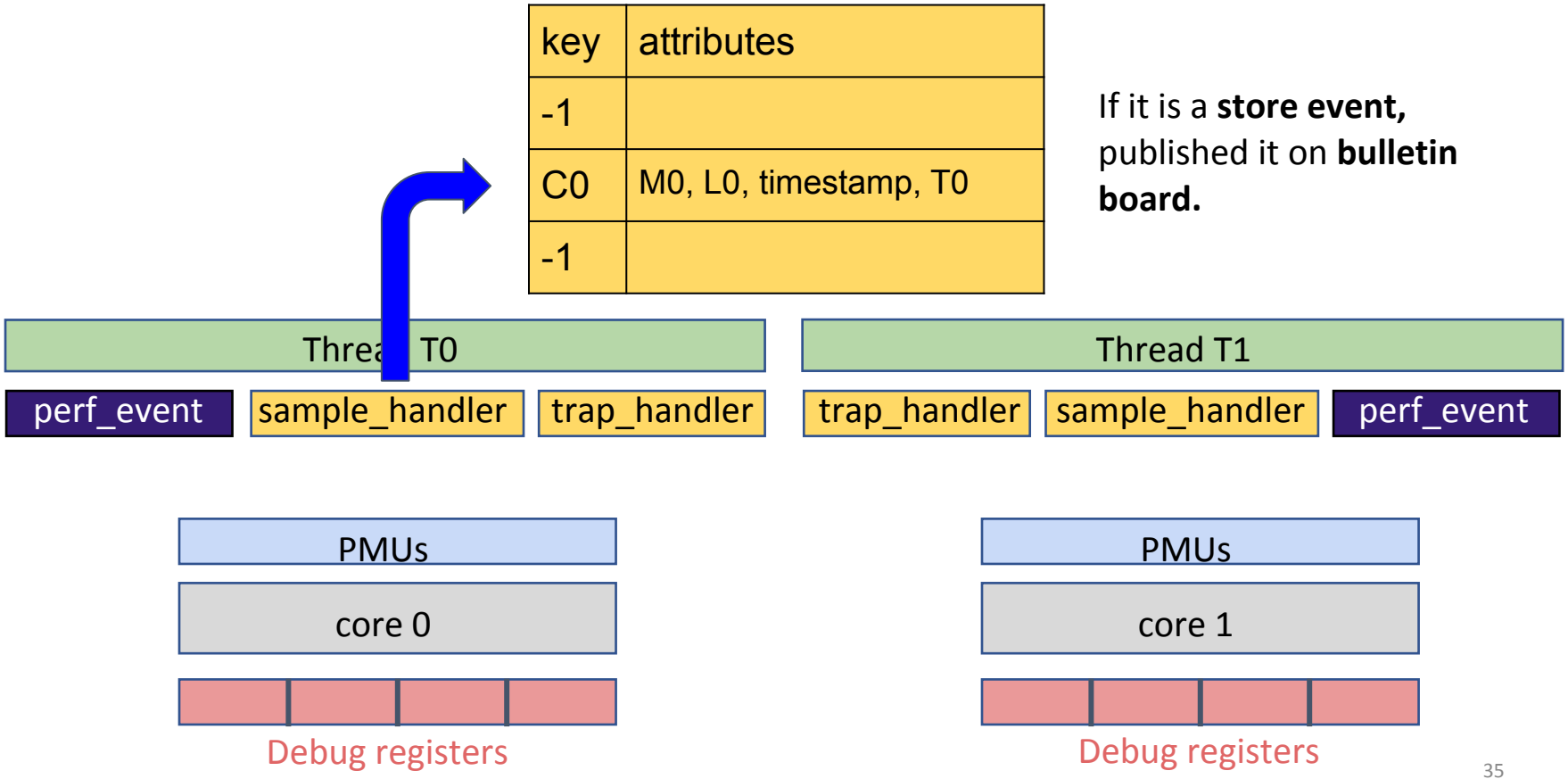
Workflow

Interrupt happens in core 0 after N events.

