Contributions to high-performance memory hierarchies: program characterization, resource control, transactional synchronization, and hardware prefetching

Agustín Navarro Torres

Supervisors: Pablo Enrique Ibáñez Marín & Jesús Alastruey Benedé

April 19, 2023

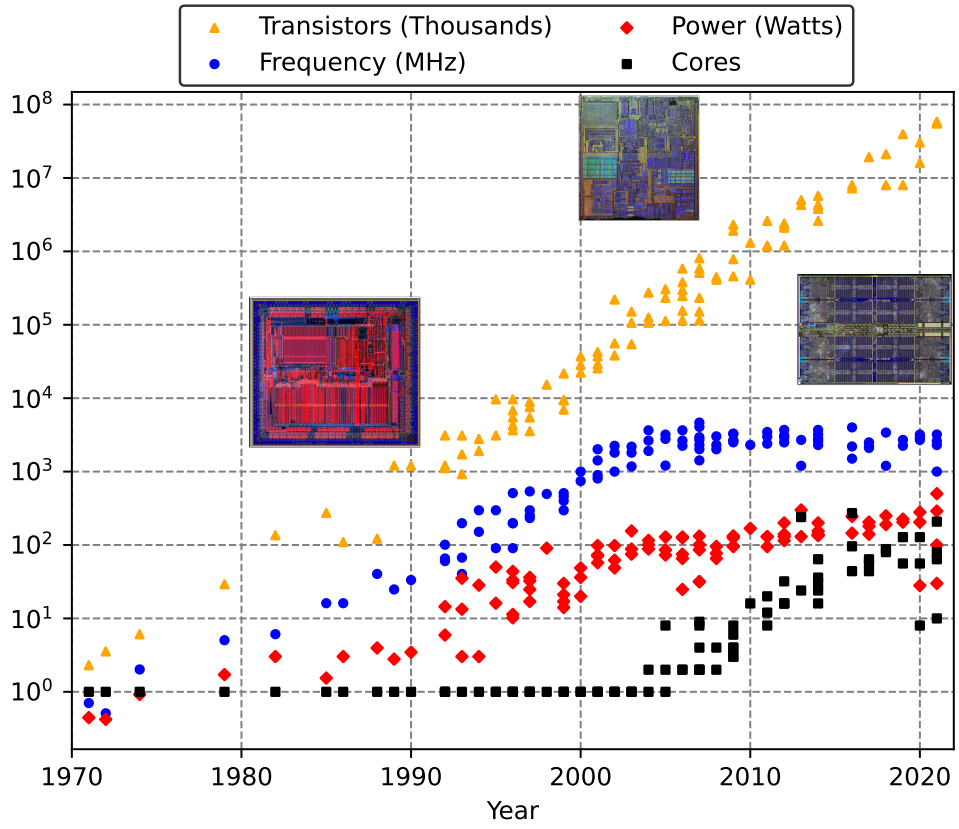
Universidad Zaragoza 1542

Grupo de Investigación en Arquitectura de Computadores (gaZ)
Universidad Zaragoza 1542

Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Instituto Universitario de Investigación de Ingeniería de Aragón
Universidad Zaragoza

Computer Processor History



MIPS R2K	Intel Pentium 4	AMD Epyc
1986	2000	2017
110K	55M	40B
200MHz	3.8GHz	3.9GHz
1	1	64

https://www.csl.cornell.edu/courses/ece4750/2016f/handouts/ece4750-T01-proc-concepts.pdf

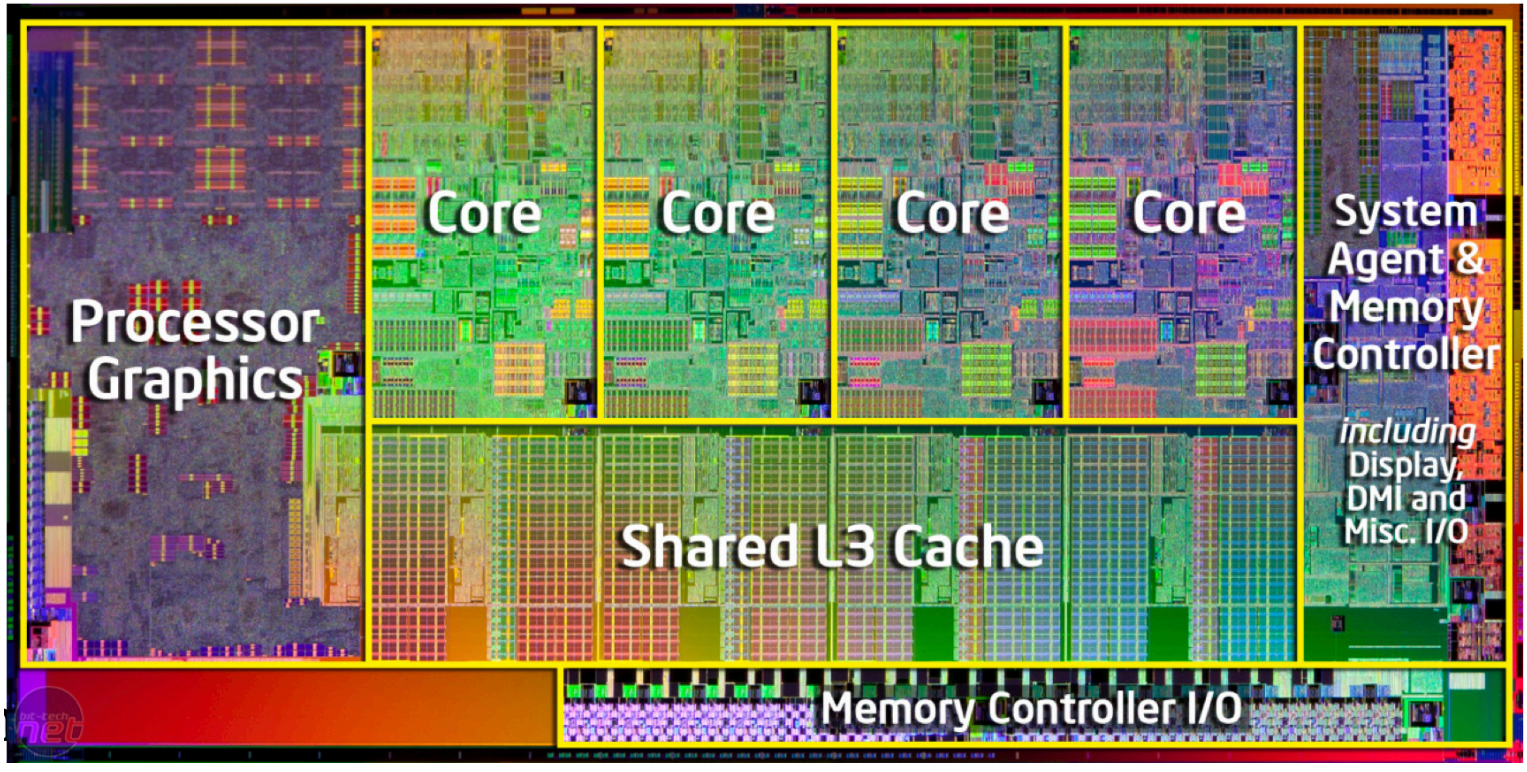
https://en.wikipedia.org/wiki/Pentium_4

https://www.flickr.com/photos/130561288@N04/49045449908

https://github.com/karlrupp/microprocessor-trend-data

Chip Multiprocessor

Intel Core i7, Ivy Bridge

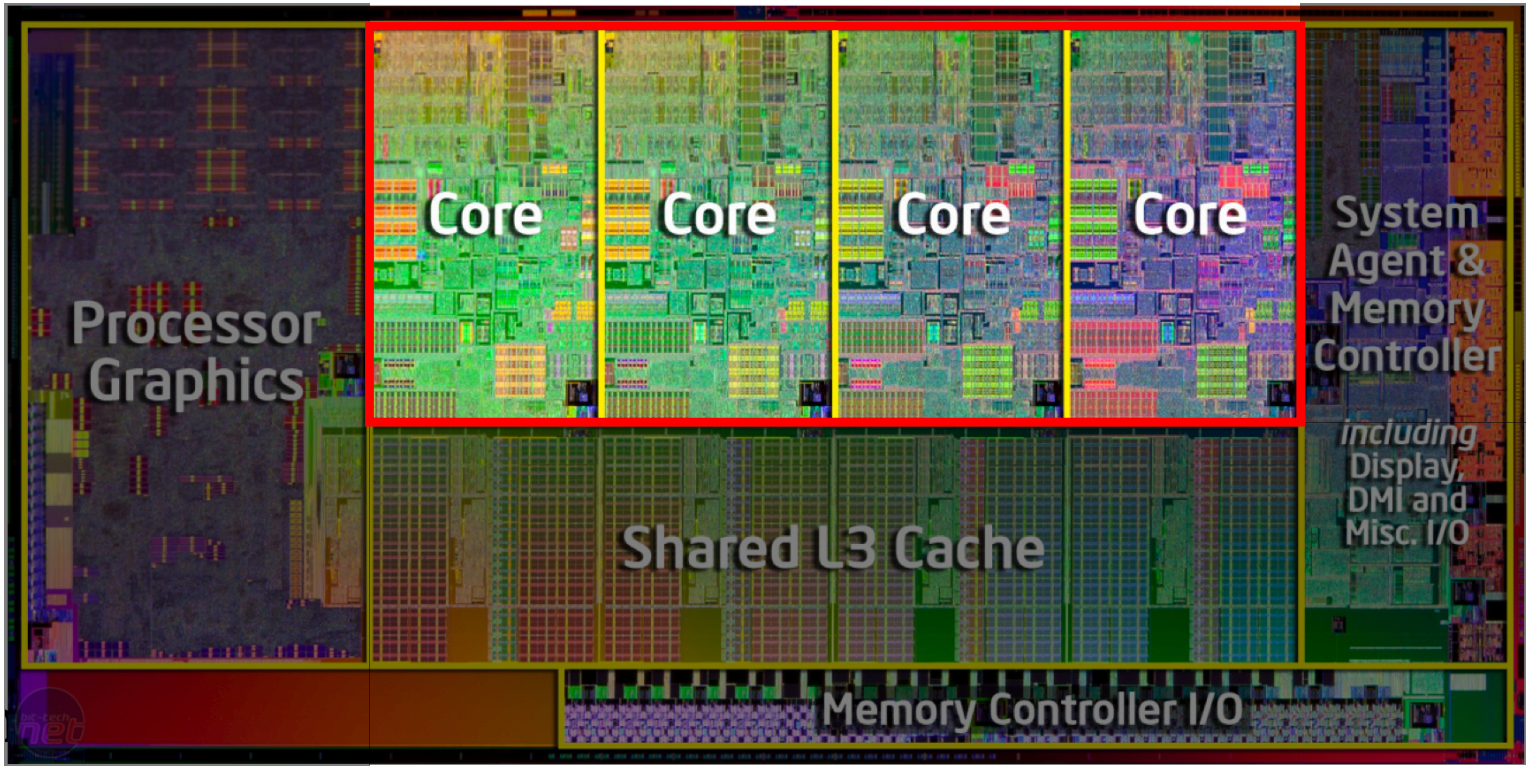


https://www.inf.ed.ac.uk/teaching/courses/pa/Notes/lecture10-multicores.pdf

Chip Multiprocessor

Multiple cores with private resources

Intel Core i7, Ivy Bridge

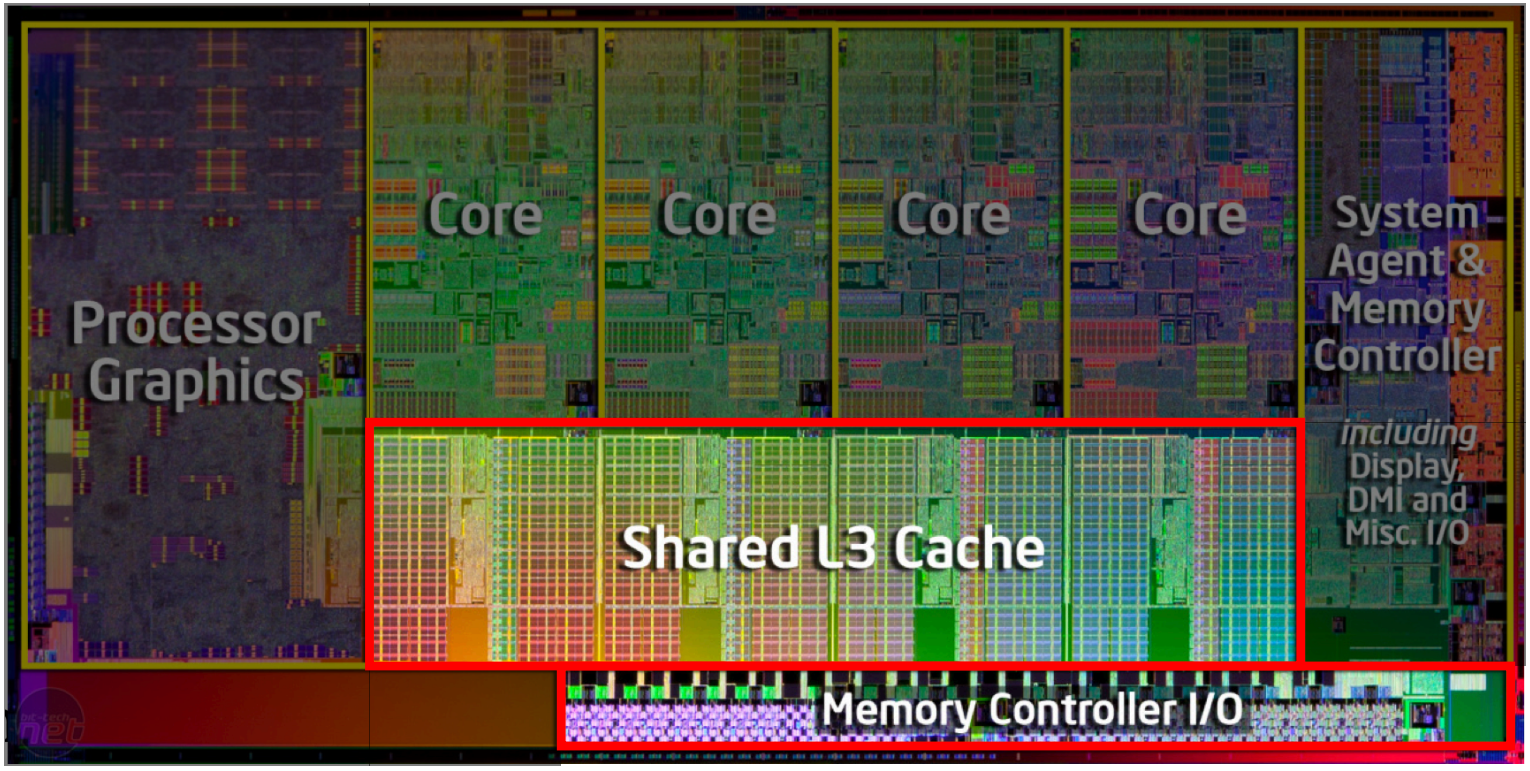


https://www.inf.ed.ac.uk/teaching/courses/pa/Notes/lecture10-multicores.pdf

Chip Multiprocessor

Multiple cores with shared resources

Intel Core i7, Ivy Bridge



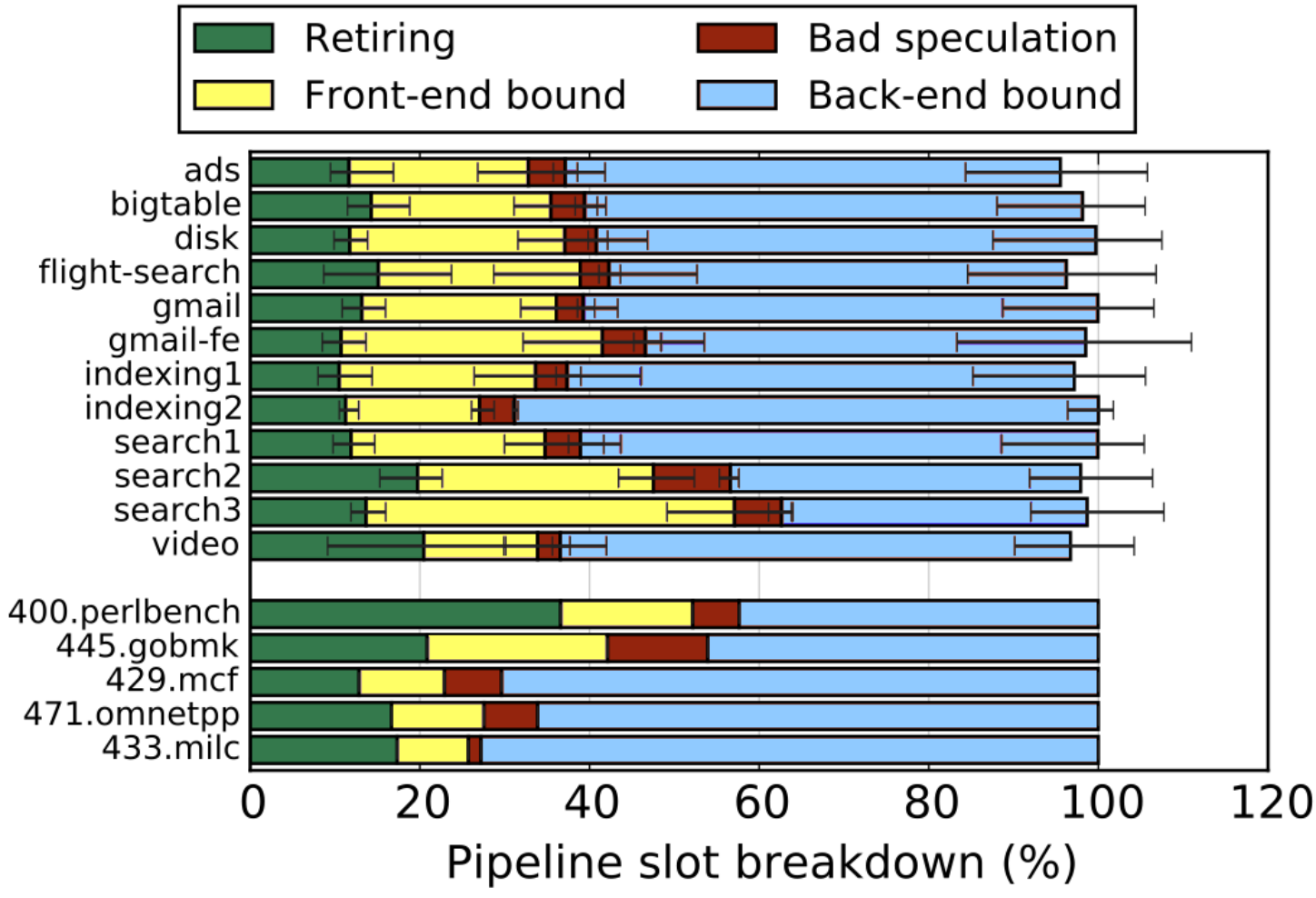
https://www.inf.ed.ac.uk/teaching/courses/pa/Notes/lecture10-multicores.pdf

Chip Multiprocessor

- Where are the memory bottlenecks?
- How can we improve the performance/fairness?

Chip Multiprocessor

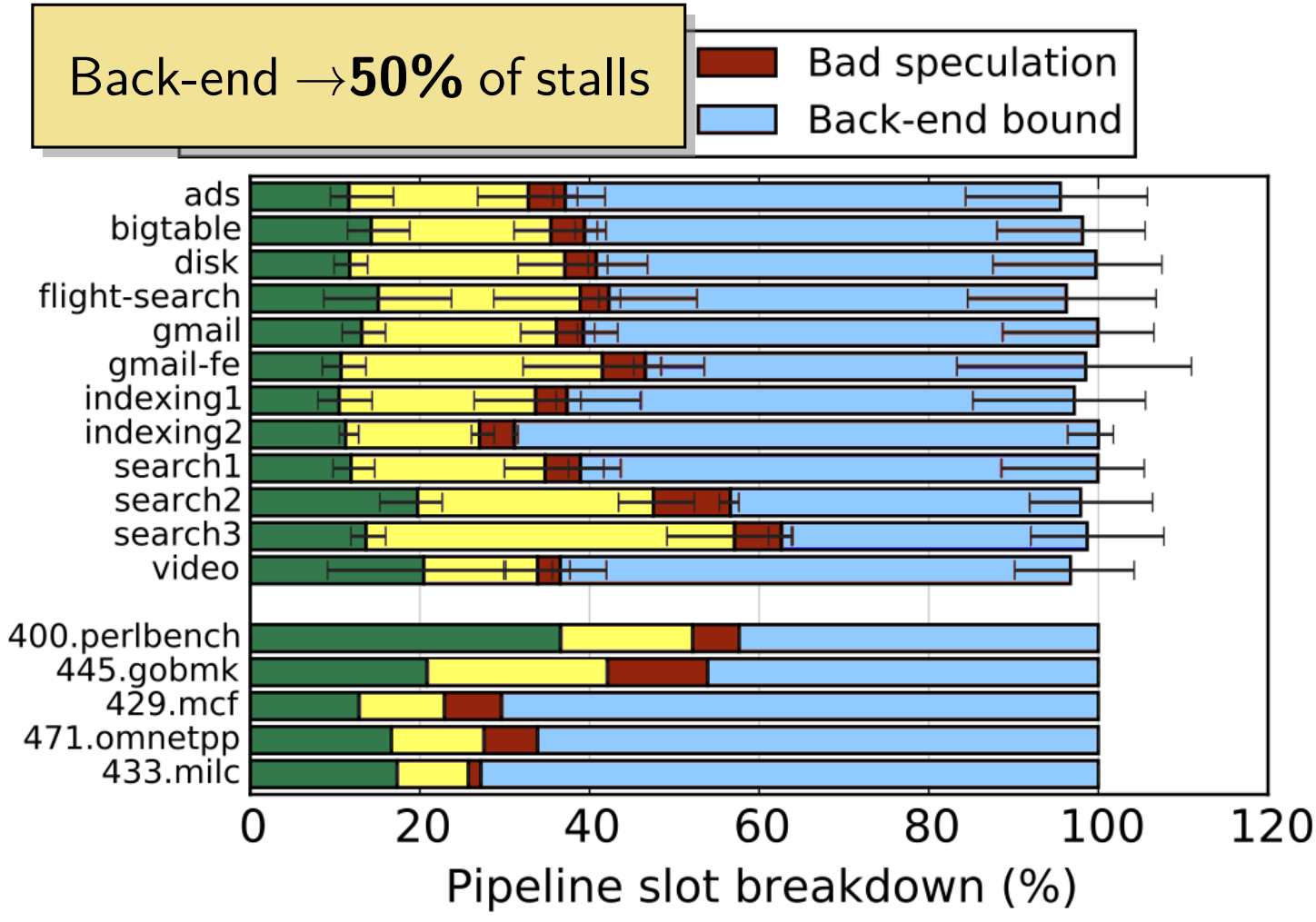
Where is the stall?



Kanev et. al. *Profiling a warehouse-scale computer*. ISCA 2015

Chip Multiprocessor

Where is the stall?



Kanev et. al. *Profiling a warehouse-scale computer*. ISCA 2015

Contributions

- *Memory hierarchy characterization of SPEC CPU2006 and SPEC CPU2017 on the Intel Xeon Skylake-SP*
- *BALANCER: Bandwidth Allocation and Cache Partitioning for Multicore Processors*
- *Synchronization Strategies on Many-Core SMT Systems*
- *Berti: an Accurate Local-Delta Data Prefetcher*

Contributions

- *Memory hierarchy characterization of SPEC CPU2006 and SPEC CPU2017 on the Intel Xeon Skylake-SP*
  - **Characterization**
- *BALANCER: Bandwidth Allocation and Cache Partitioning for Multicore Processors*
- *Synchronization Strategies on Many-Core SMT Systems*
- *Berti: an Accurate Local-Delta Data Prefetcher*

Contributions

- *Memory hierarchy characterization of SPEC CPU2006 and SPEC CPU2017 on the Intel Xeon Skylake-SP*
  - **Characterization**
- *BALANCER: Bandwidth Allocation and Cache Partitioning for Multicore Processors*
  - **Resource management**
- *Synchronization Strategies on Many-Core SMT Systems*
- *Berti: an Accurate Local-Delta Data Prefetcher*

Contributions

- *Memory hierarchy characterization of SPEC CPU2006 and SPEC CPU2017 on the Intel Xeon Skylake-SP*
  - **Characterization**
- *BALANCER: Bandwidth Allocation and Cache Partitioning for Multicore Processors*
  - **Resource management**
- *Synchronization Strategies on Many-Core SMT Systems*
  - **Synchronization strategies**
- *Berti: an Accurate Local-Delta Data Prefetcher*

Contributions

- *Memory hierarchy characterization of SPEC CPU2006 and SPEC CPU2017 on the Intel Xeon Skylake-SP*
  - **Characterization**
- *BALANCER: Bandwidth Allocation and Cache Partitioning for Multicore Processors*
  - **Resource management**
- *Synchronization Strategies on Many-Core SMT Systems*
  - **Synchronization strategies**
- *Berti: an Accurate Local-Delta Data Prefetcher*
  - **Data hardware prefetching**

Contributions

- *Memory hierarchy characterization of SPEC CPU2006 and SPEC CPU2017 on the Intel Xeon Skylake-SP*
  - **Characterization**
- *BALANCER: Bandwidth Allocation and Cache Partitioning for Multicore Processors*
  - **Resource management**
- *Synchronization Strategies on Many-Core SMT Systems*
  - **Synchronization strategies**
- *Berti: an Accurate Local-Delta Data Prefetcher*
  - **Data hardware prefetching**

Introduction

Memory hierarchy characterization of SPEC CPU2006 & CPU2017

Balancer: Bandwidth Allocation and Cache Partitioning

Synchronization strategies on many core systems

Berti: an accurate local delta data prefetcher

Conclusions and Future work

Memory hierarchy characterization

- How do the applications behave?
- How do the applications interact with the shared resources?

Memory hierarchy characterization

- How do the applications behave?
- How do the applications interact with the shared resources?
- We analyze SPEC CPU benchmarks on real system

System

Why real system?

- Full benchmark execution
- Capture behavior of state-of-the-art hardware components

System

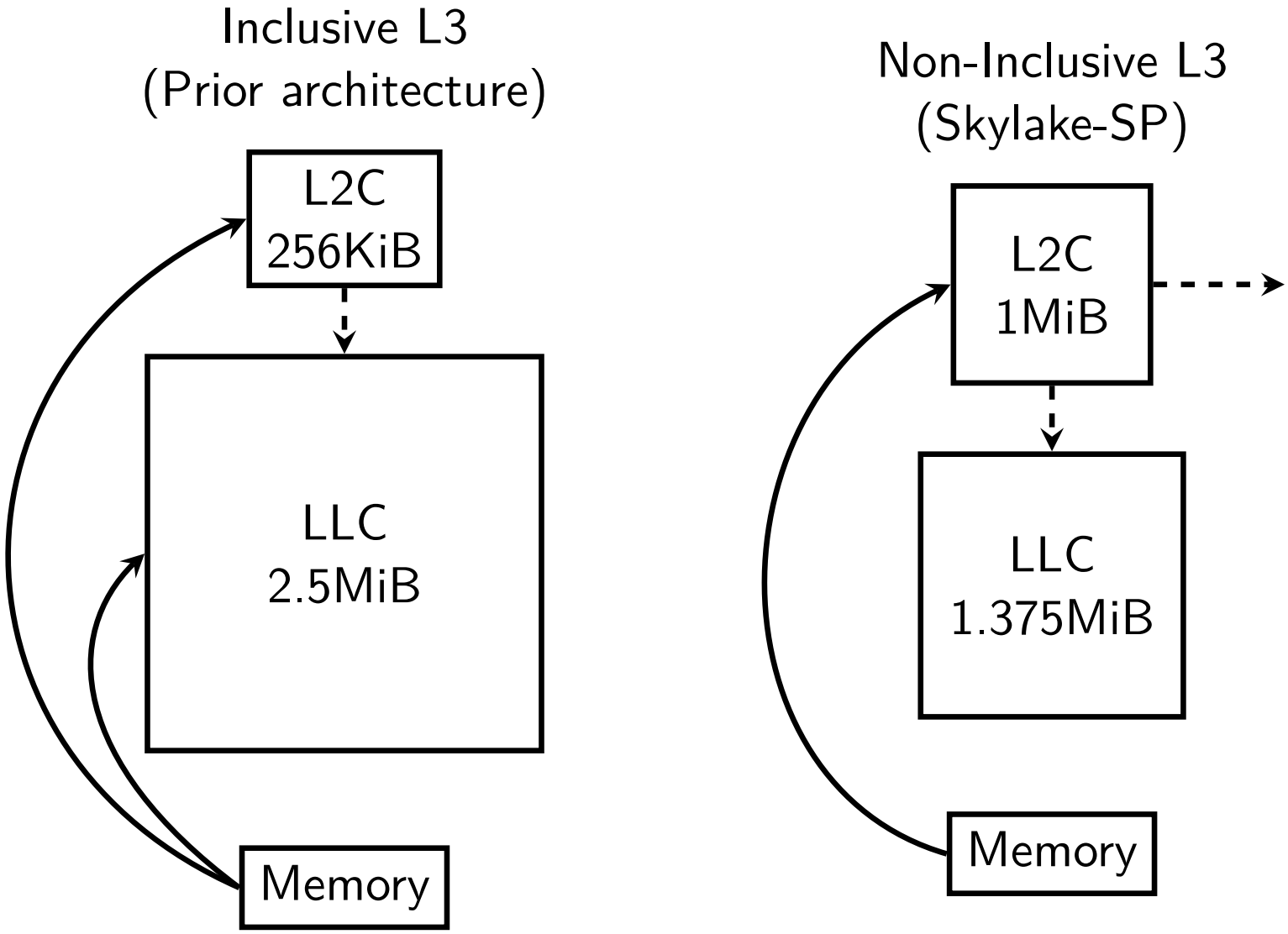
Why real system?

- Full benchmark execution
- Capture behavior of state-of-the-art hardware components
- We *cannot* change the hardware

Intel Xeon SKL

- 14 cores with shared LLC
- Four Hardware Prefetchers (2 L1D + 2 L2C)
  - L1D: IP-Stride and Next-Line
  - L2C: Adjacent cache line and Streamer
- Moving from inclusive SLLC to non-inclusive SLLC

Intel Xeon SKL



Intel Xeon SKL

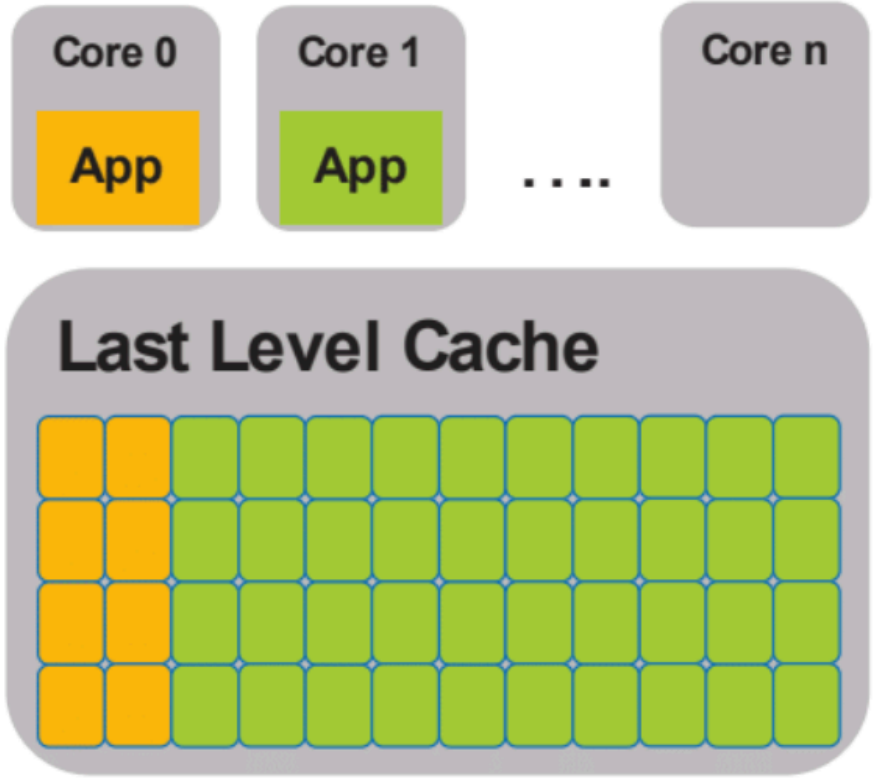
Intel Resource Director Technology (RDT)

- *"New levels of visibility and control over how shared resources [...] are used by applications"* [1]
- **SLLC** and memory bandwidth monitoring
- **SLLC** and memory bandwidth allocation

[1] https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html

Intel Xeon SKL

Intel Cache Allocation Technology (CAT)



A. Herdrich et al., *"Cache QoS: From concept to reality in the Intel® Xeon® processor E5-2600 v3 product family"*, HPCA 2016

Intel Xeon SKL

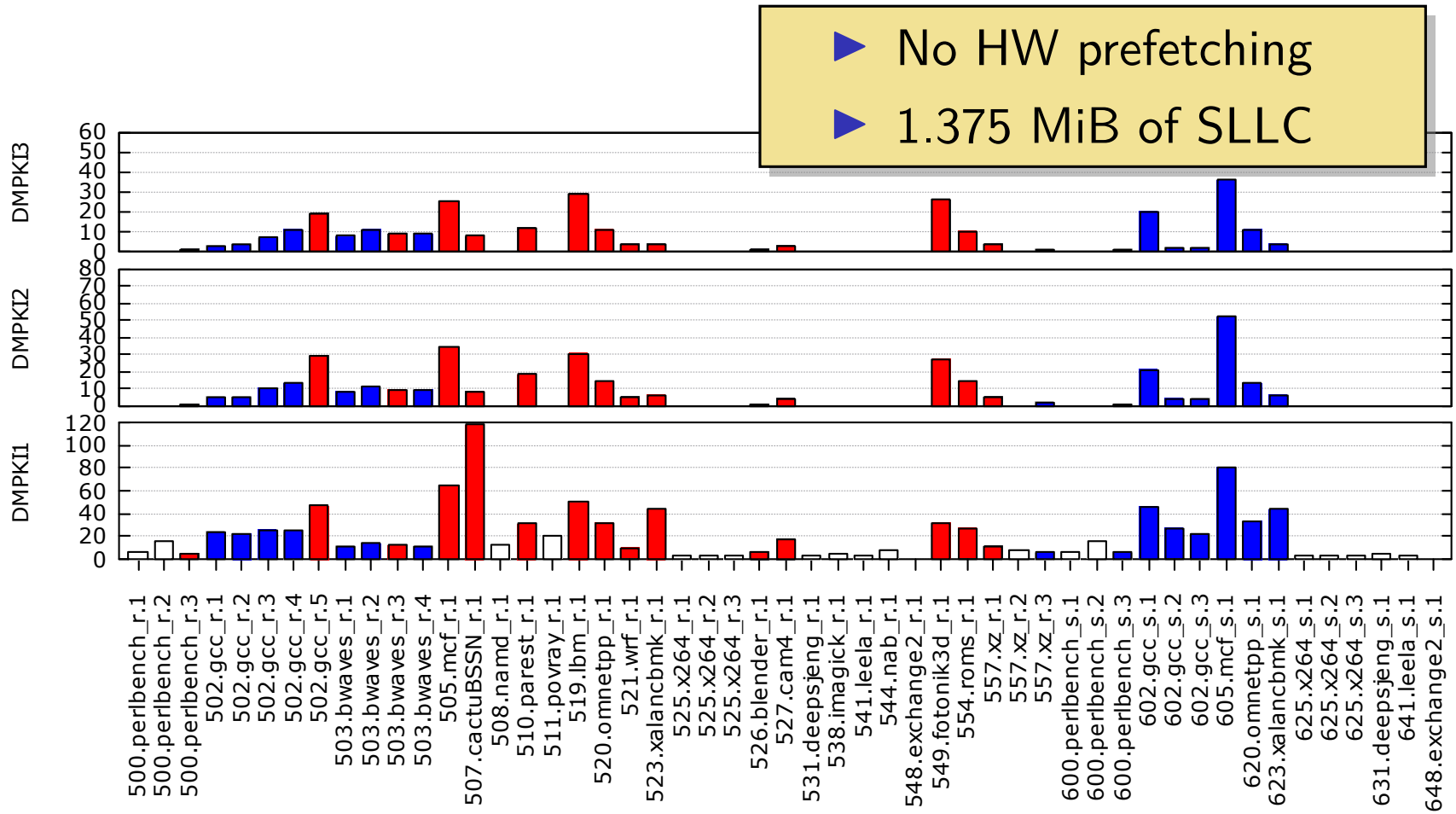
Hardware Counters

- Record events that happens during the execution
- Analyze the behavior and interaction of the applications

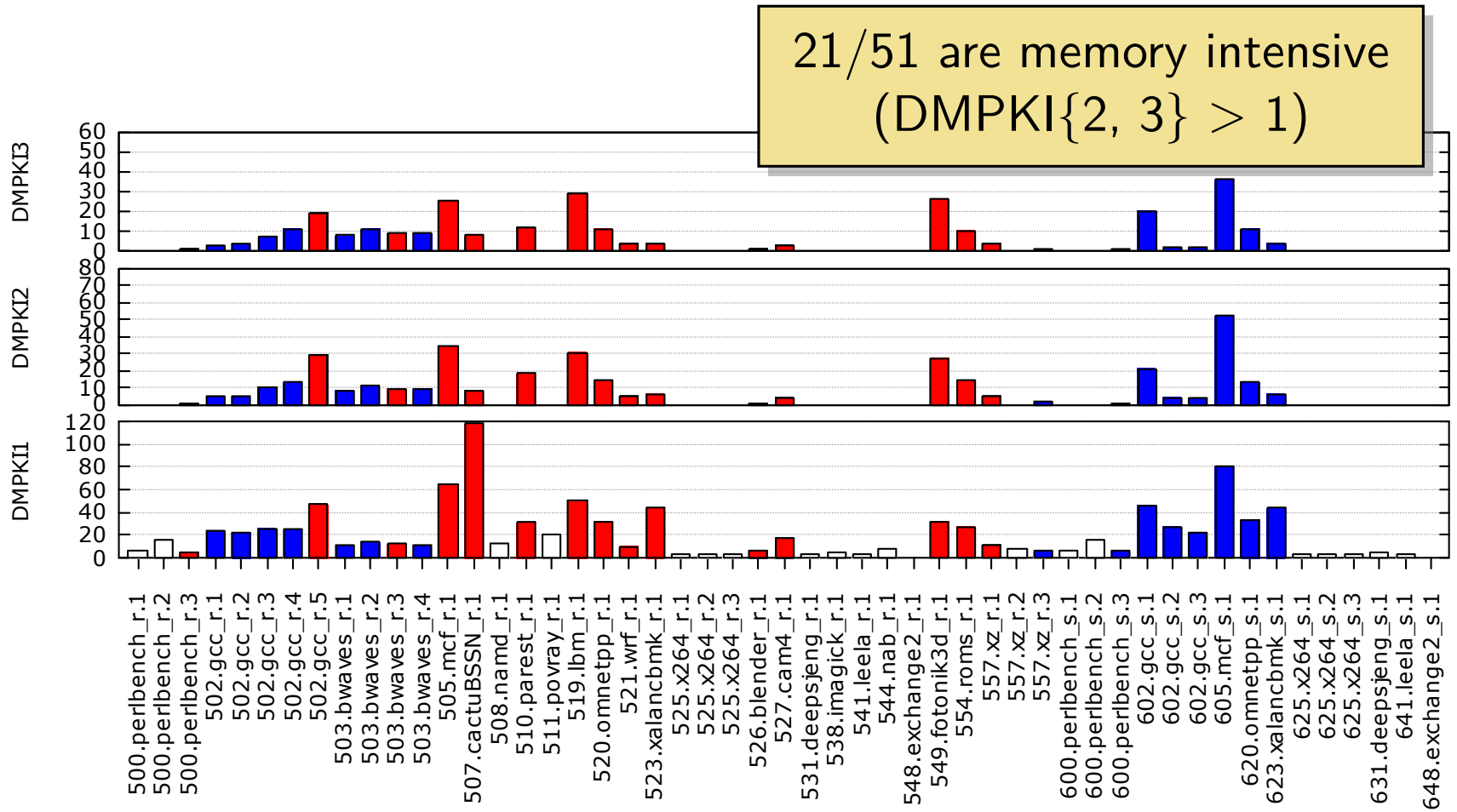
SPEC CPU2006 & CPU2017

- Most widely used benchmark suites for CPU & memory performance
- 106 benchmarks from 43 applications (17 integers and 26 FP)
- Only single-thread
- I will show only CPU2017

Which are the interesting benchmarks?



Which are the interesting benchmarks?



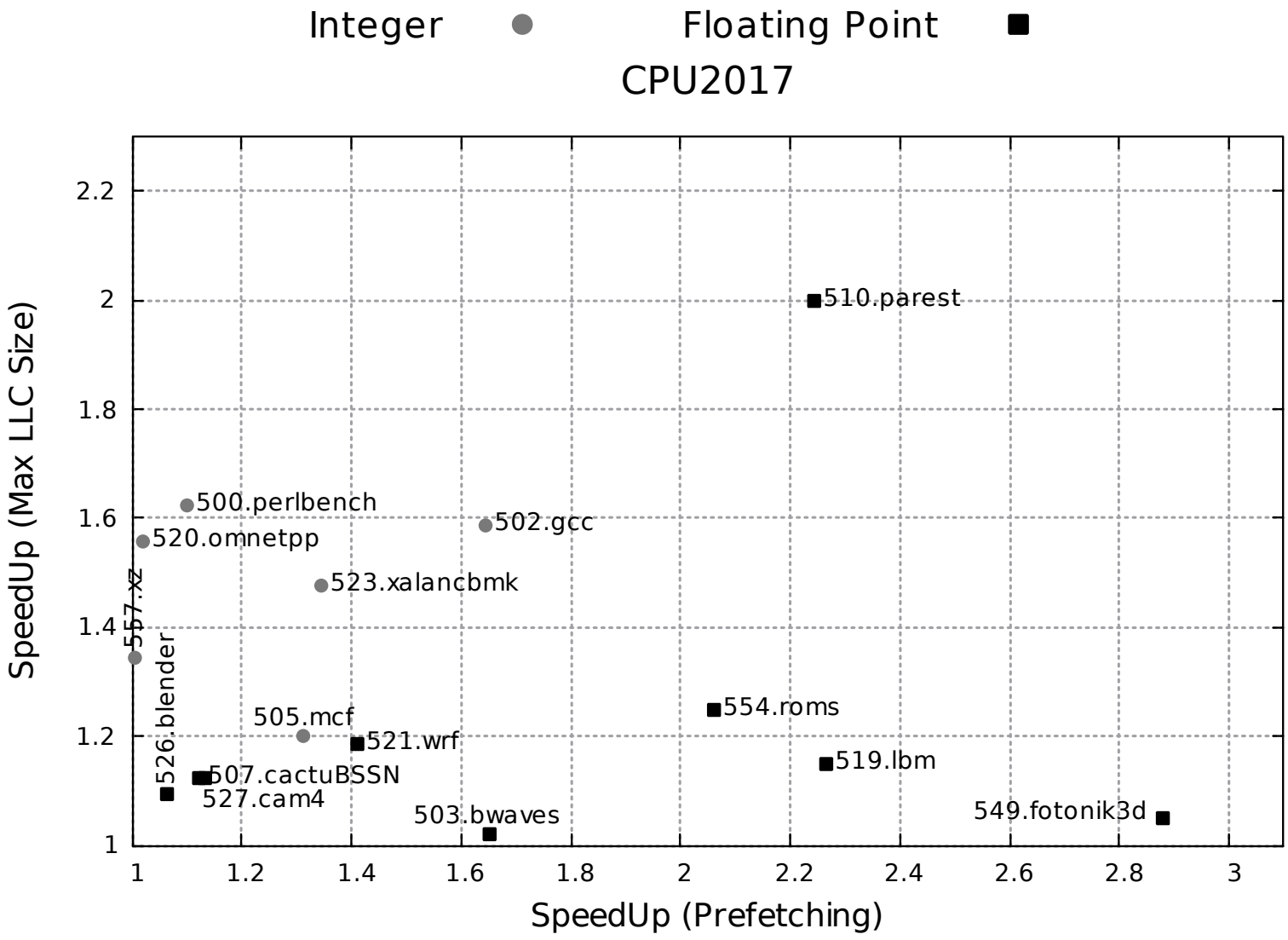
Sensitivity to LLC size & HW prefetching

- How do applications behave under different SLLC sizes?
- Is hardware prefetching efficient?