key	attributes
-1	
-1	
-1	



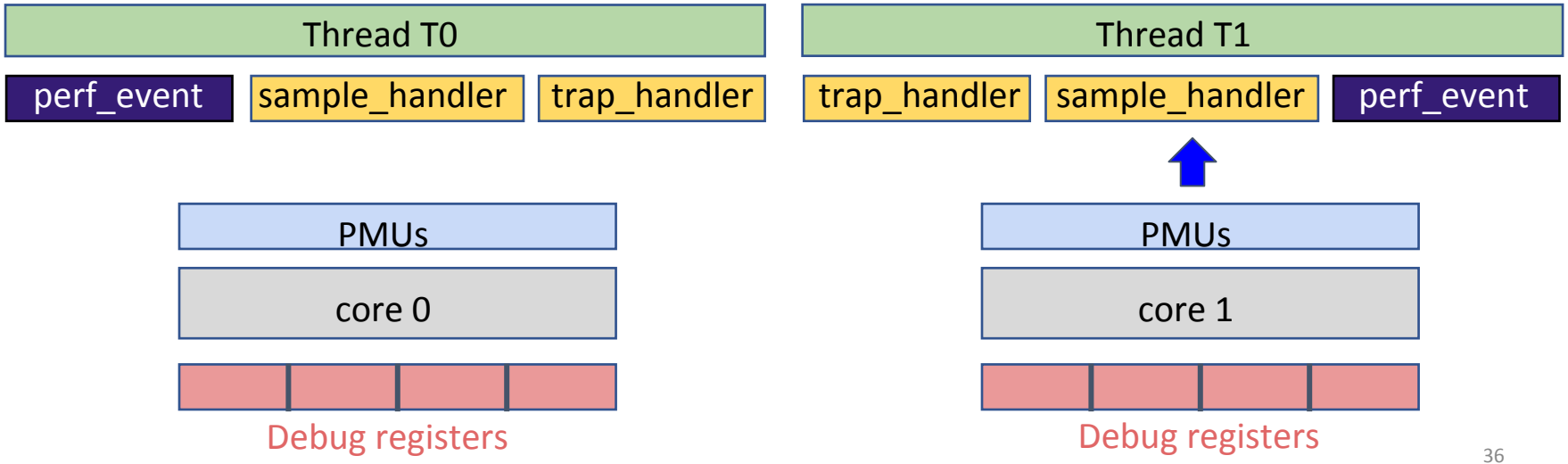
Workflow


key
attributes
-1
C0
M0, L0, timestamp, T0
-1
If it is a **store event,** published it on **bulletin board.**
Thread T0
Thread T1
perf_event
sample_handler
trap_handler
trap_handler
sample_handler
perf_event
PMUs
core 0
Debug registers
PMUs
core 1
Debug registers

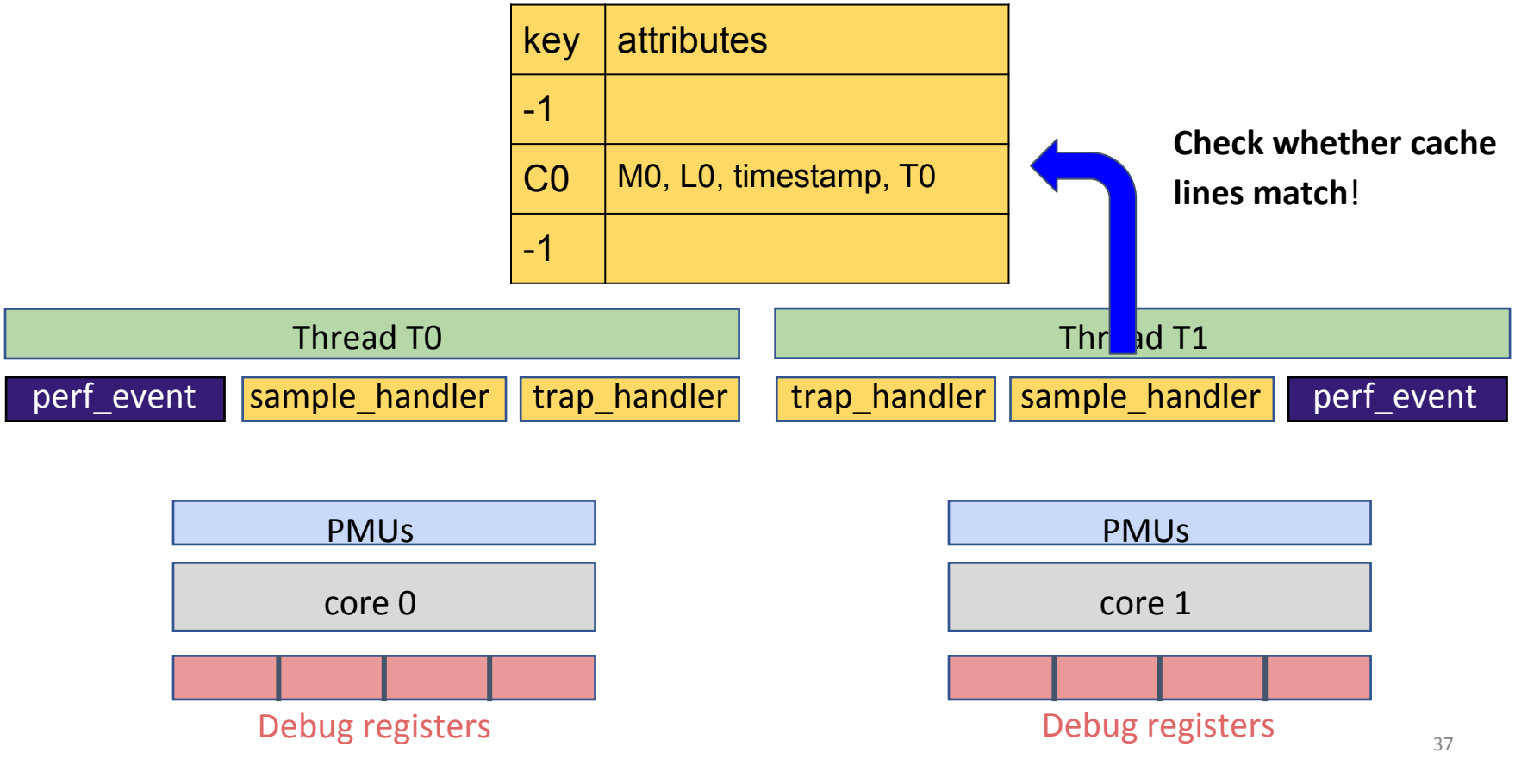
Workflow

key	attributes
-1	
C0	M0, L0, timestamp, T0
-1	

Interrupt happens in core 1. The triggering event is **a store to memory address M1.**