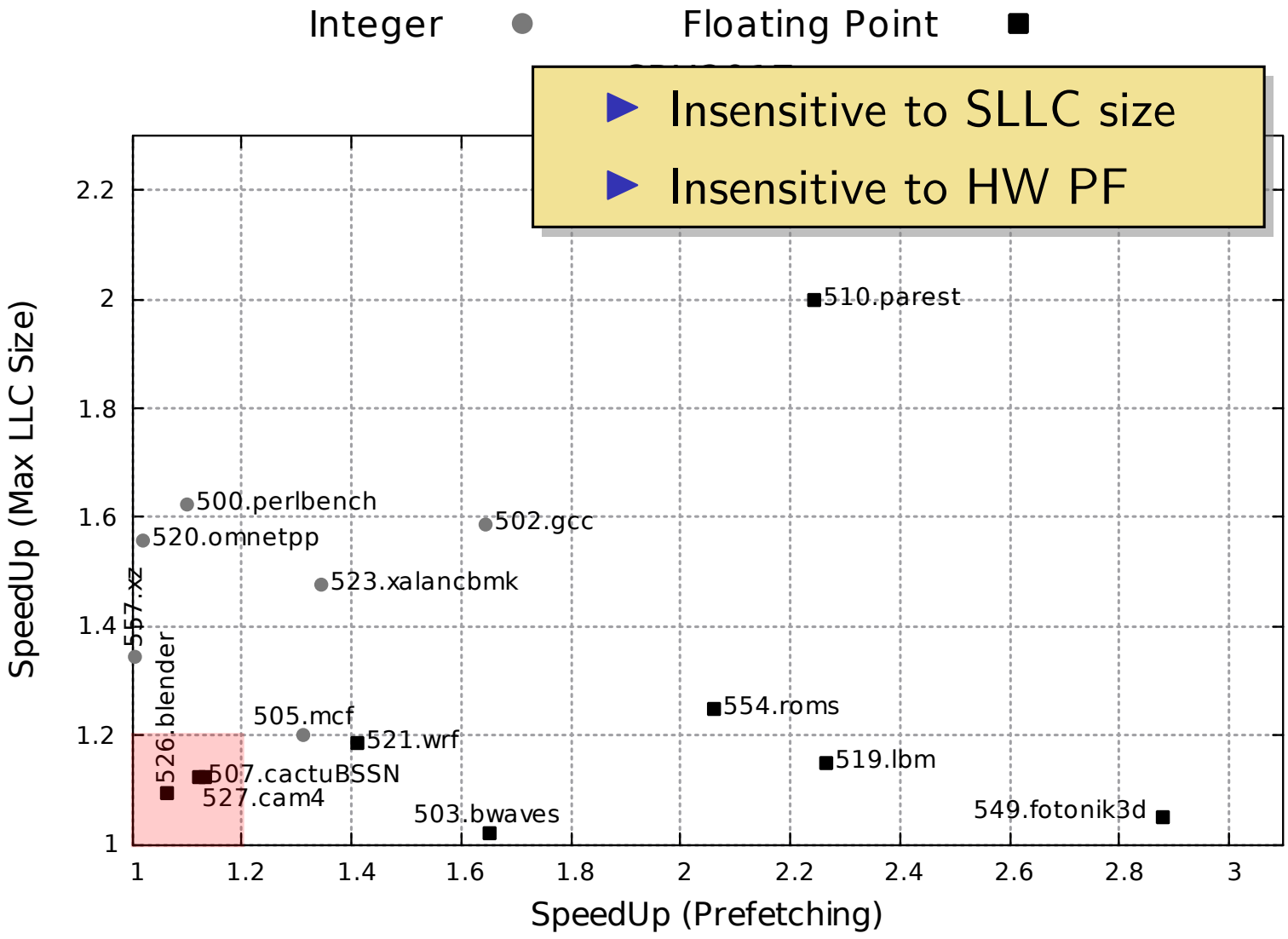
Sensitivity to LLC size & HW prefetching

- How do applications behave under different SLLC sizes?
- Is hardware prefetching efficient?
- Running each application:
  - From 1.375 MiB to 19.75 MiB of SLLC
  - With and without HW prefetching

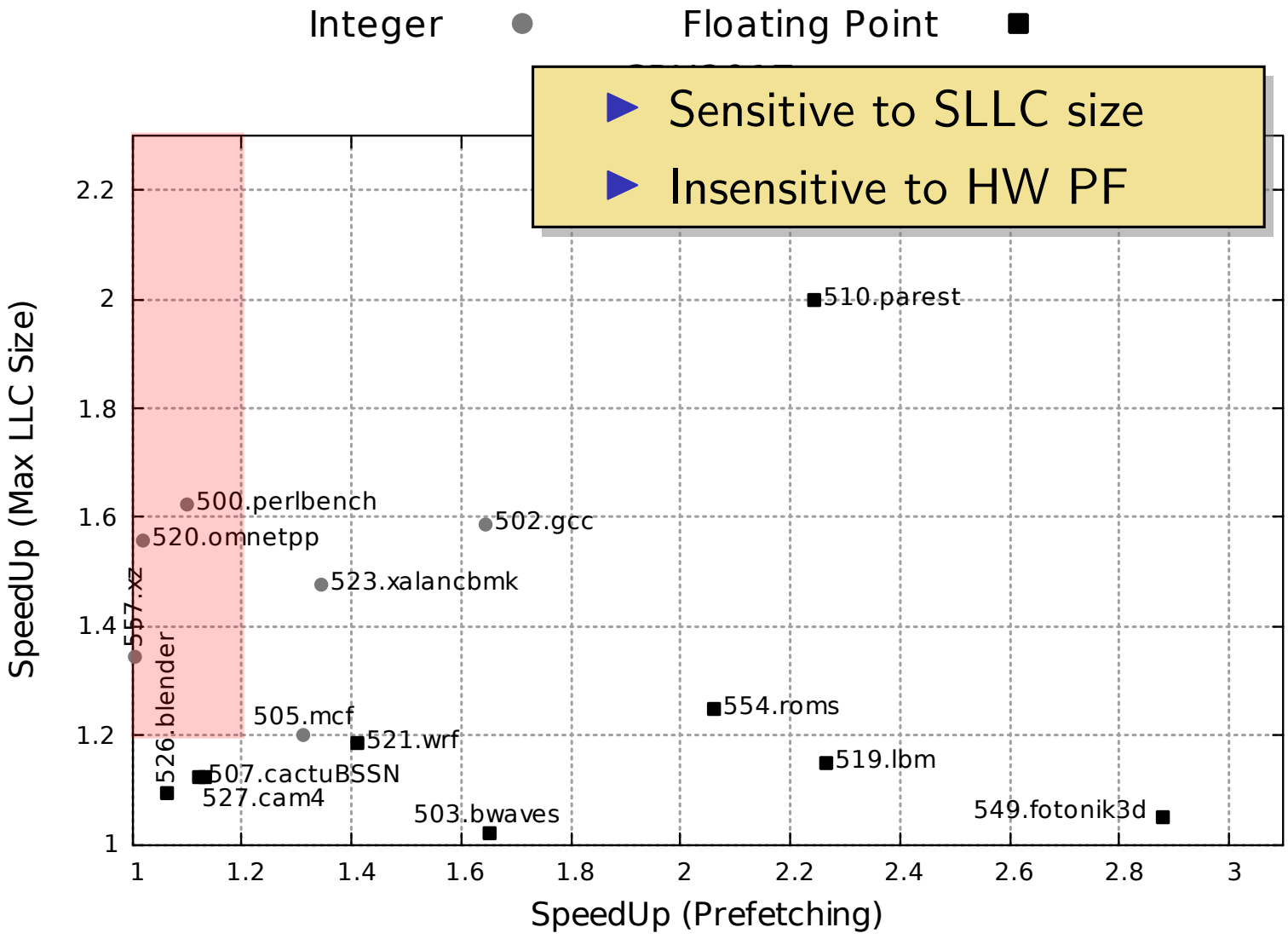
Sensitivity to LLC size & HW prefetching



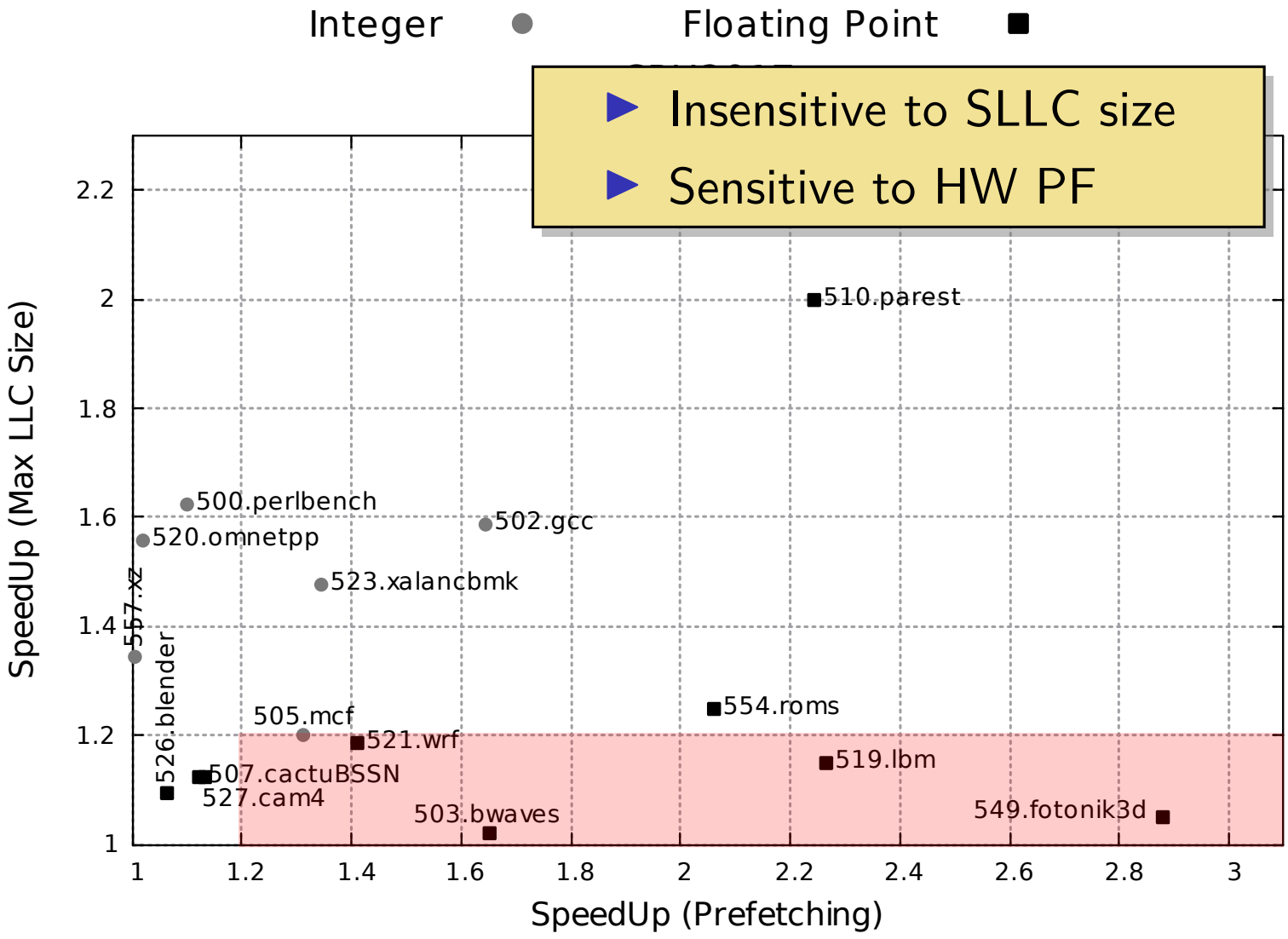
Sensitivity to LLC size & HW prefetching



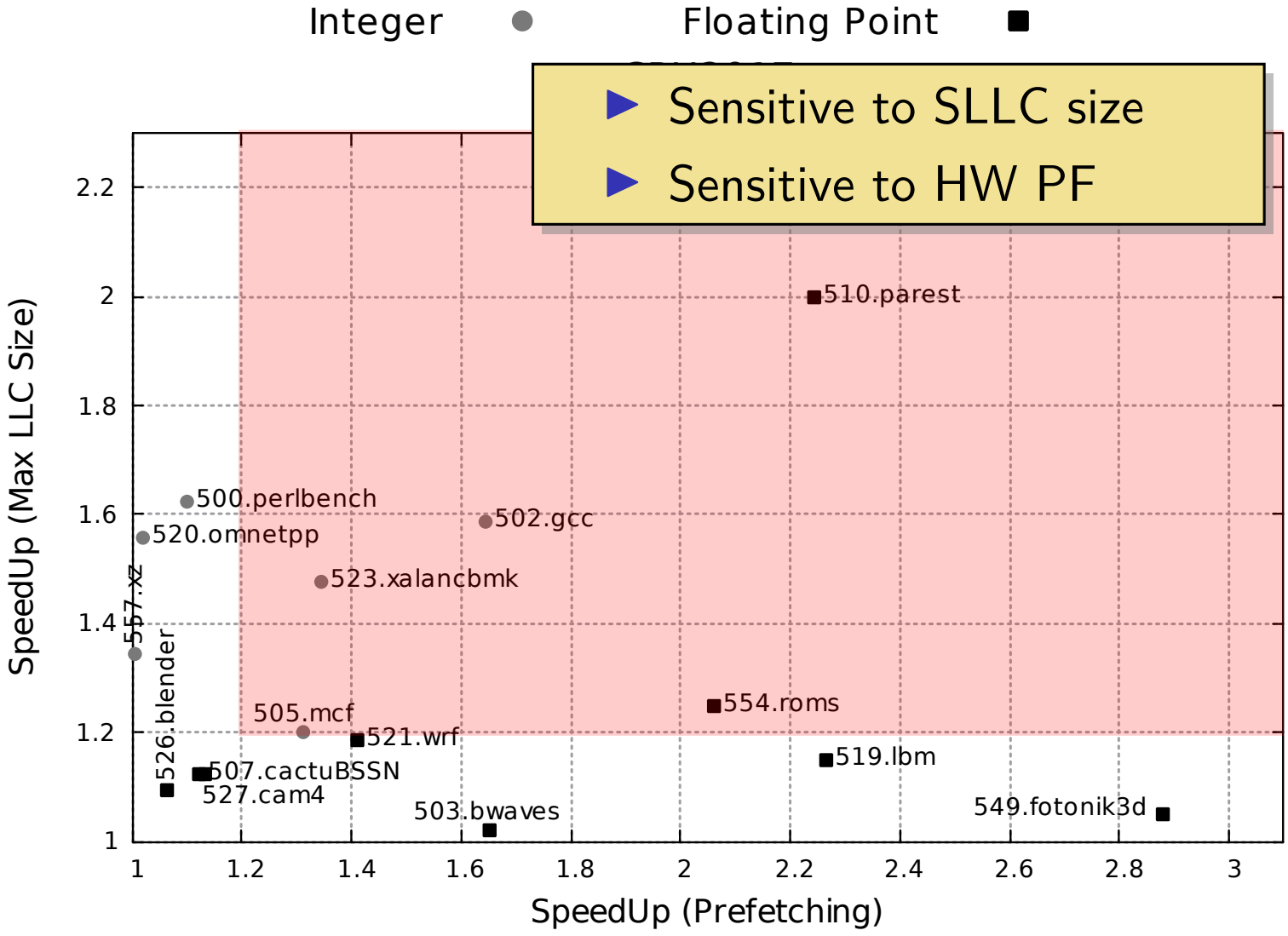
Sensitivity to LLC size & HW prefetching



Sensitivity to LLC size & HW prefetching



Sensitivity to LLC size & HW prefetching



Hardware prefetching performance

- How does each hardware prefetcher influence performance?
- Are hardware prefetchers accurate?

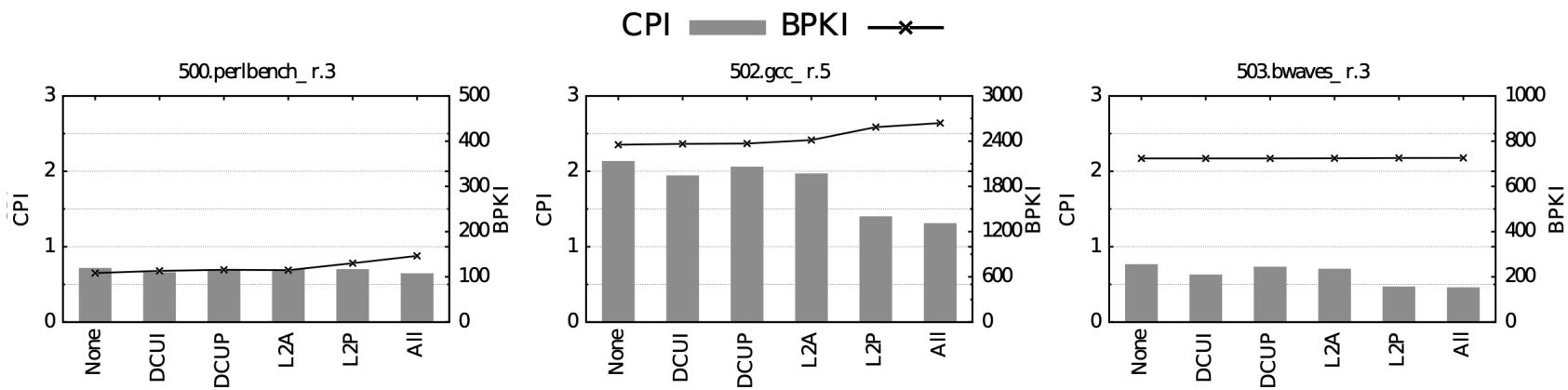
Hardware prefetching performance

- How does each hardware prefetcher influence performance?
- Are hardware prefetchers accurate?
- Run application with each prefetcher individually enabled

Hardware prefetching performance

CPI & BPKI

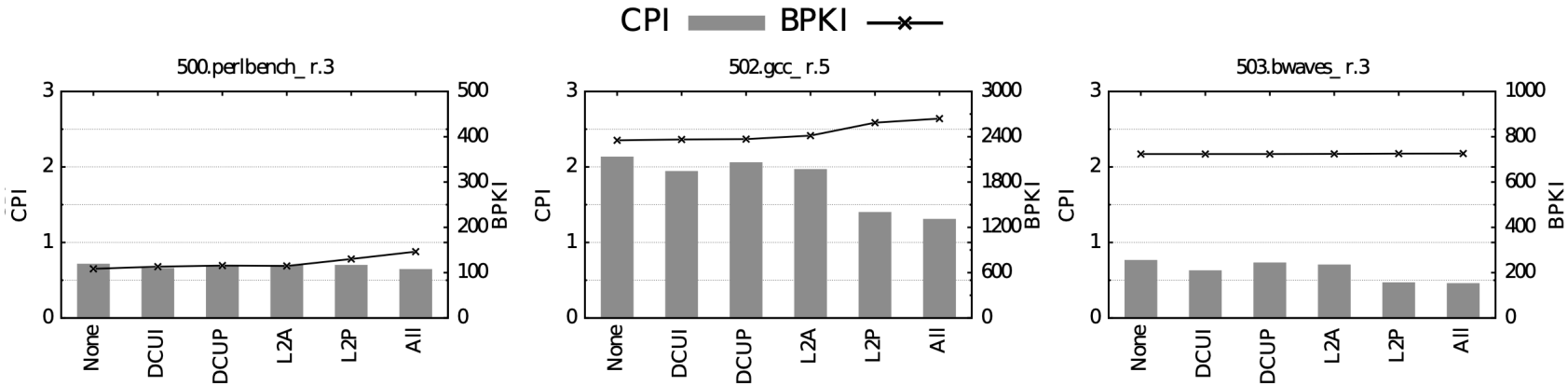
- **CPI**: Cycles per Instruction
- **BPKI**: Bytes from DRAM per Kilo Instruction



Hardware prefetching performance

CPI & BPKI

- Streamer prefetch (L2P) achieves 82% of CPI reduction
- Hardware prefetching is pretty accurate



Conclusion remarks

1. SLLC size improves performance for many benchmarks
2. Hardware prefetching boost performance for most of the benchmarks
3. Hardware prefetching has high accuracy

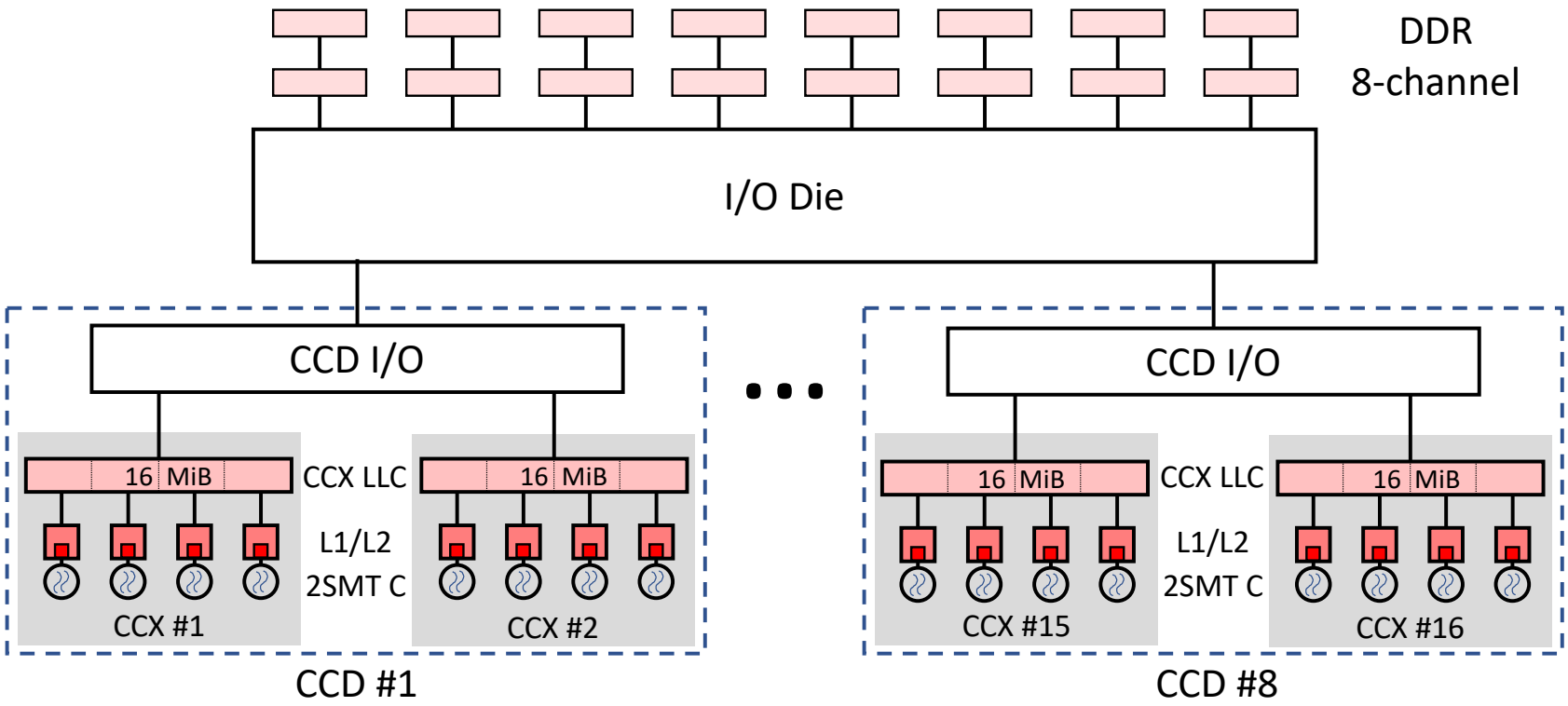
Introduction

- More cores →More pressure on shared resources
- Manage shared resources is key to performance/fairness

Introduction

- More cores →More pressure on shared resources
- Manage shared resources is key to performance/fairness
- New hardware support to limit SLLC and off-chip BW by thread
- Can we improve system performance and fairness?

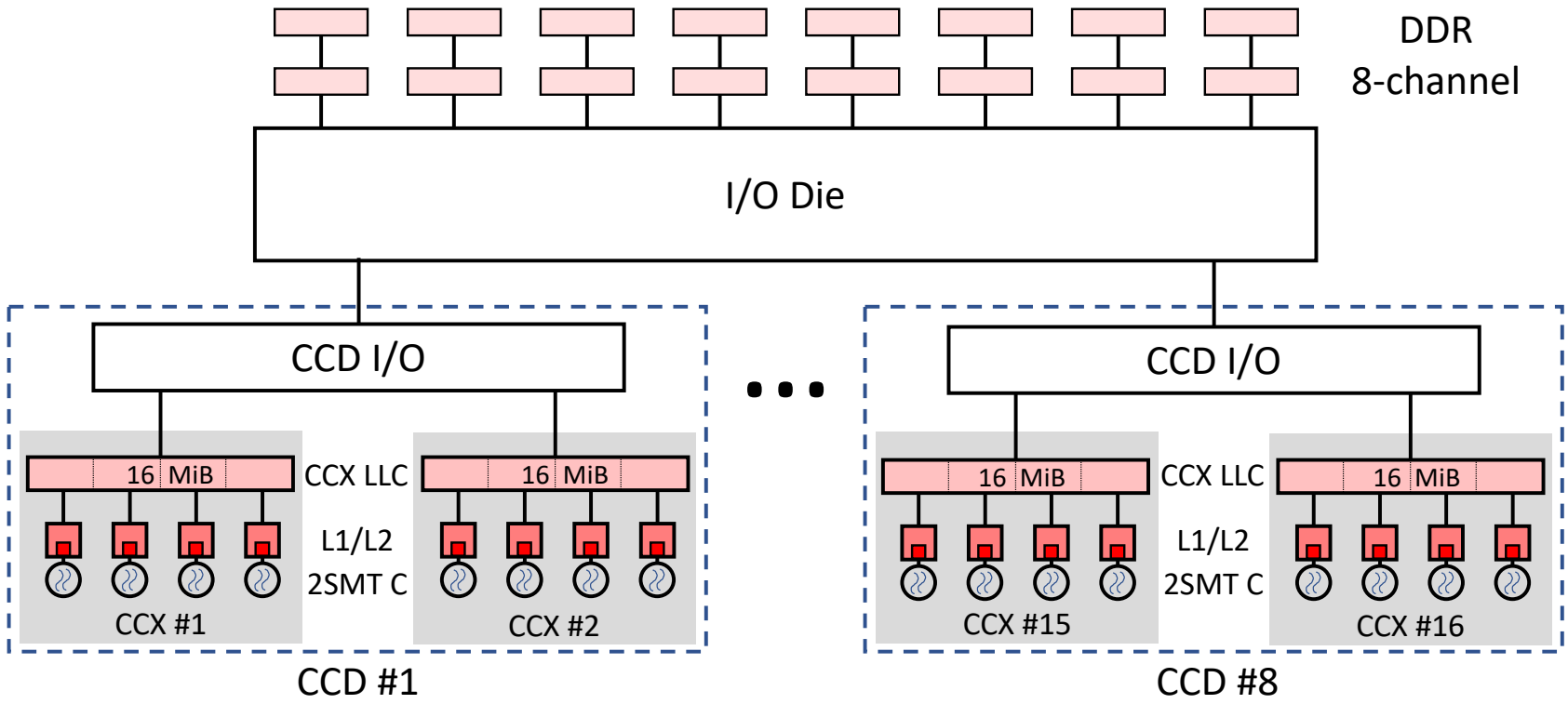
AMD EPYC Rome



AMD EPYC Rome

64 cores grouped by:

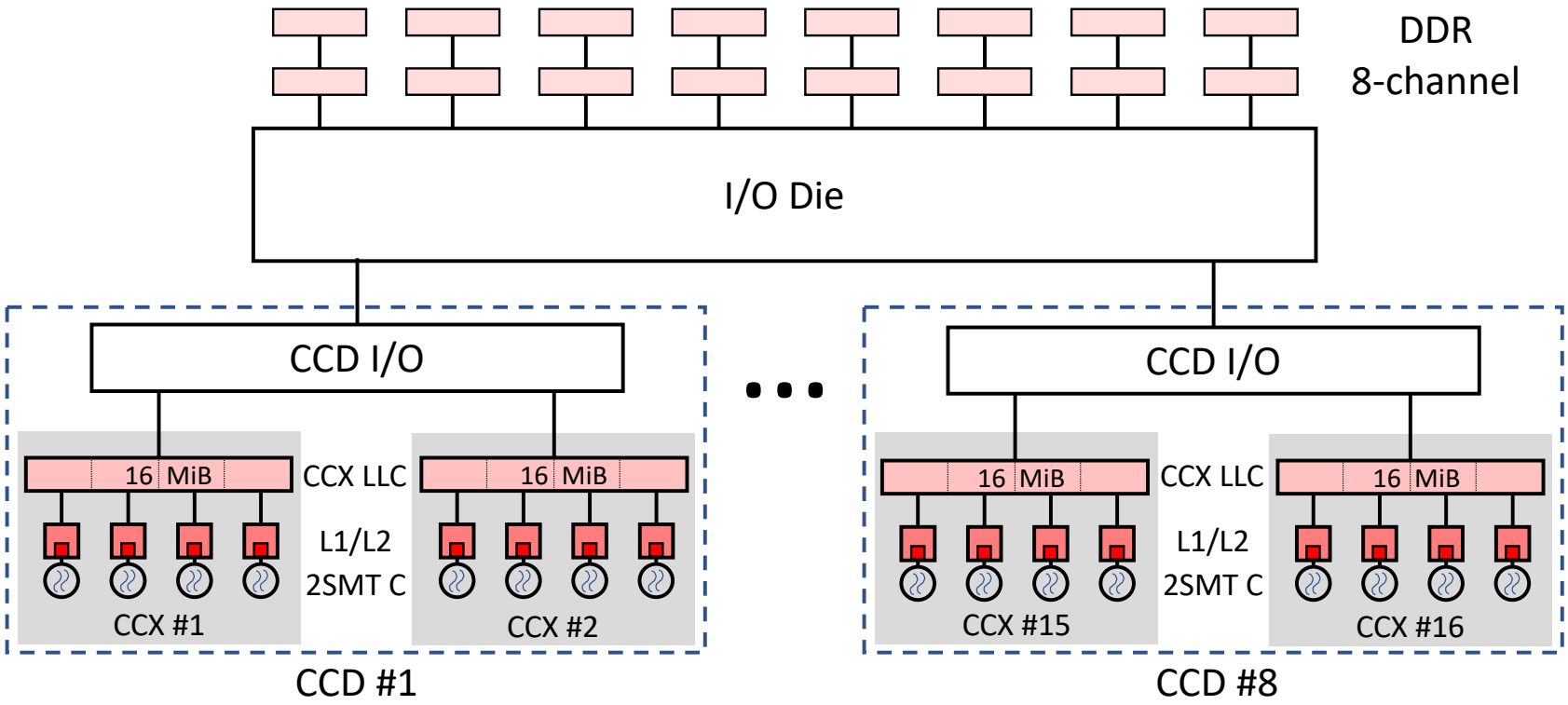
- CCD: 8 Cores, shared DRAM BW
- CCX: 4 Cores, 16 MiB of SLLC



AMD EPYC Rome

Clustered SLLC seems to be a trend

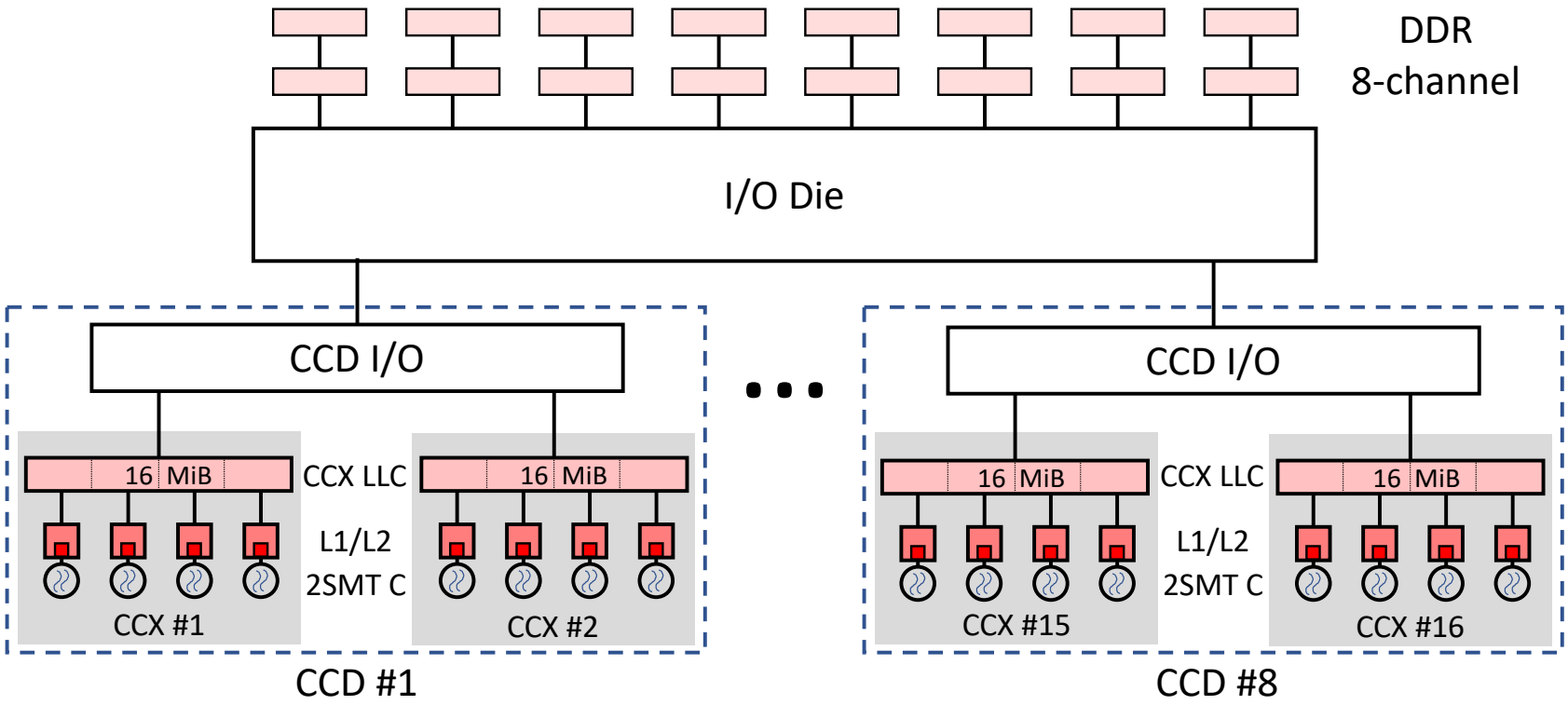
- Fujitsu Arm A64X
- IBM Power



AMD EPYC Rome

AMD PQoSE:

- Management of SLLC & BW
- Like Intel RDT



Performance vs. Memory Bandwidth

- How do SLLC miss latency increases with DRAM bandwidth?
- Triads to generate synthetic DRAM traffic

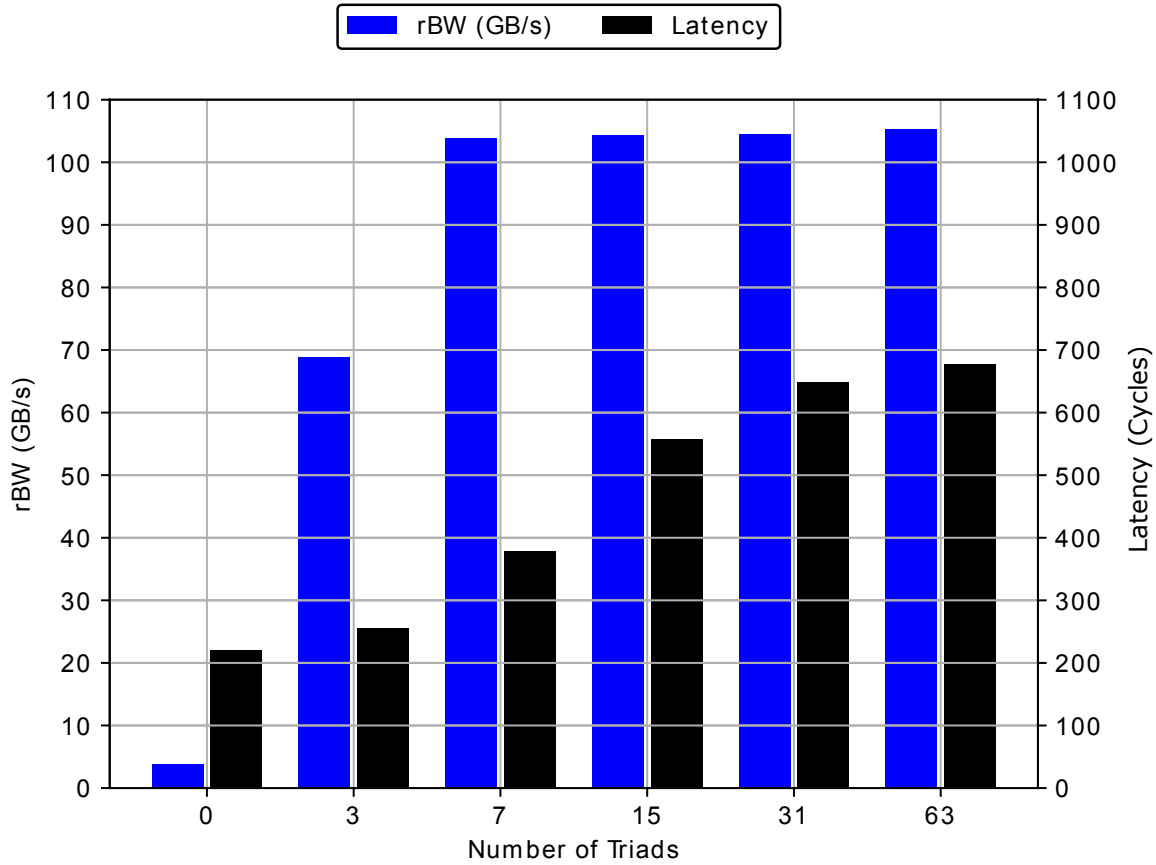
Triad:

```
for (i =0; i<N; i++)
{
    a[i] = b[i] + c[i] * SCALAR;
}
```

https://www.cs.virginia.edu/stream/ref.html

Performance vs. Memory Bandwidth

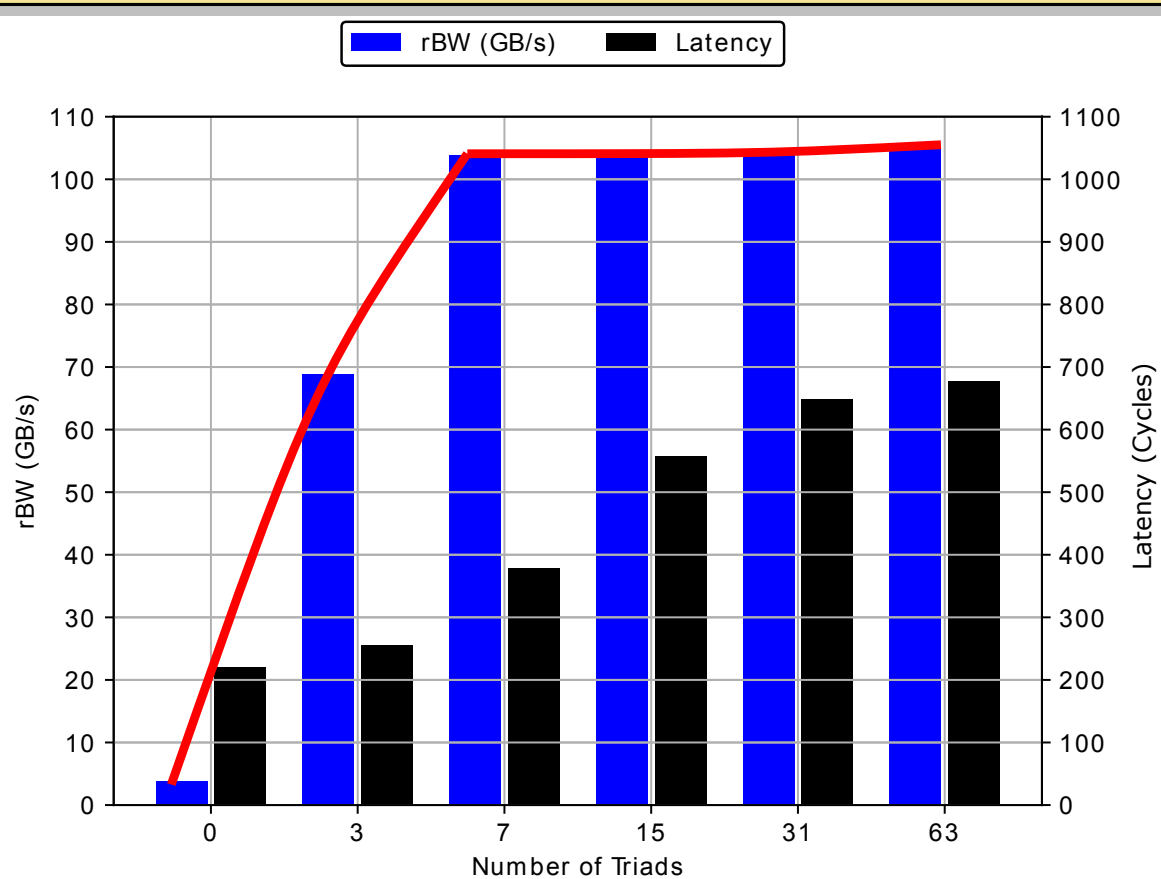
Memory bandwidth and memory access latency