Workflow

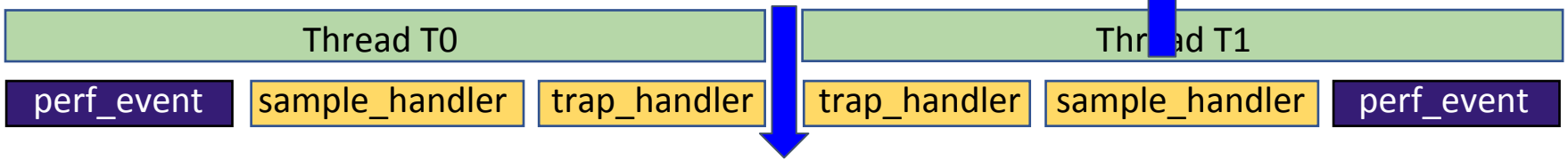

key
attributes
-1
C0
M0, L0, timestamp, T0
-1
Check whether cache lines match!
Thread T0
Thread T1
perf_event
sample_handler
trap_handler
trap_handler
sample_handler
perf_event
PMUs
core 0
Debug registers
PMUs
core 1
Debug registers

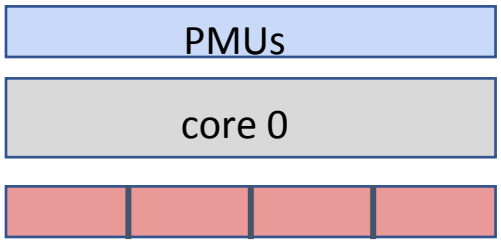
Workflow

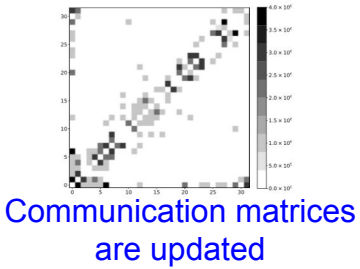
Check hash table entry whether there is the entry is 'recent'.

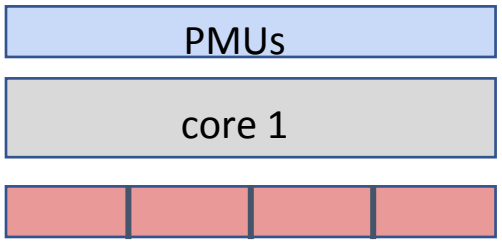
key	attributes
-1	
C0	M0, L0, timestamp, T0
-1	

If C0's entry is 'recent', **communication is detected** between T0 and T1.





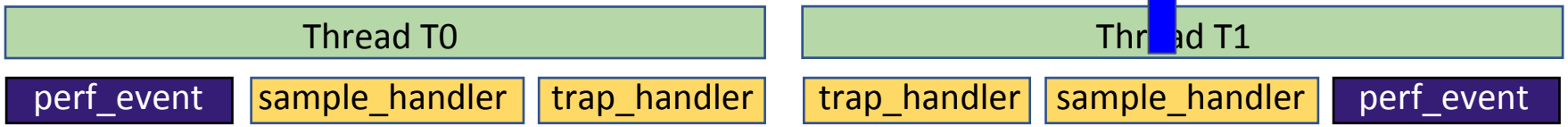


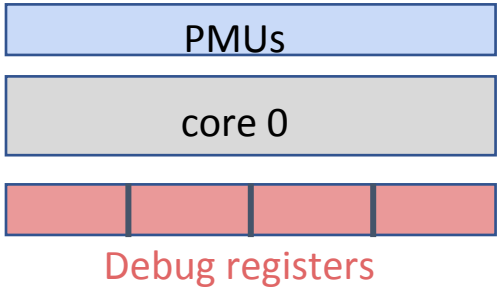


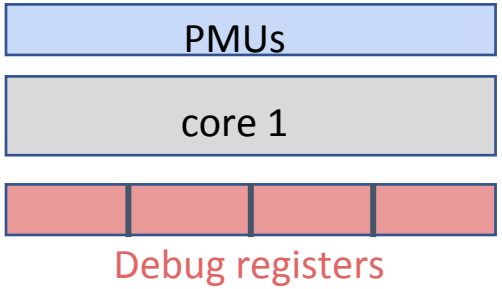
Workflow

key	attributes
-1	
C1	M1, L1, timestamp, T1
-1	

If C0's entry is **not 'recent'**, **replace** the entry with M1.