Performance vs. Memory Bandwidth

Memory bandwidth and memory access latency

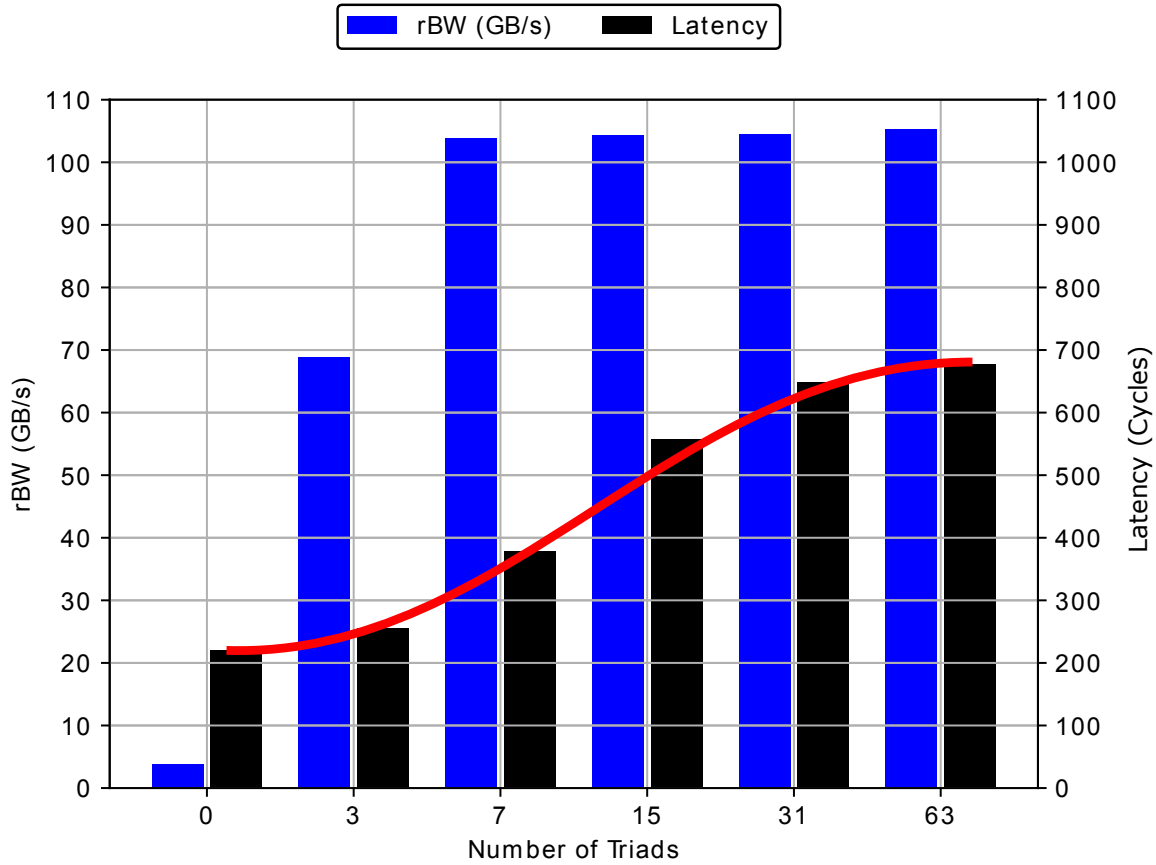
- Increase of DRAM BW up to 7 Triads



Performance vs. Memory Bandwidth

Memory bandwidth and memory access latency

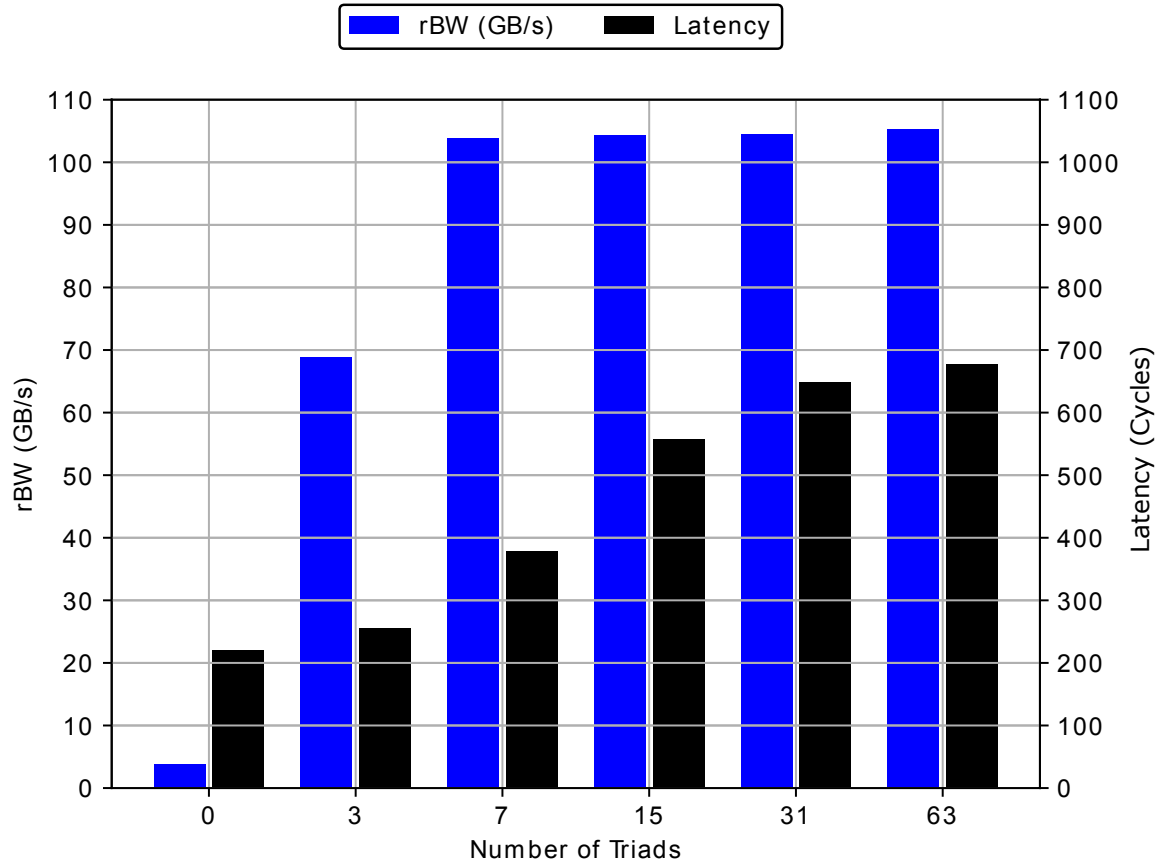
- Increase of DRAM BW up to 7 Triads
- Increase of memory access latency up to 63 Triads



Performance vs. Memory Bandwidth

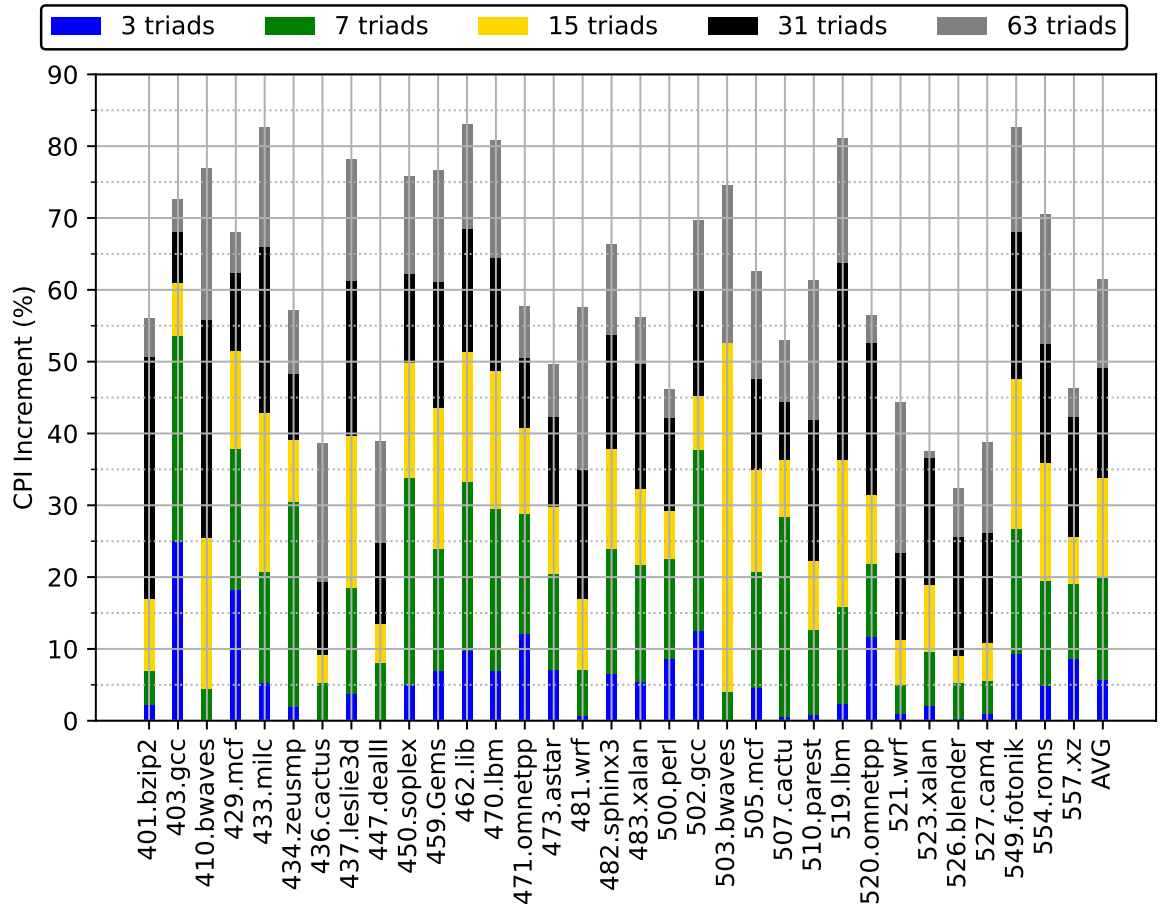
Memory bandwidth and memory access latency

Memory latency is a better indicator than bandwidth for memory contention



Performance vs. Memory Bandwidth

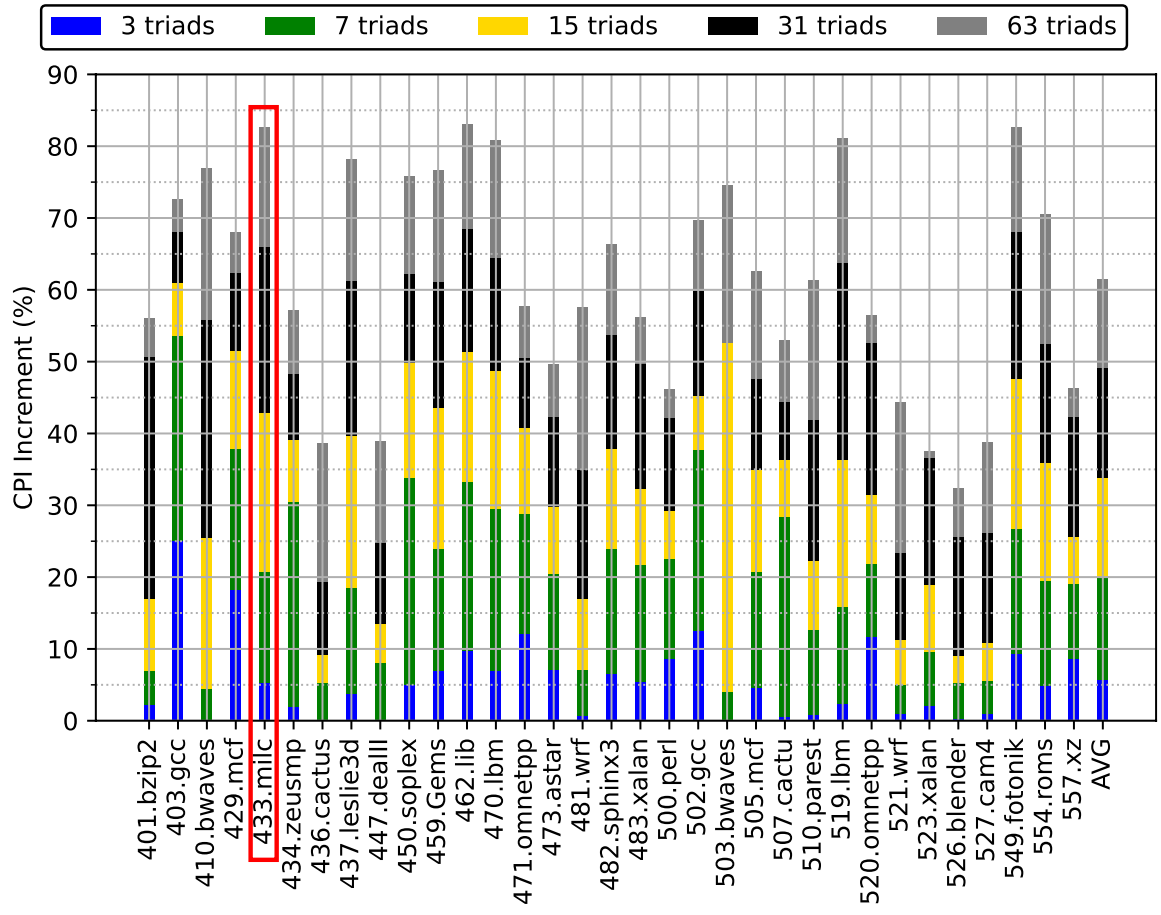
Memory bandwidth and memory access latency



Performance vs. Memory Bandwidth

Memory bandwidth and memory access latency

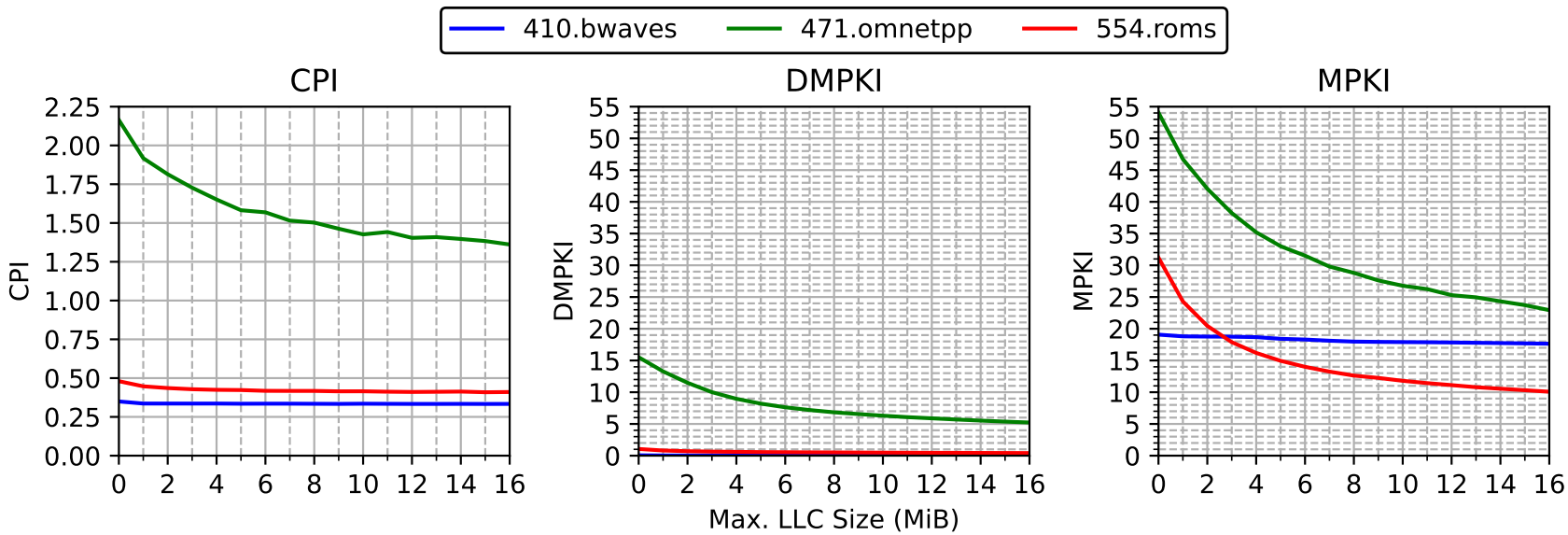
CPI increases up to 83% with memory contention



Performance vs. SLLC Capacity

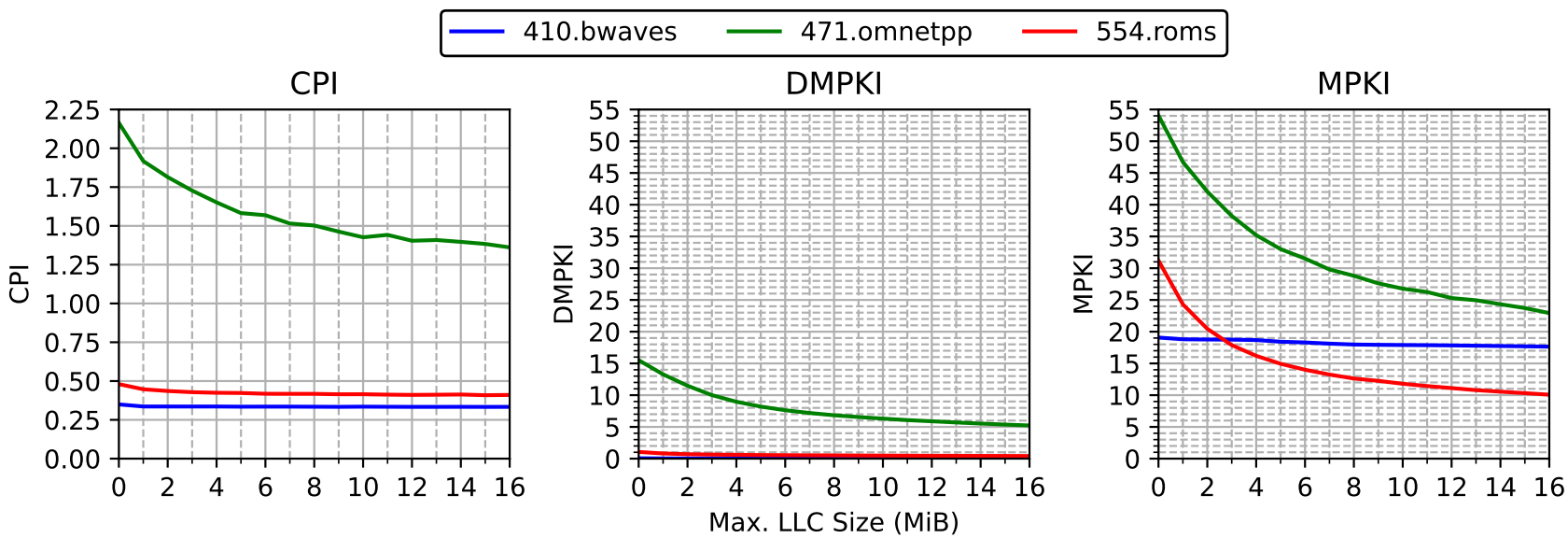
- How do applications behave with different SLLC sizes?
- SLLC: from 0 to 16 MiB
- All hardware prefetchers enabled