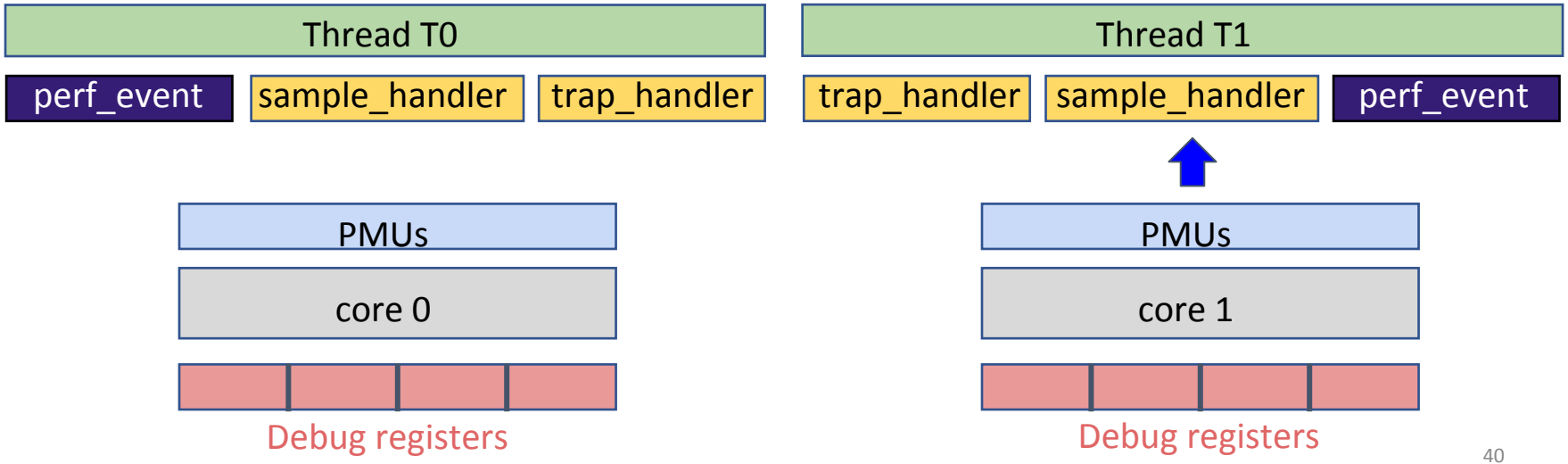


Workflow

key	attributes
…	…..
…	……
…	……

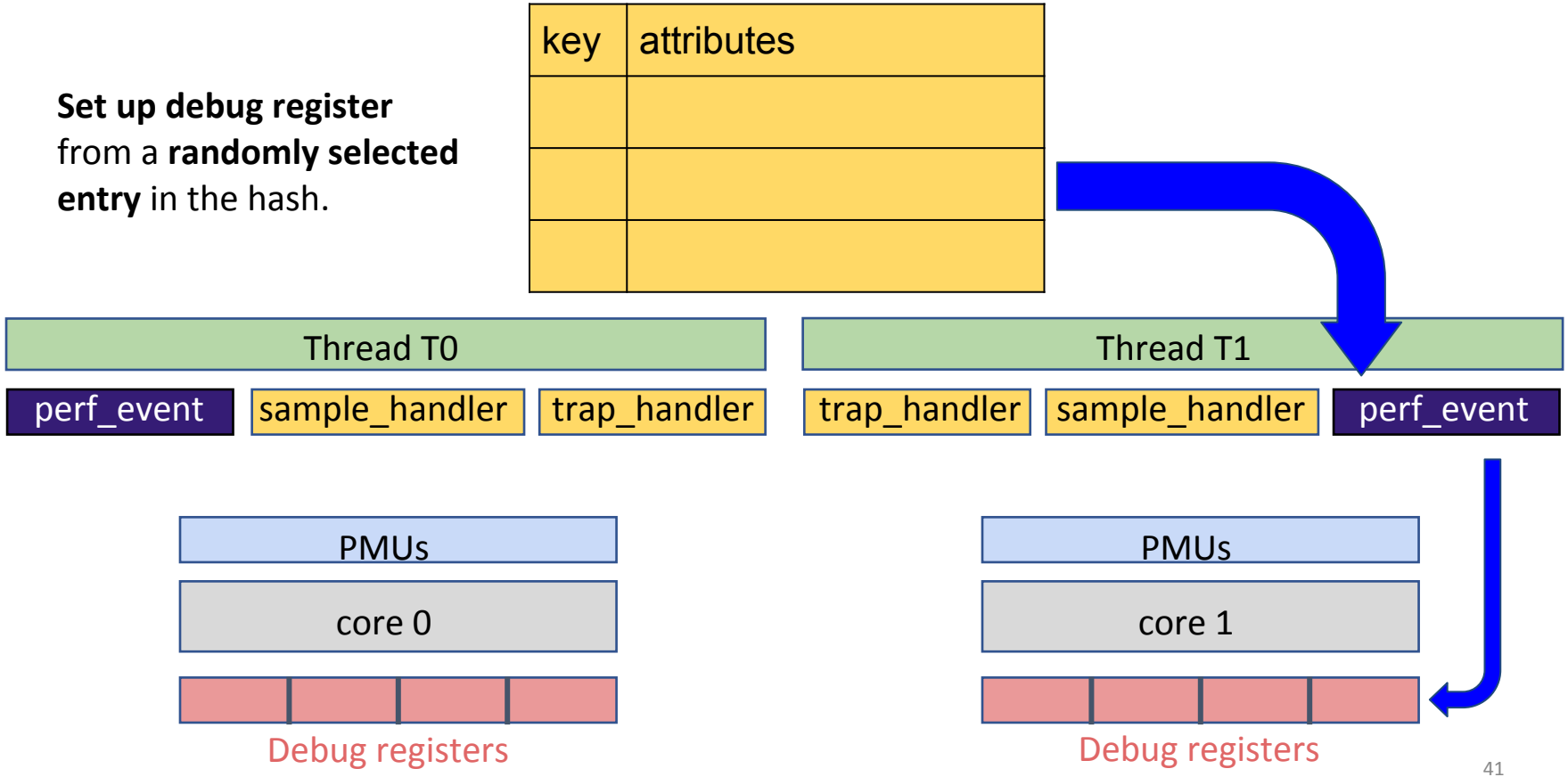
Another store sample happens on address M3.

No matching cache line and all entries are **‘recent’**, so **none can be replaced**.