Performance vs. SLLC Capacity



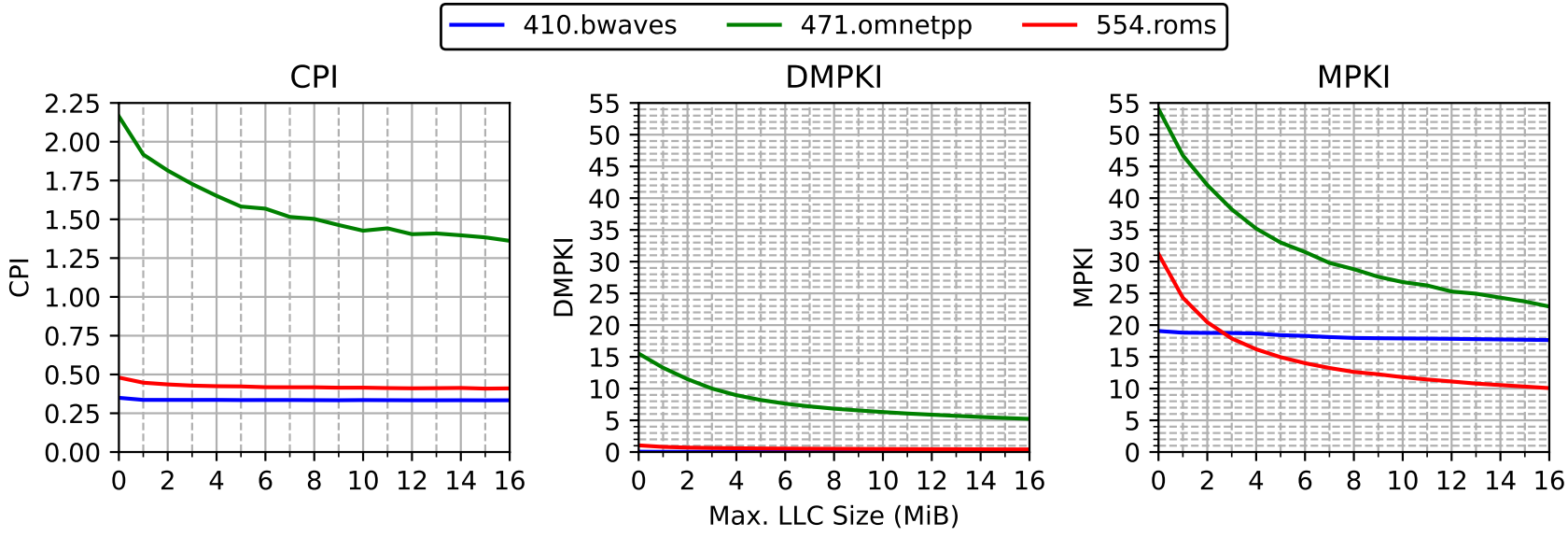
Performance vs. SLLC Capacity

1. Sensitive to SLLC size



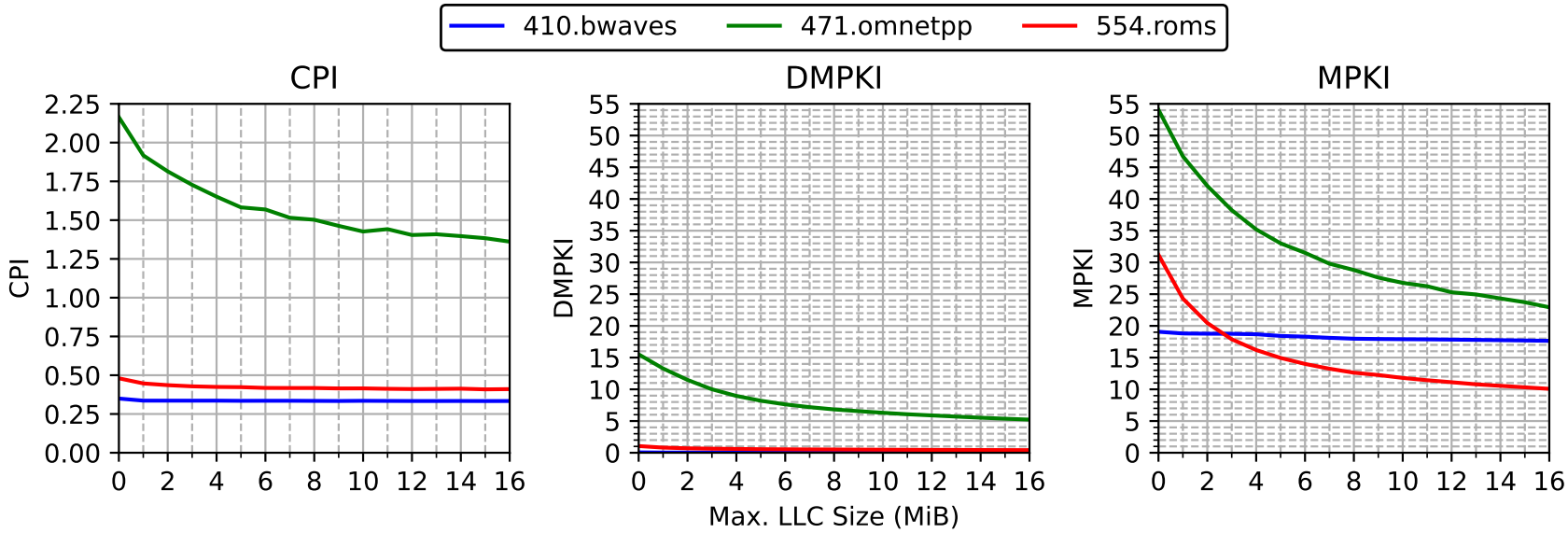
Performance vs. SLLC Capacity

1. Sensitive to SLLC size
2. Insensitive to SLLC size



Performance vs. SLLC Capacity

1. Sensitive to SLLC size
2. Insensitive to SLLC size
3. *Pseudo*-Insensitive to SLLC size



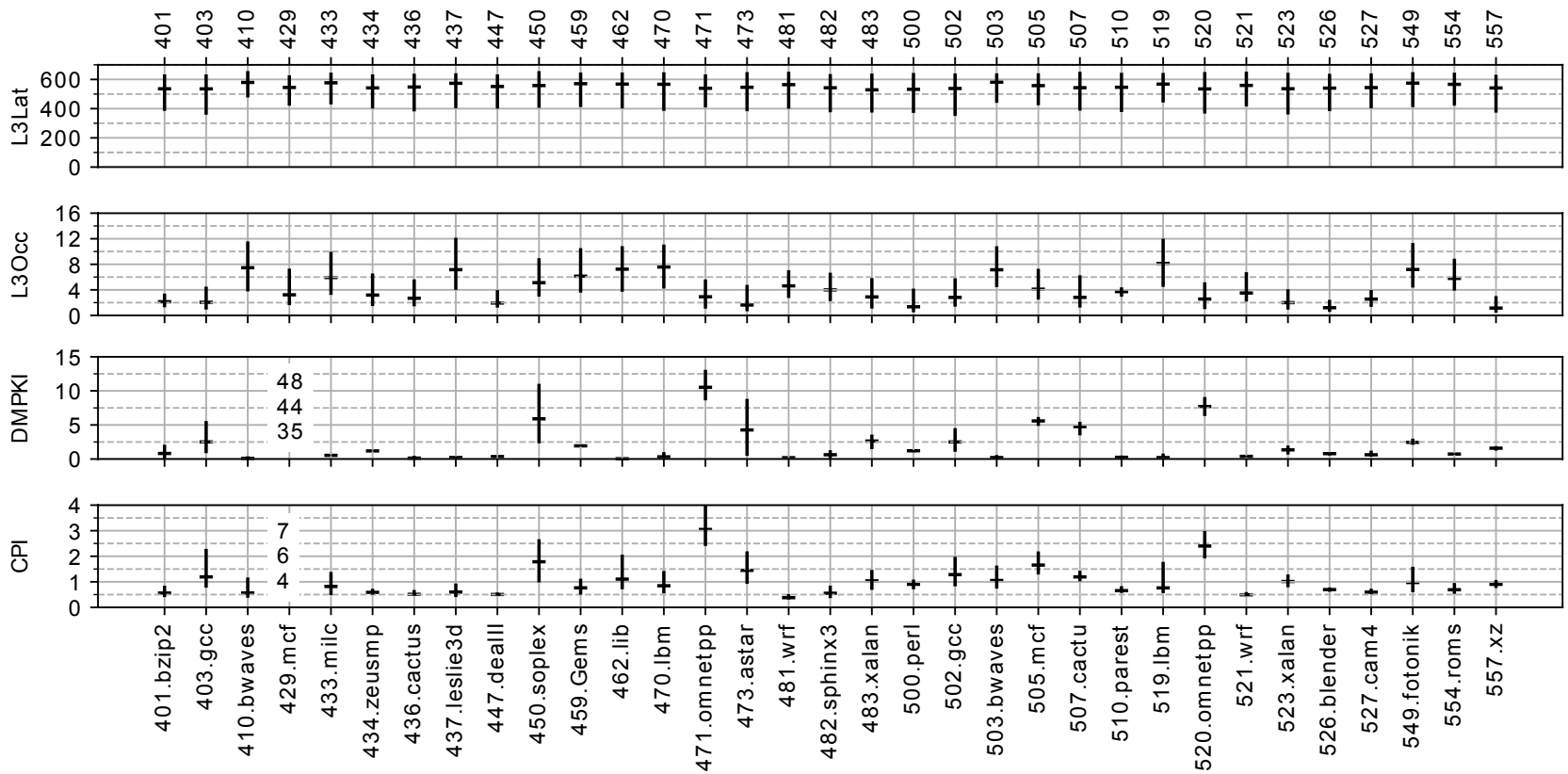
Multiprogrammed Workload

- How do applications interfere with each other?
- How much SLLC does each application use?
- Where are the memory bottlenecks?

Multiprogrammed Workload

Evaluation

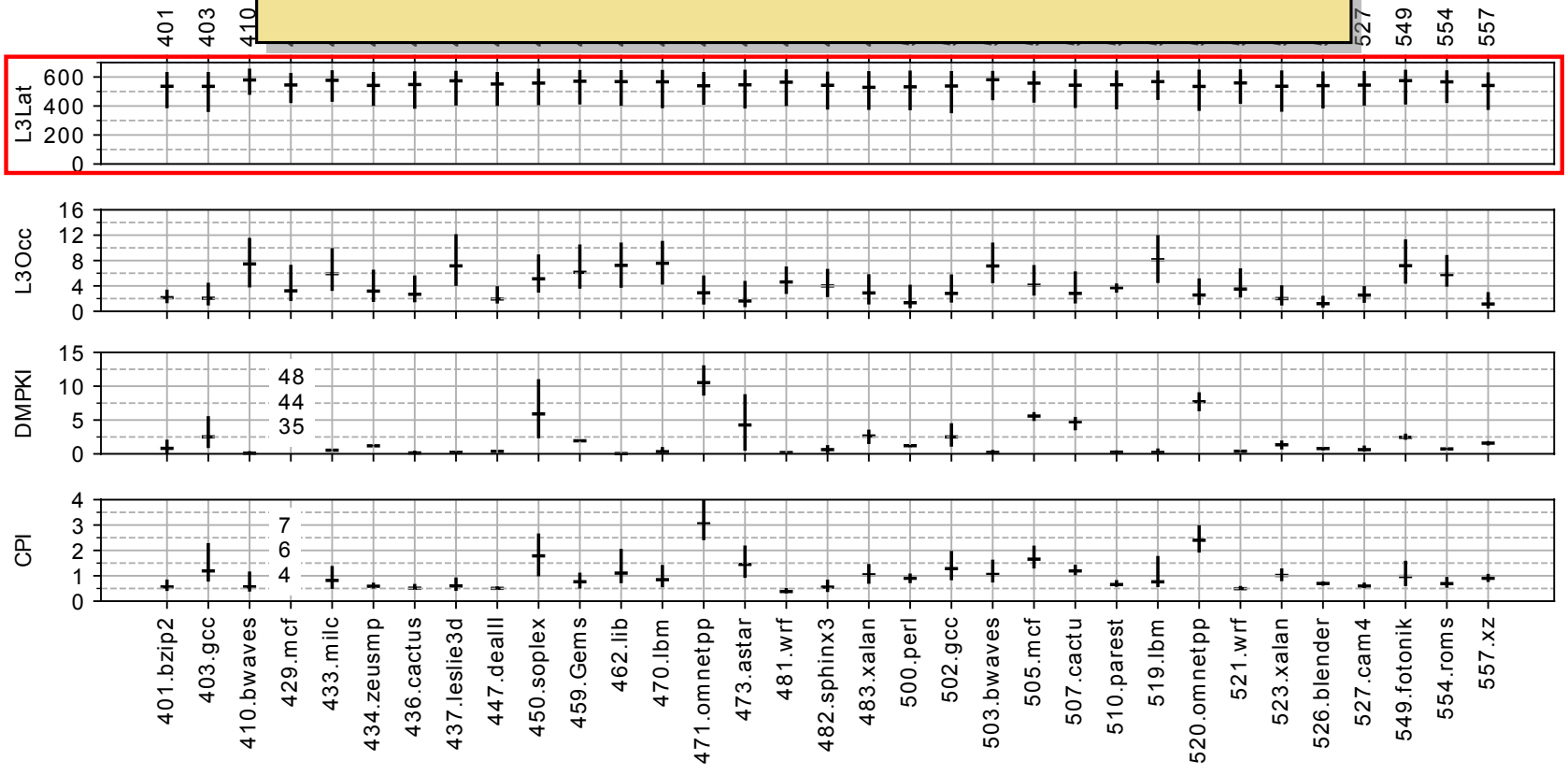
Run 100 mixes of 64 applications



Multiprogrammed Workload

Evaluation

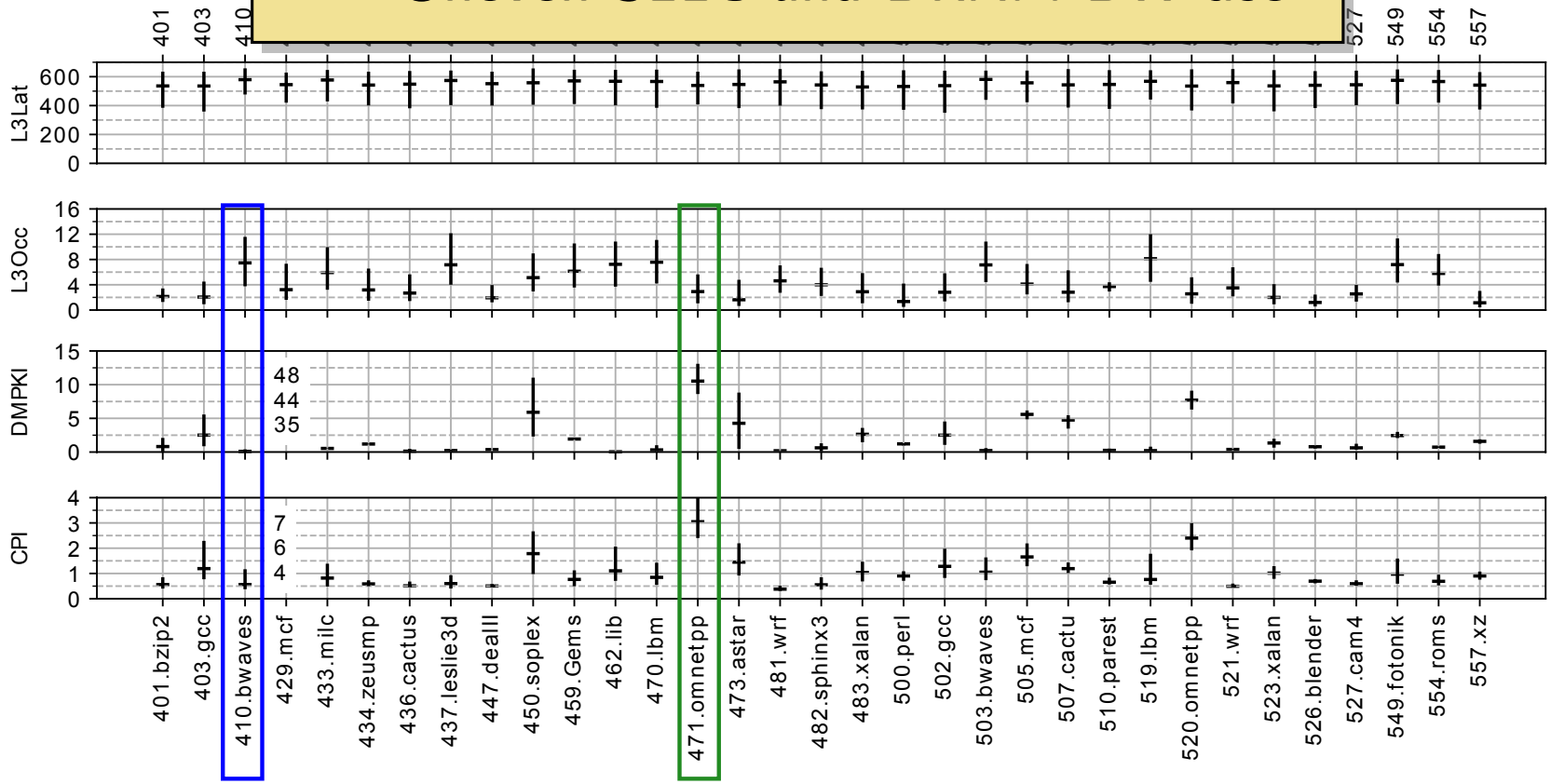
- DRAM congestion
- High latencies = high execution time



Multiprogrammed Workload

Evaluation

- DRAM congestion
- High latencies = high execution time
- **Uneven SLLC and DRAM BW use**



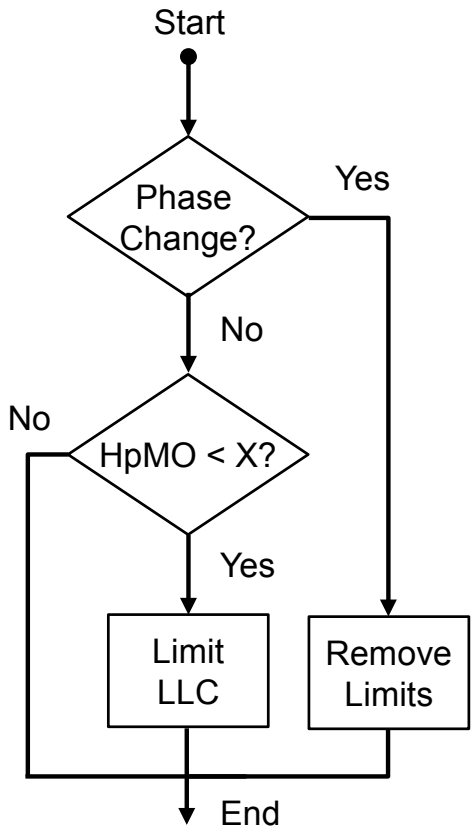
Balancer

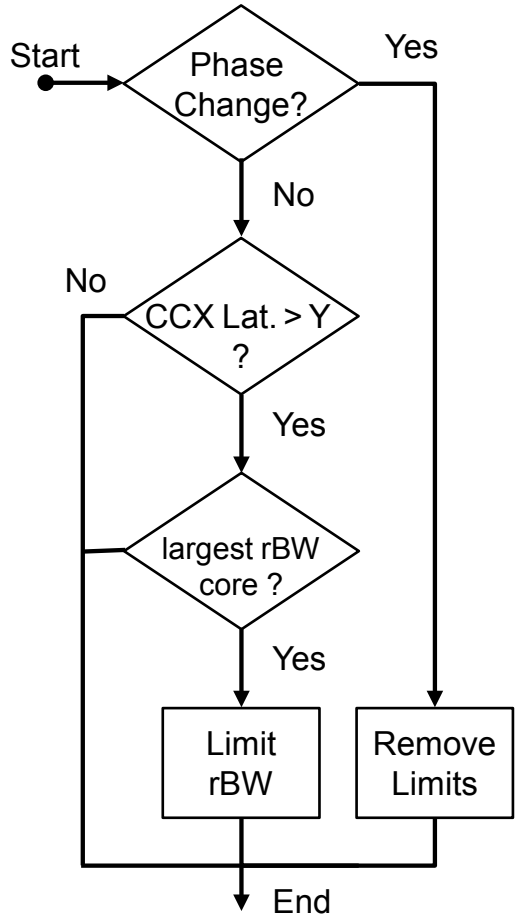
The idea

- Constrain applications which do not take advantage of shared resources
- Penalize a few applications to improve system performance
- It can be tuned to improve performance and/or fairness
- Designed for clustered SLLC systems

Balancer

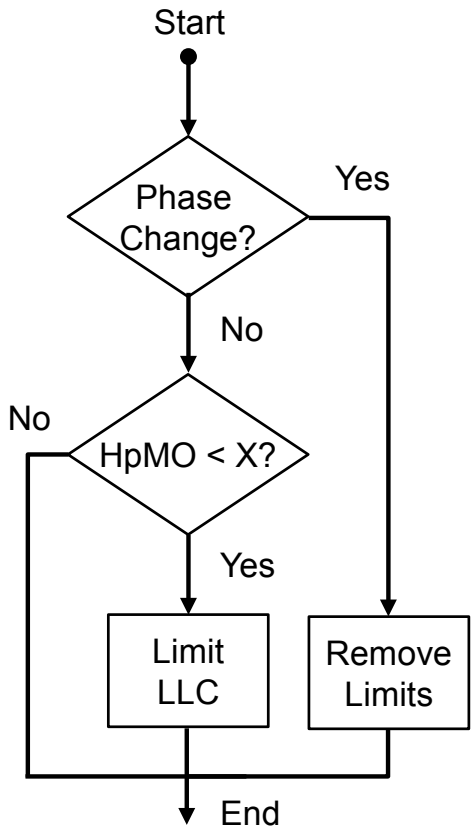
- Control of SLLC Occupancy + Control of Memory Traffic
- Wake up every 1s →Read hardware counters →Balancer

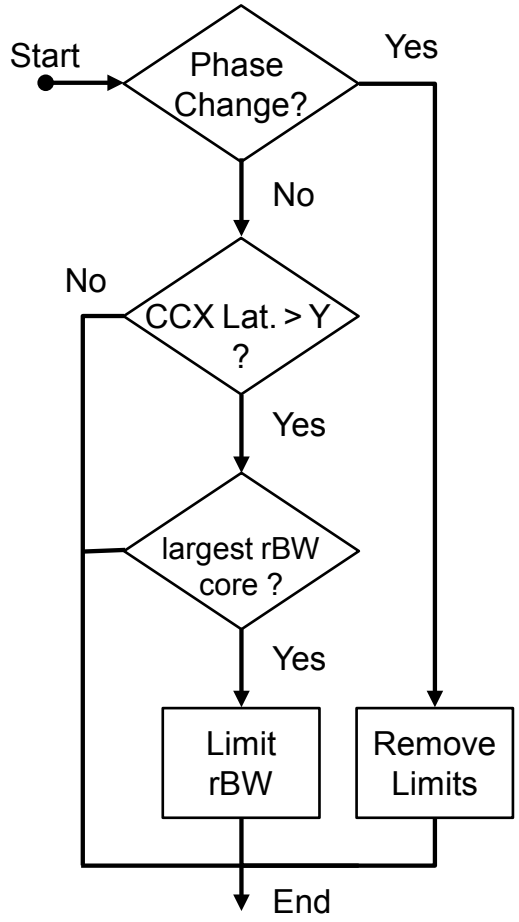




Balancer

- Control of SLLC Occupancy + Control of Memory Traffic
- Wake up every 1s →Read hardware counters →Balancer
- Detect phase change: memory accesses varies $> 20\%$

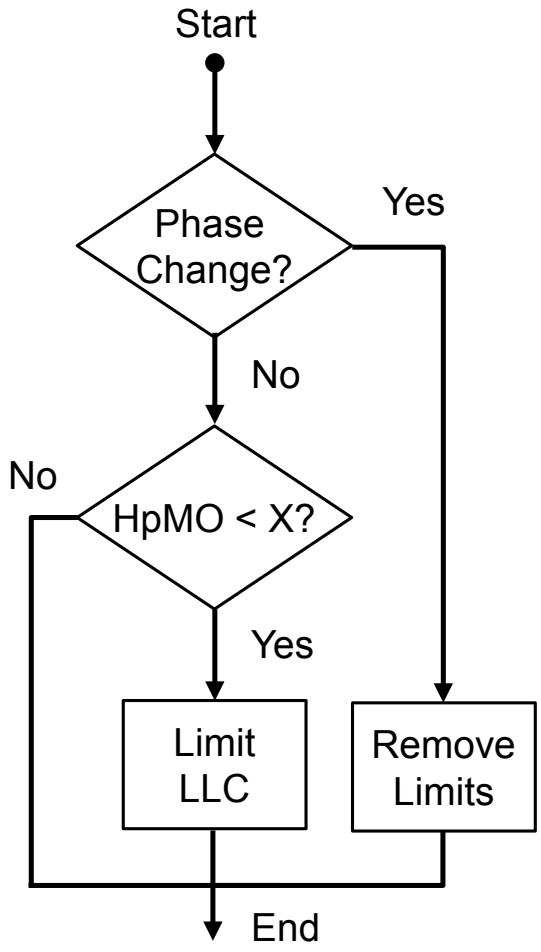




Balancer

Control of SLLC Occupancy

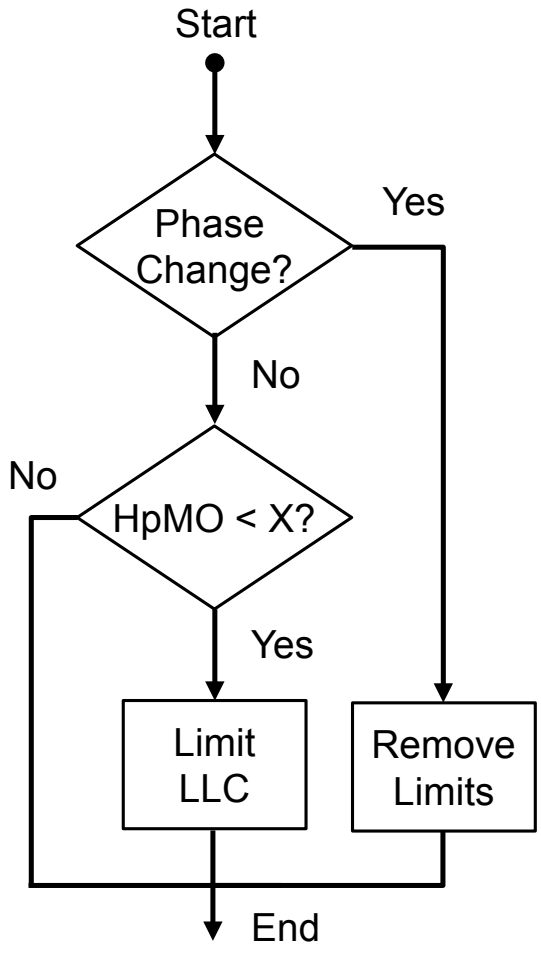
For each core in each CCX



Balancer

Control of SLLC Occupancy

For each core in each CCX

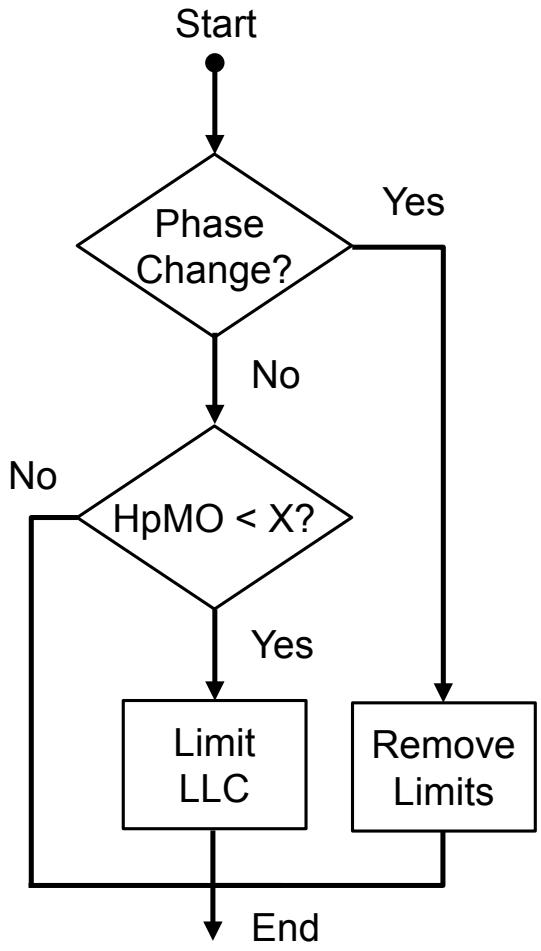


Low utilization of SLLC

Balancer

Control of SLLC Occupancy

For each core in each CCX

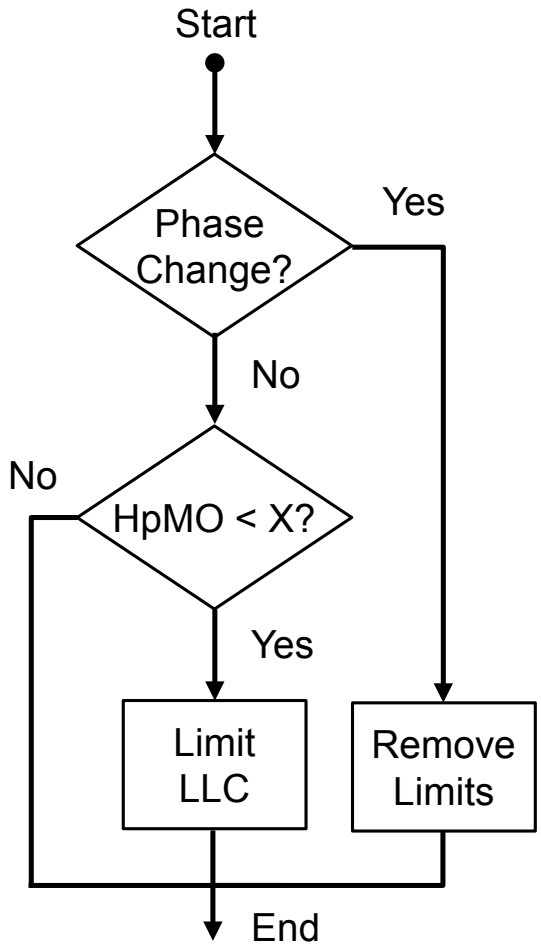


$$HpMO = \frac{Hits_{LLC}}{Misses_{LLC} \cdot Occupancy_{LLC}}$$

Balancer

Control of SLLC Occupancy

For each core in each CCX

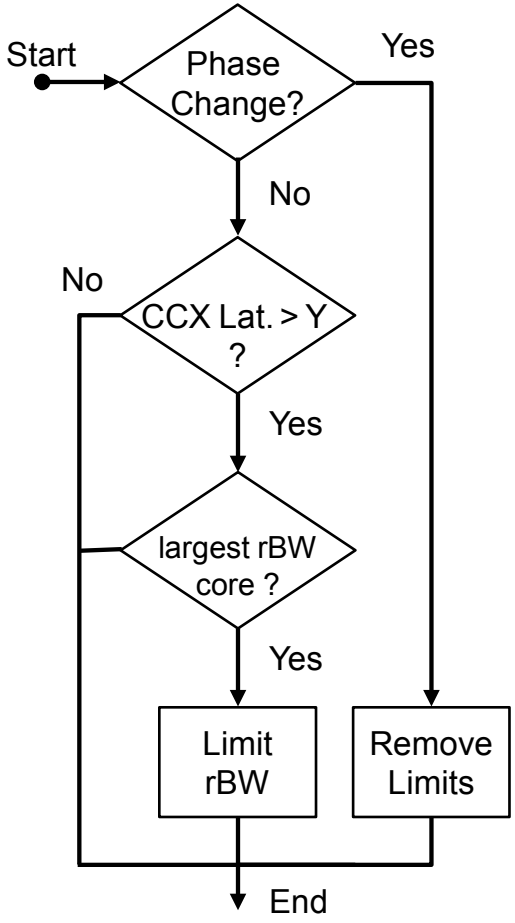


Constrain core to 1 MiB of SLLC

Balancer

Control of Memory Traffic Occupancy

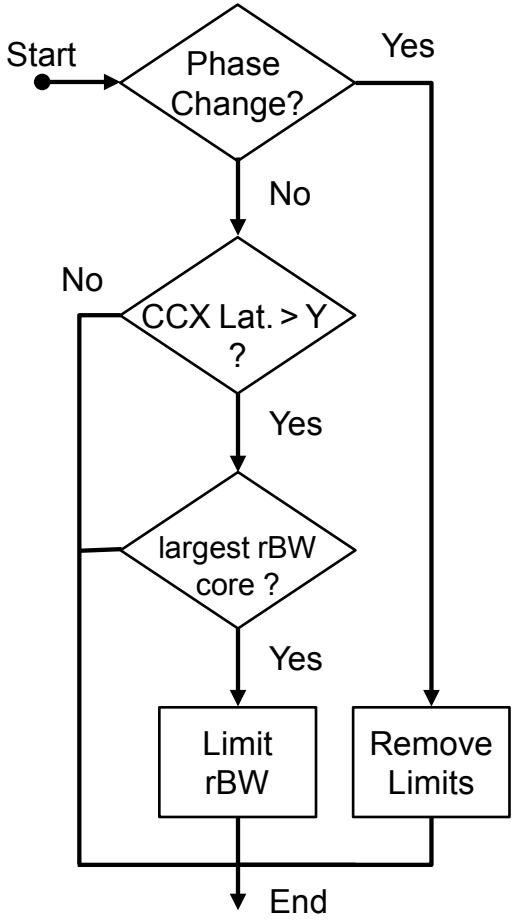
For each CCX in each CCD



Balancer

Control of Memory Traffic Occupancy

For each CCX in each CCD

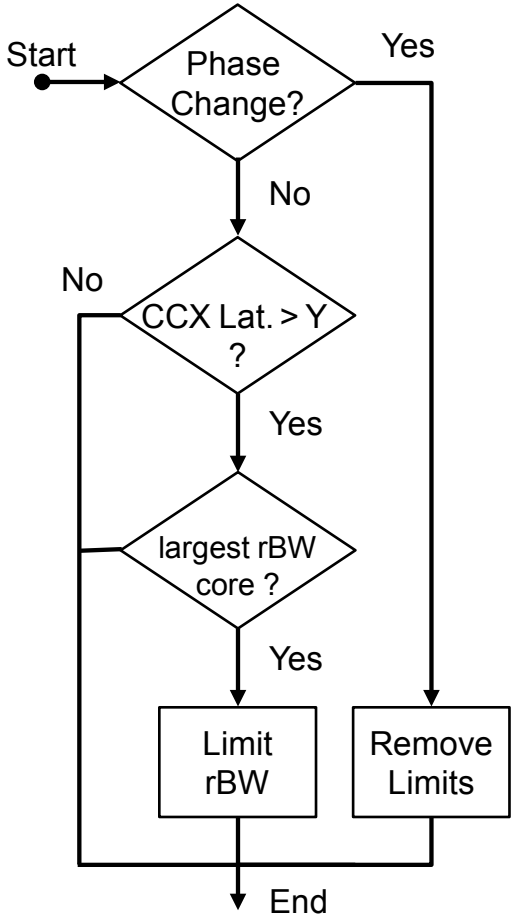


High memory latency?

Balancer

Control of Memory Traffic Occupancy

For each CCX in each CCD

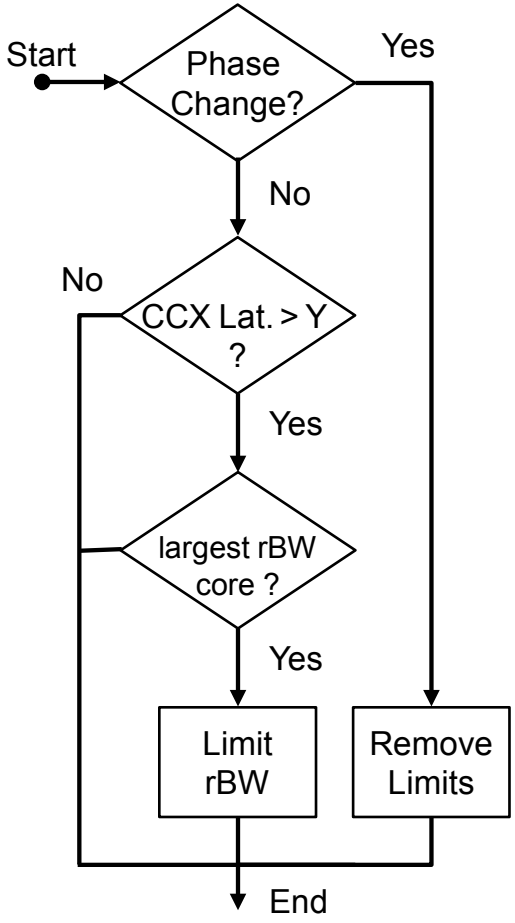


Pick core that use more rBW

Balancer

Control of Memory Traffic Occupancy

For each CCX in each CCD



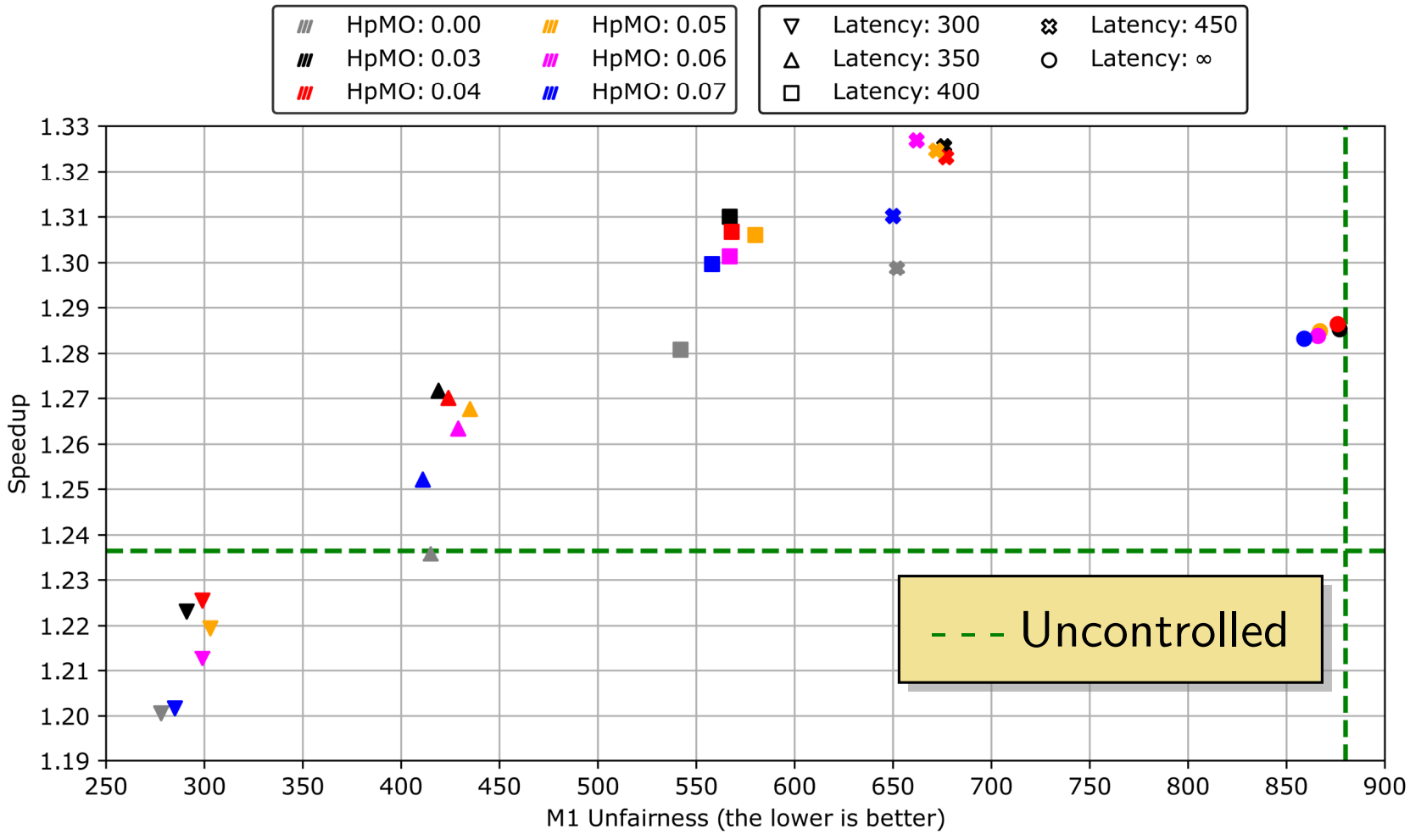
Constrain core max. rBW

Evaluation

- Different HpMO and latency thresholds
- How does Balancer improve performance and fairness?
- Balancer vs. Static vs. Uncontrolled