Workflow

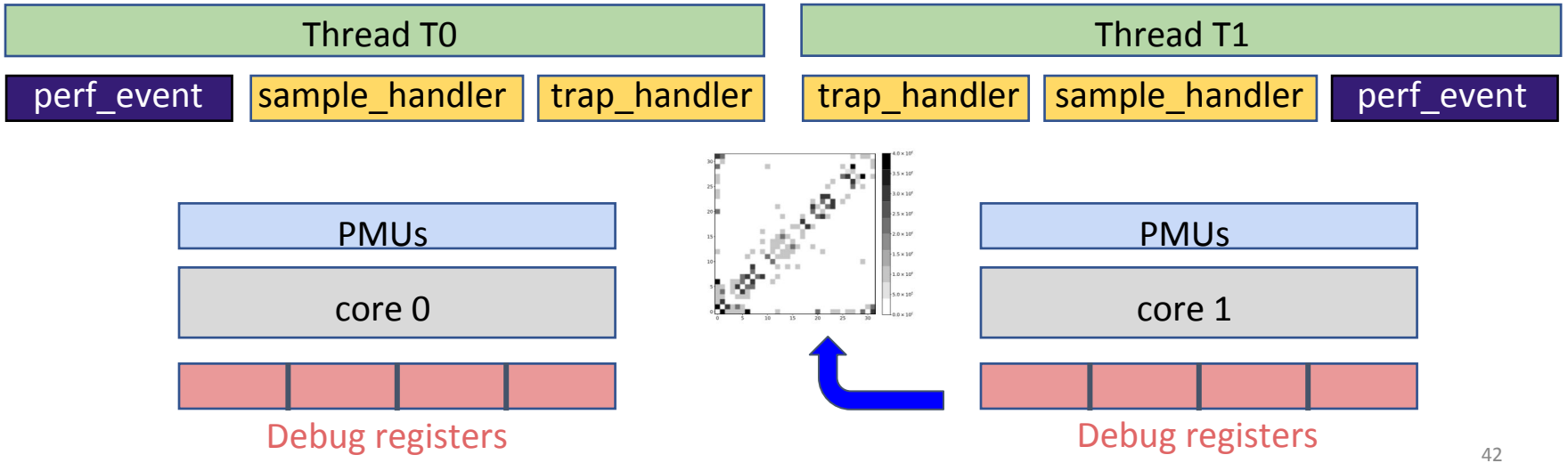
Set up debug register from a **randomly selected entry** in the hash.



Workflow

When **trap in a debug register** happens, **communication matrices** are updated.

key	attributes



Evaluation

- Accuracy verification with micro-benchmarks
  - Communication volume
  - True/false sharing ratio (reported only in our paper)
  - Point-to-point communication ratio
  - Read/write communication volume (reported only in our paper)
- Communication matrices of large benchmarks
  - 12 PARSEC and 6 CORAL applications
- Use cases: code refactoring
- Sensitivity Analysis (reported only in our paper)
  - Sampling interval impact
  - Debug register count
  - Hash table size

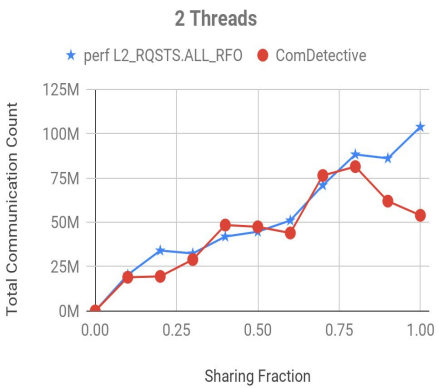
Communication Volume Verification

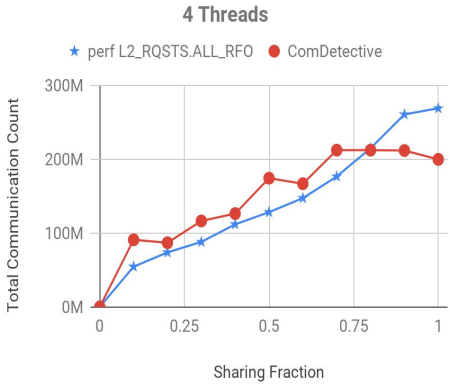
- Communication volume verification microbenchmark
  - Each thread performs **only store operations** to either **shared data** or **private data** depending on **sharing fraction parameter.**

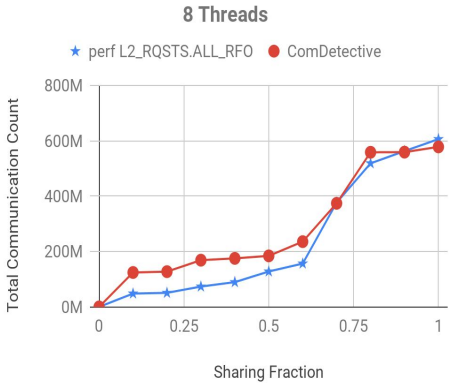
```
1  #pragma omp parallel shared(sharedData) private(privateData) \
2   num_threads(nThreads)
3  {
4    for(int i = 0 ; i < N_ITER; i++) {
5      int rNum  = rand_r(); // thread private
6      If (rNum < SHARING_FRACTION)  {
7        sharedData = rNum;
8      } else {
9        privateData = rNum;
10 }}}
```

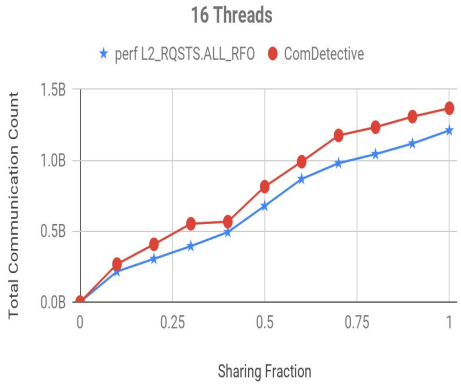
Listing 1: Write-Volume Benchmark

Communication Volume Verification





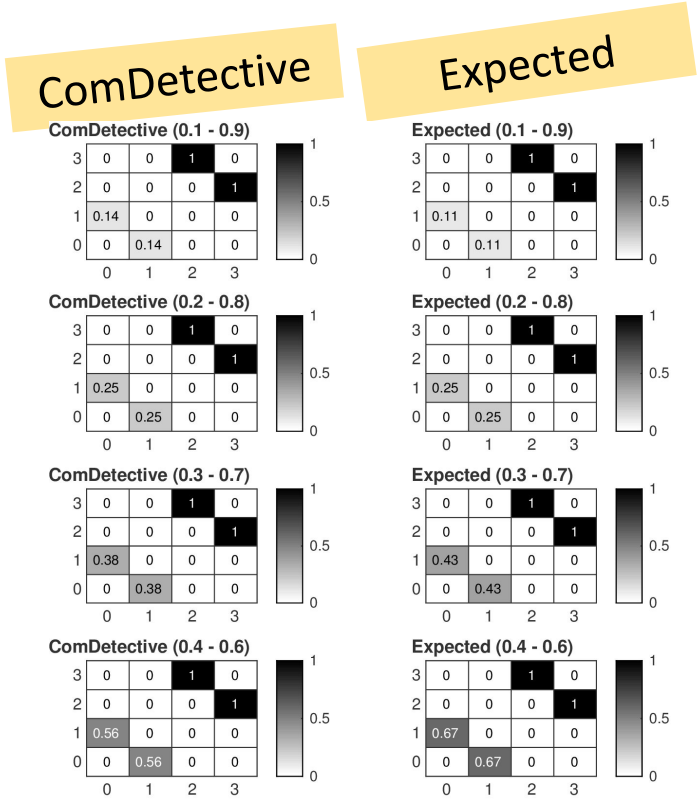




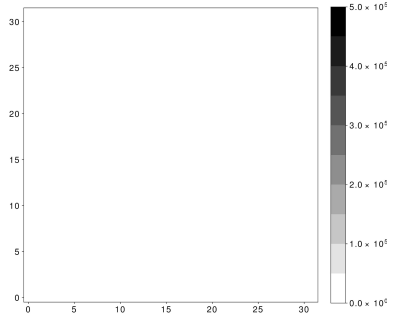
- ComDetective count vs **RFO** (request for ownership) count when 2-16 threads are mapped to 2 sockets.
- Each thread **only performs store operations to shared data**.
  - Real total communication count (ground truth) is RFO count

Point-to-point Communication Ratio Verification

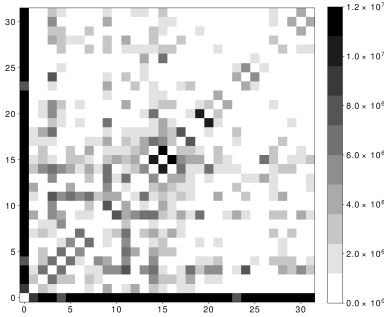
- Point-to-point communication microbenchmark
  - Enables selection of threads which communicate in pairs.
  - In all cases, thread 0 communicates with thread 1 and thread 2 communicates with thread 3.



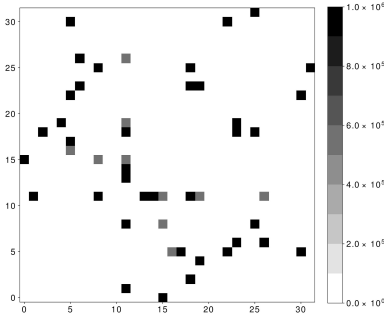
Snapshot of PARSEC Matrices (only 6 shown)



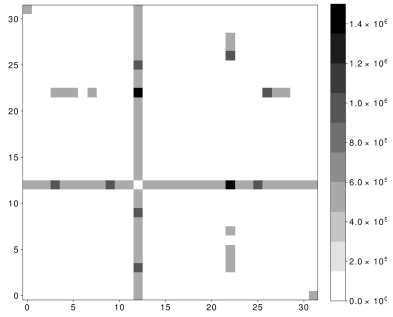
(a) Blackscholes



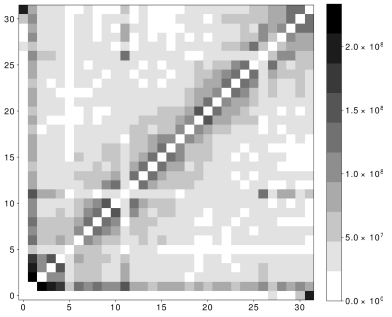
(b) Bodytrack



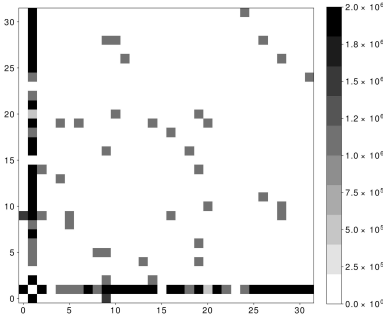
(c) Canneal



(d) Dedup

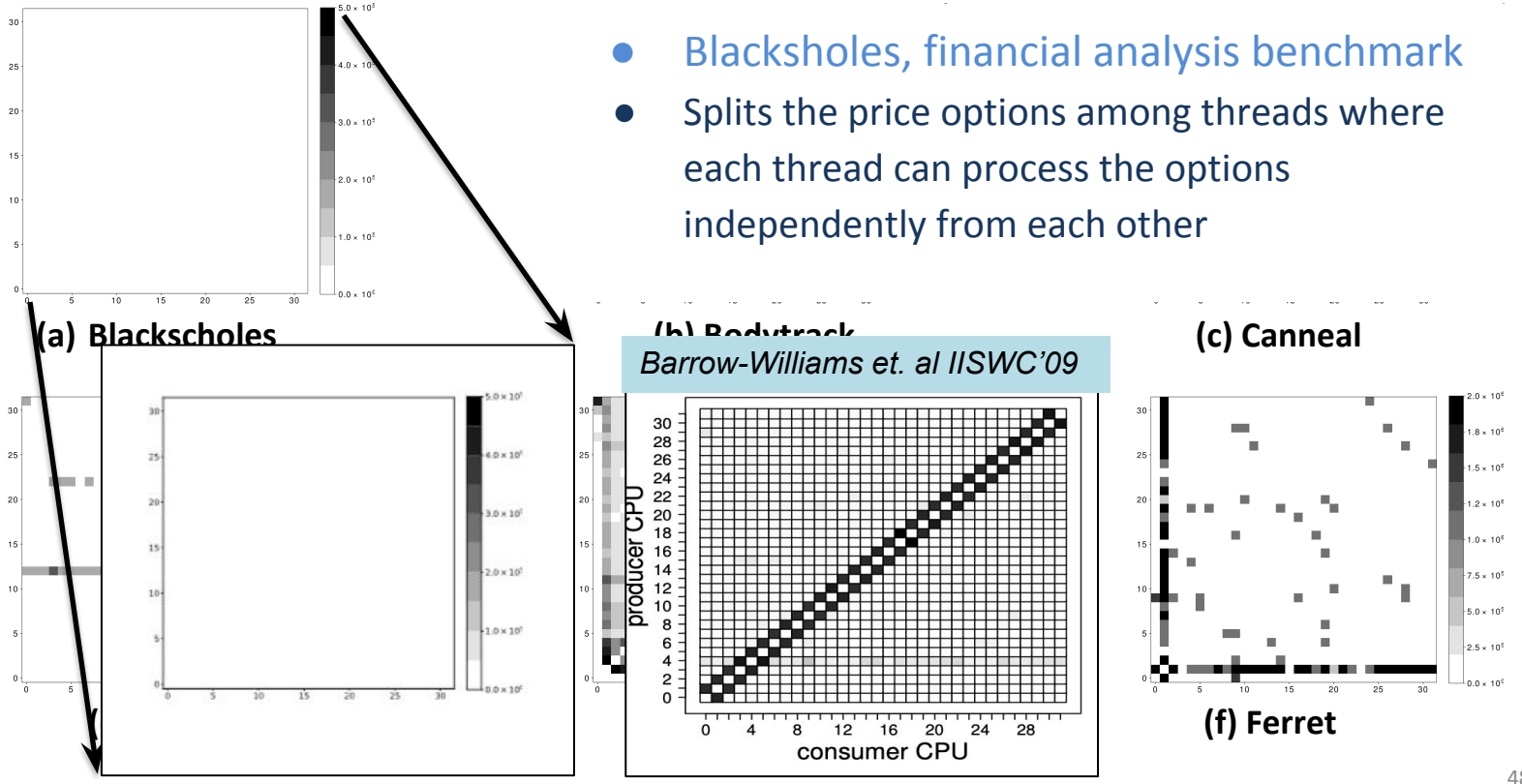


(e) Facesim



(f) Ferret

Snapshot of PARSEC Matrices (only 6 shown)



- Blacksholes, financial analysis benchmark
- Splits the price options among threads where each thread can process the options independently from each other

CORAL Benchmarks

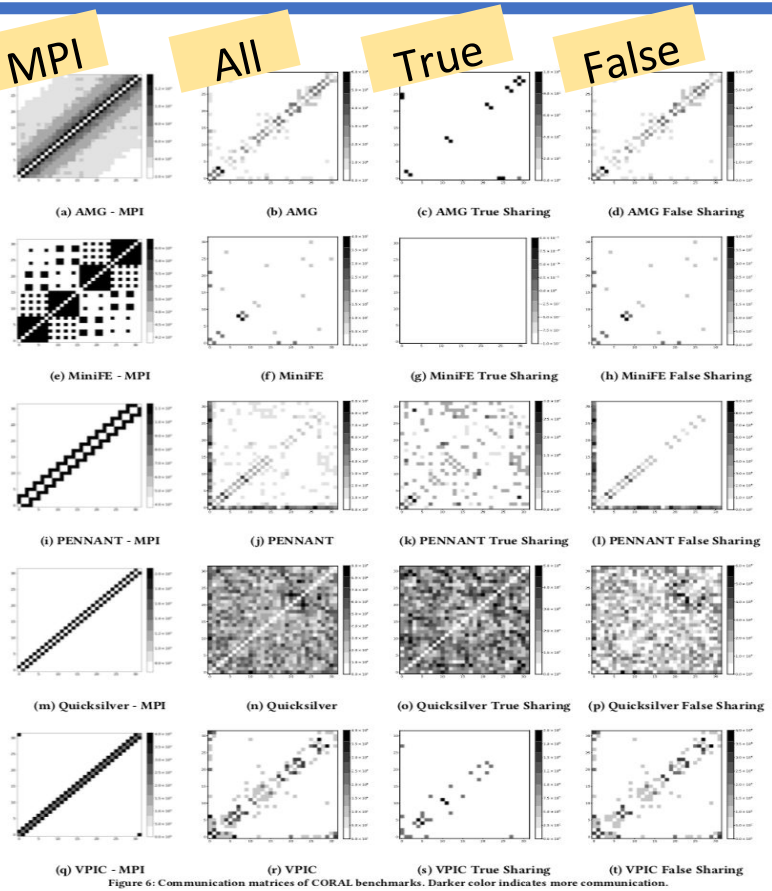


Figure 6: Communication matrices of CORAL benchmarks. Darker color indicates more communication.

AMG

MiniFE

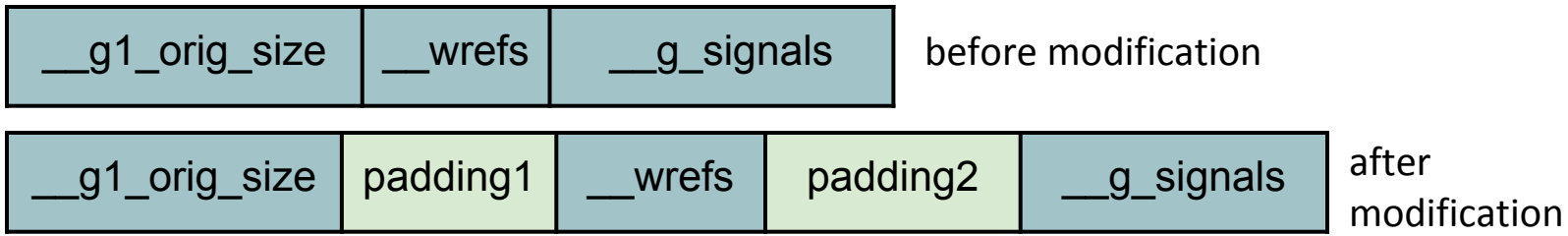
PENNANT

QuickSilver

VPIC

Use Cases: Code Refactoring

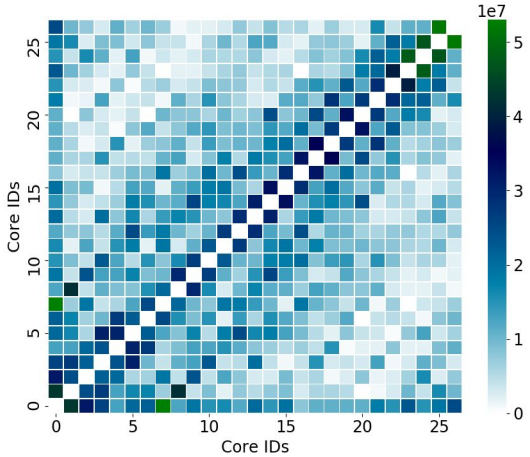
- **False sharing** in **streamcluster** happens on **pthread_mutex_t** typed variables
  - **6% improvement** is achieved after we put paddings among attributes in pthread_mutex_t struct
- **False sharing** in **fluidanimate** happens on a **pthread_cond_t** typed variable
  - **13% improvement** is achieved after we put paddings among attributes in pthread_cond_t struct



Summary

A practical tool for capturing inter-thread communication

- Low overhead: 1.27x runtime and 1.3x memory
- High accuracy
- Ability to quantify communication
- Ability to distinguish true vs. false sharing
- Attribute communication to program objects

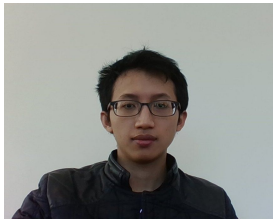


[Available for download: https://github.com/comdetective-tools]

ParCoreLab@ Koç

http://parcorelab.com

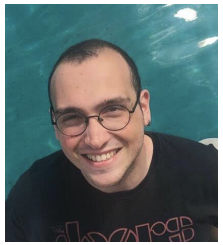
Optimization of Sparse Solvers



Detecting thread communication

Quantum computing

Accelerated Deep Learning



GPU Communication Optimization

References

[1] Nick Barrow-Williams, et al. 2009. A communication characterisation of Splash-2 and Parsec. In IEEE International Symposium on Workload Characterization, 2009. IISWC 2009.

[2] Eduardo Henrique Molina da Cruz, et al. 2011. Using Memory Access Traces to Map Threads and Data on Hierarchical Multi-core Platforms. In 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW).

[3] Matthias Diener, et al. 2016. Communication in Shared Memory: Concepts, Definitions, and Efficient Detection. In 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing.

[4] Reza Azimi, et al. 2009. Enhancing operating system support for multicore processors by using hardware performance monitoring. ACM SIGOPS Operating Systems Review 43, 2 (2009), 56–65.

[5] David Tam, et al. 2007. Thread clustering: sharing-aware scheduling on SMP-CMP-SMT multiprocessors. In Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007. 47–58.

[6] Eduardo H.M. Cruz, et al. 2012. Using the Translation Lookaside Buffer to Map Threads in Parallel Applications Based on Shared Memory. In 2012 IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS).

[7] Matthias Diener, et al. 2015. Characterizing communication and page usage of parallel applications for thread and data mapping. Performance Evaluation 88-89 (2015), 18–36.

[8] Matthias Diener, et al. 2016. Communication in Shared Memory: Concepts, Definitions, and Efficient Detection. In 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing.

References

[9] Arya Mazaheri, Felix Wolf, and Ali Jannesari. 2015. Characterizing Loop-Level Communication Patterns in Shared Memory Applications. In Proceedings of the 2015 44th International Conference on Parallel Processing (ICPP 2015). https://doi.org/10.1109/ICPP.2015.85

[10] Arya Mazaheri, Felix Wolf, and Ali Jannesari. 2018. Unveiling Thread Communication Bottlenecks Using Hardware-Independent Metrics. In Proceedings of the 47th International Conference on Parallel Processing (ICPP 2018). ACM, New York, NY, USA, Article 6, 10 pages. https://doi.org/10.1145/3225058.3225142