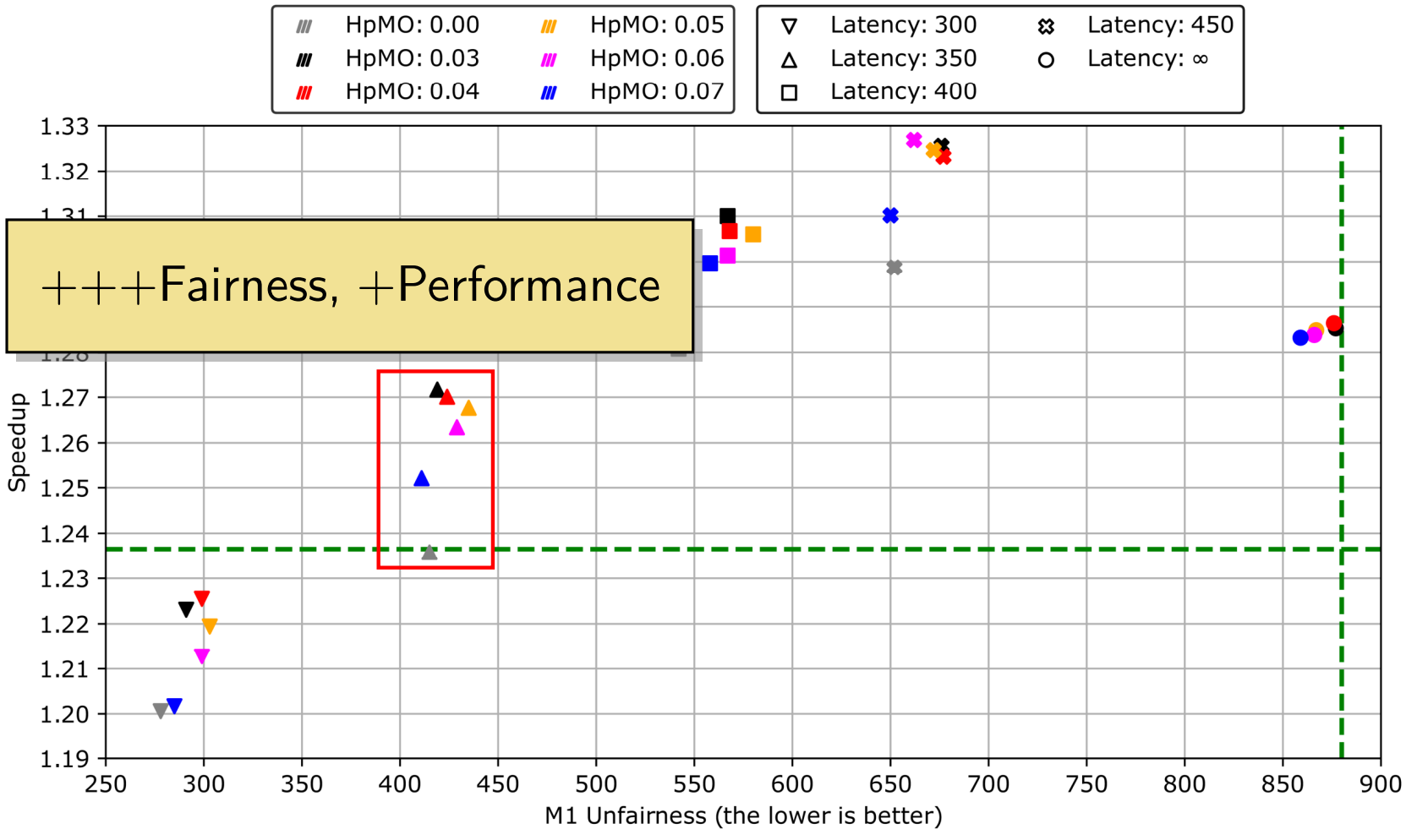
Evaluation

Performance vs. Fairness



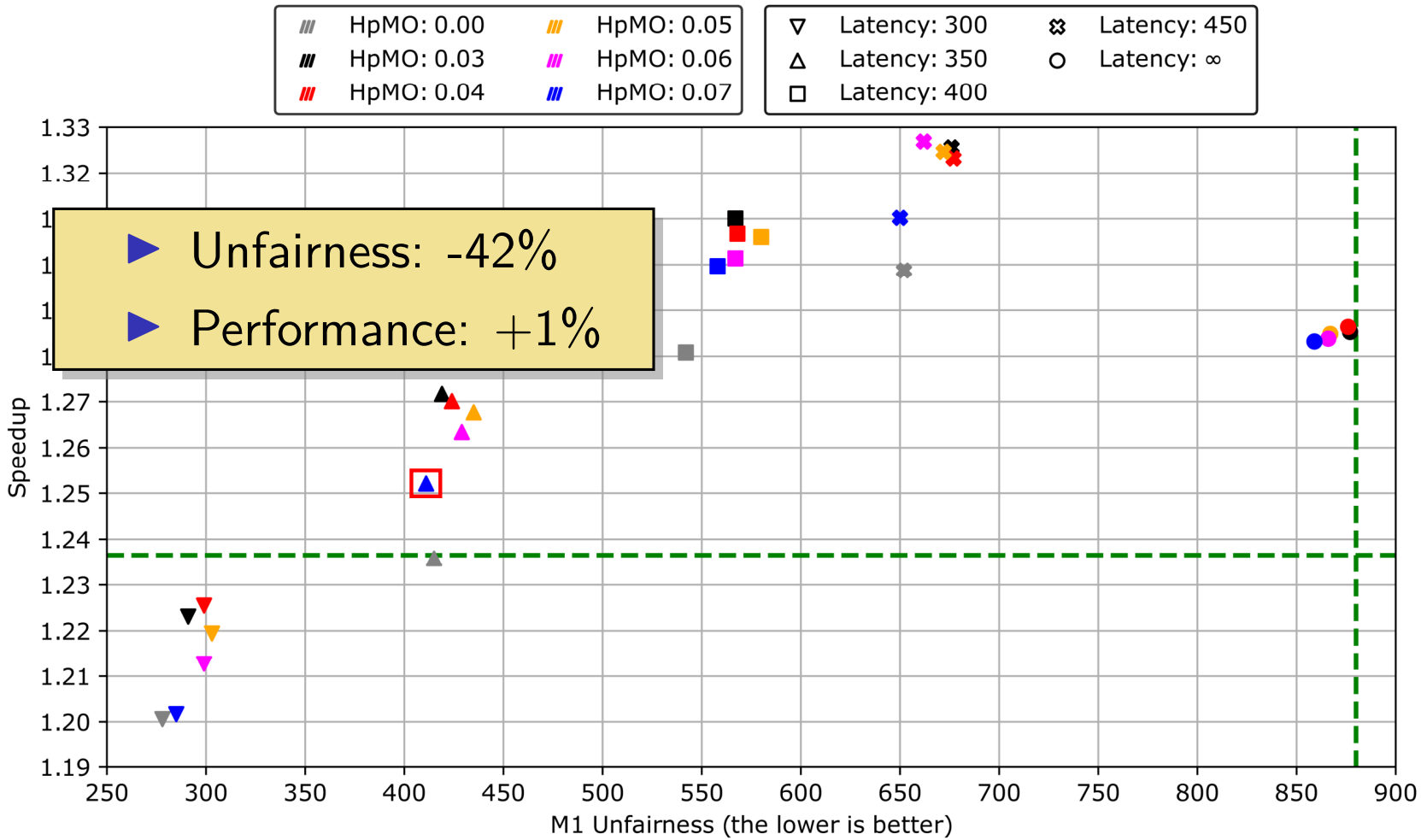
Evaluation

Performance vs. Fairness



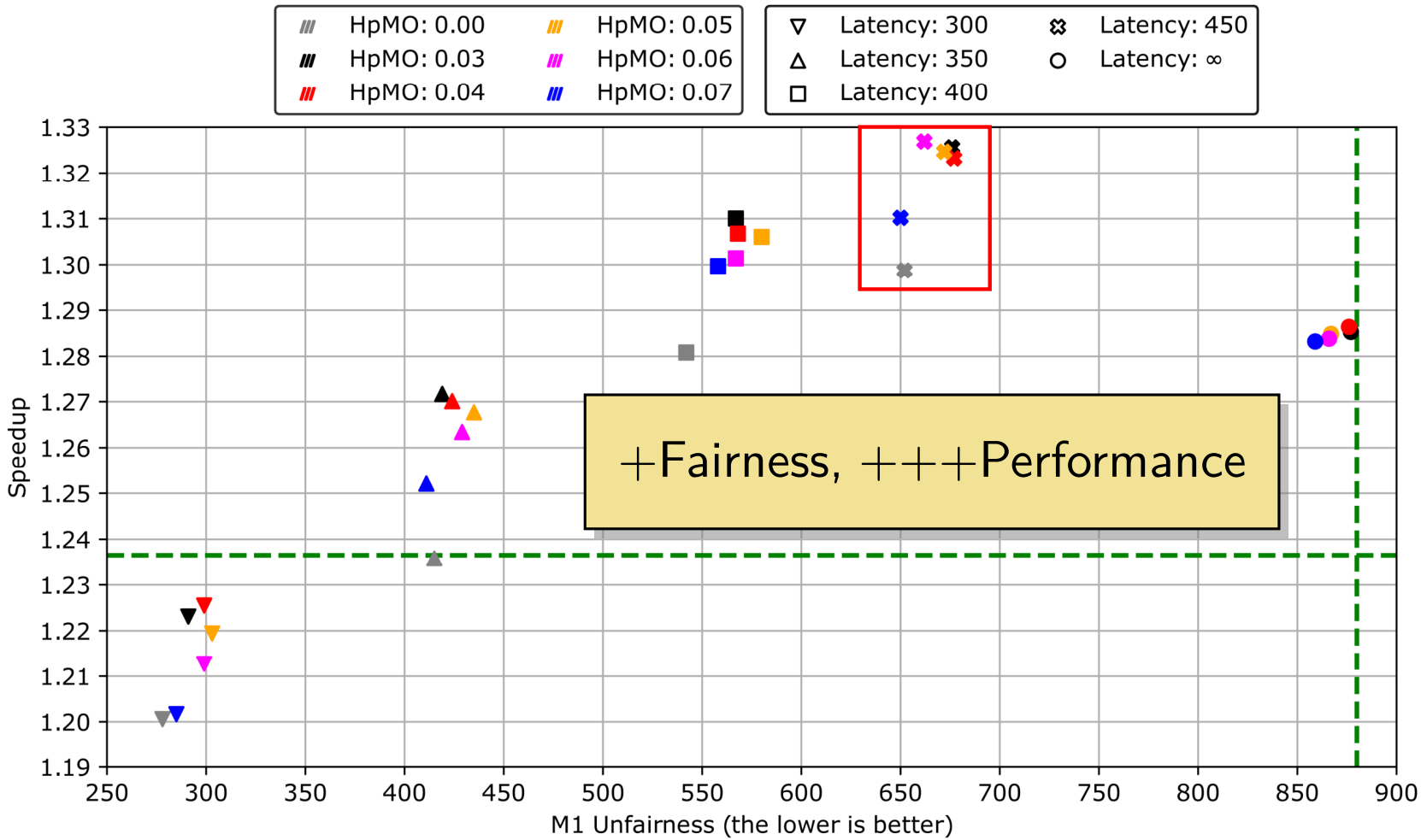
Evaluation

Performance vs. Fairness



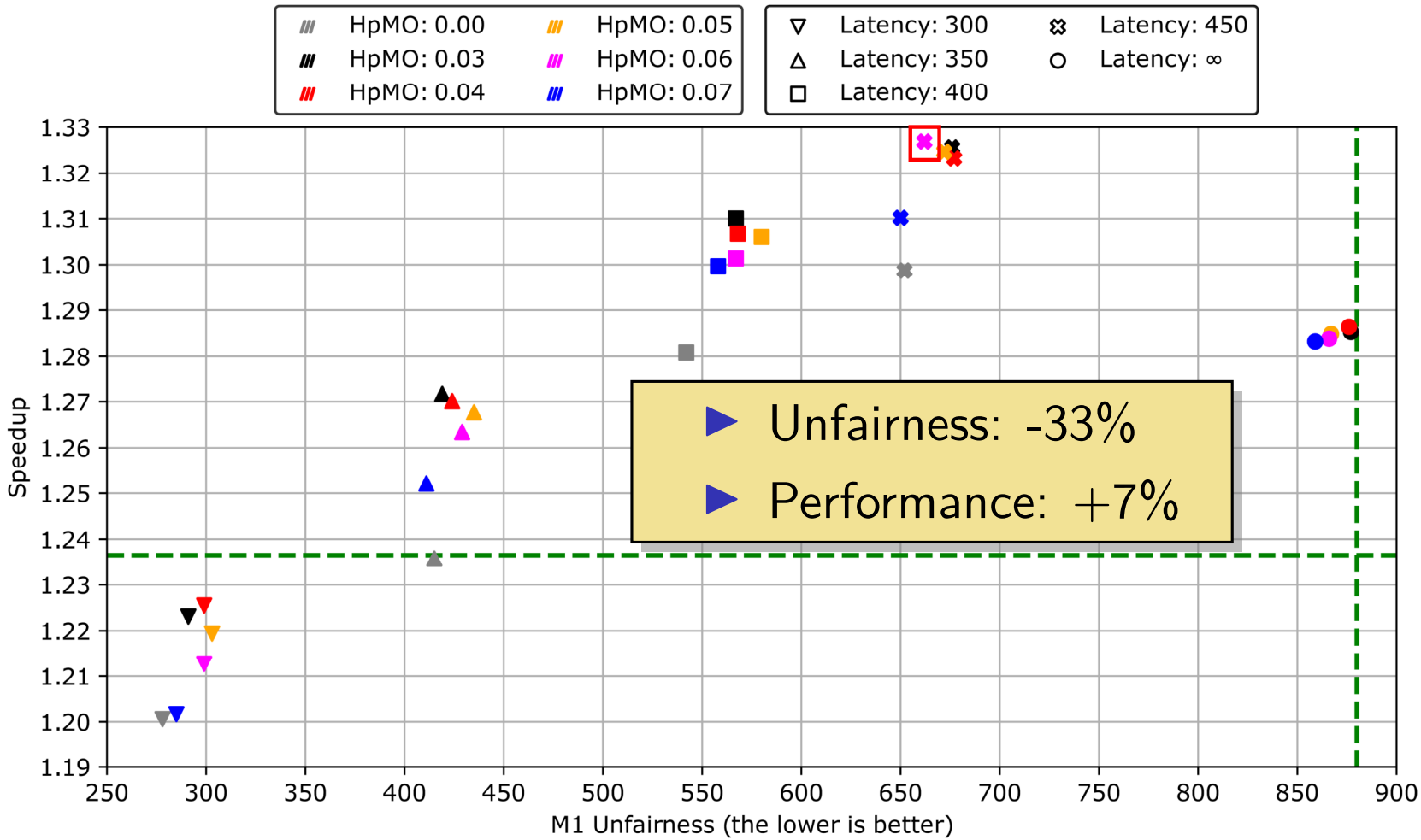
Evaluation

Performance vs. Fairness



Evaluation

Performance vs. Fairness



Conclusion remarks

- We identified three types of SLLC behaviour
- Memory latency is a better indicator than bandwidth for memory contention
- New mechanism to limit SLLC and memory bandwidth
- Balancer can improve fairness and/or performance

Introduction

- Current high-end processors can execute hundreds of threads
- Applications should be parallelized to boost performance
- Increasing complexity of synchronization
- Several synchronization mechanisms are available for programmers
  - **Classical Mechanism: Locks & Lock-Free**
  - **Transactional Memory: Software & Hardware**

Work done during an internship at Huawei Zürich Research Center

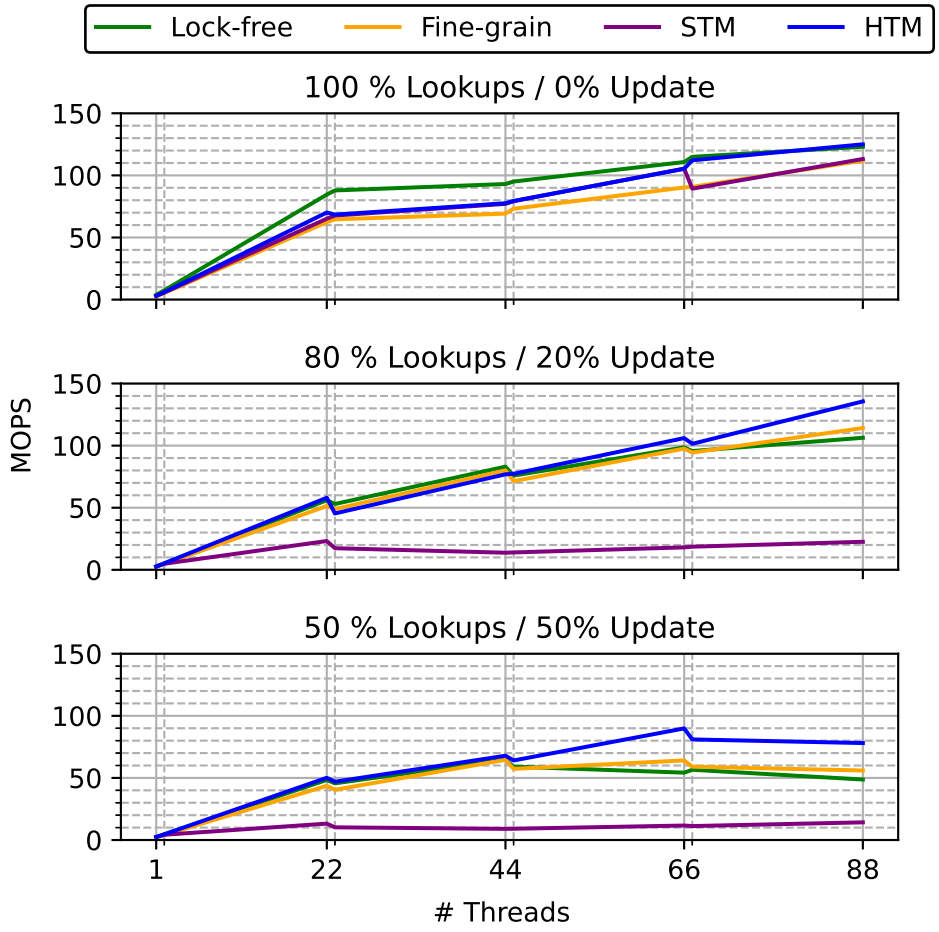
Data structures

Methodology

- We evaluate two widely used data structures: **Hash Table (HT)** & **Binary Search Tree (BST)**
- Three workloads:
  - 100% lookup operations
  - 80% lookup and 20% update operations
  - 50% lookup and 50% update operations
- 2xIntel Xeon Gold (up to 88 threads)

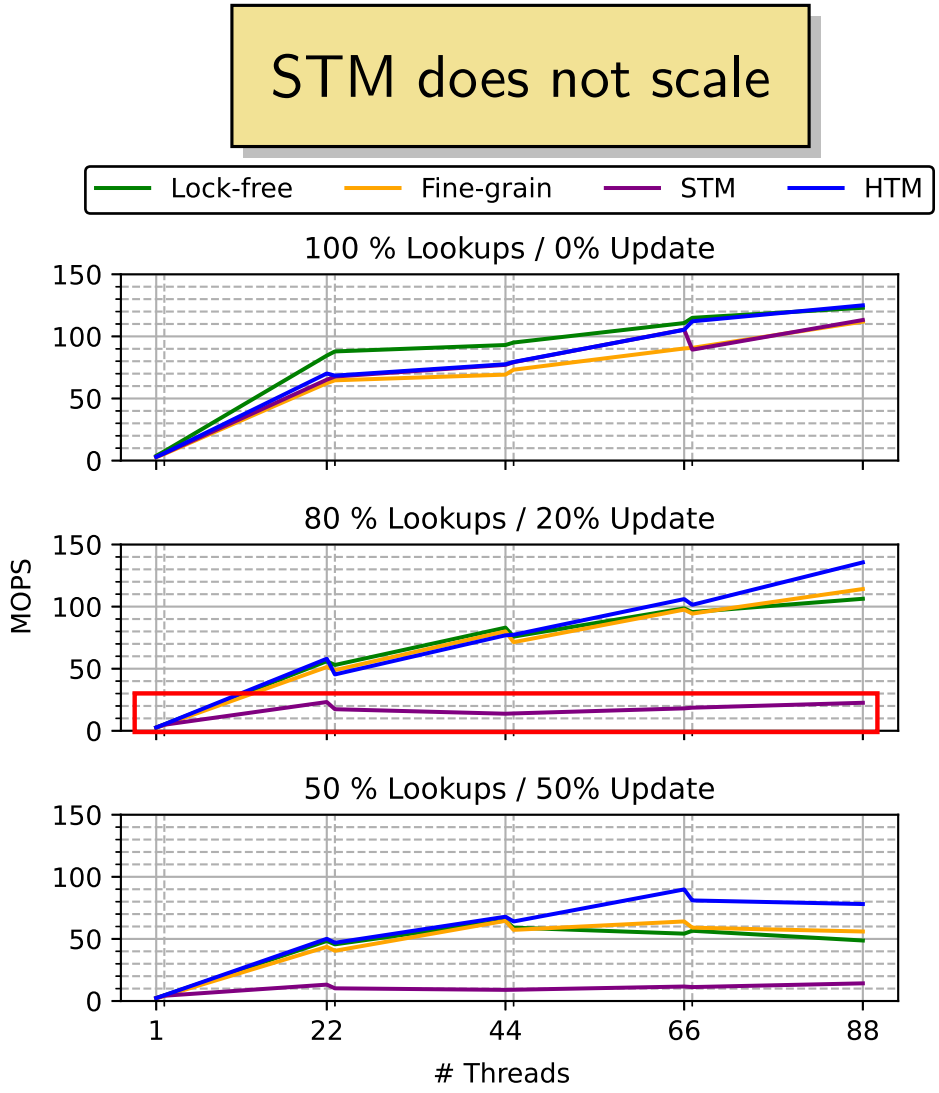
Hash Table

Throughput



Hash Table

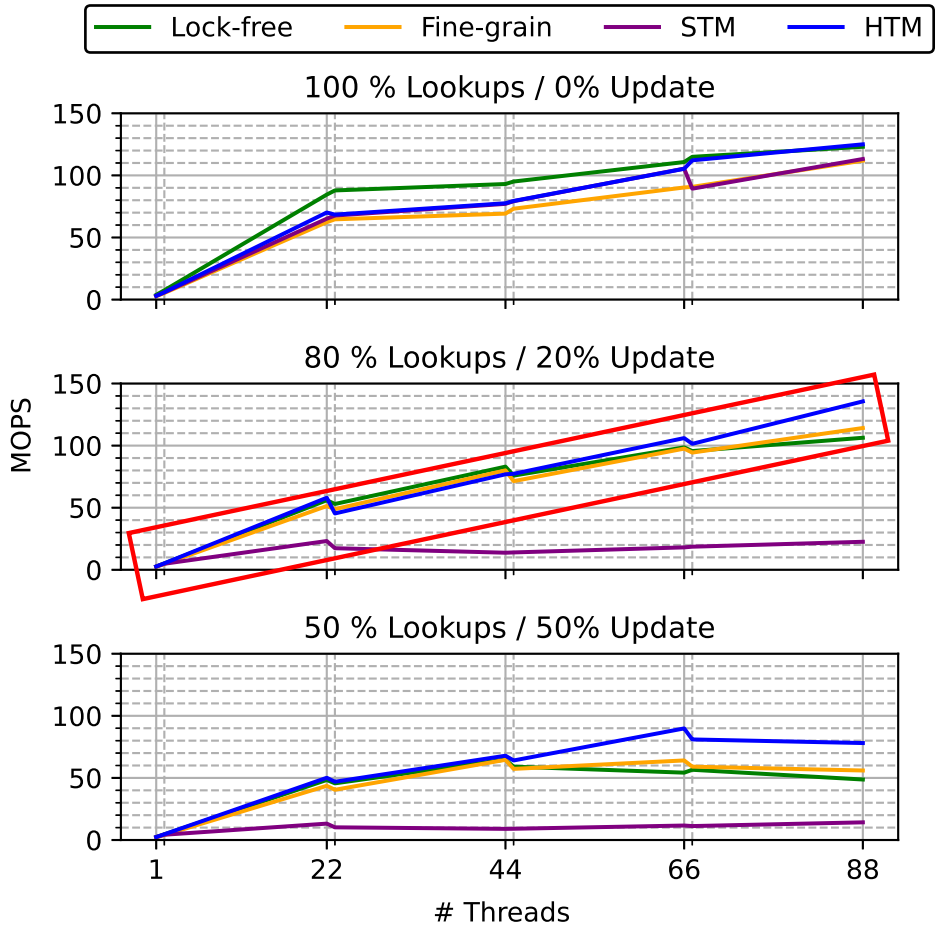
Throughput



Hash Table

Throughput

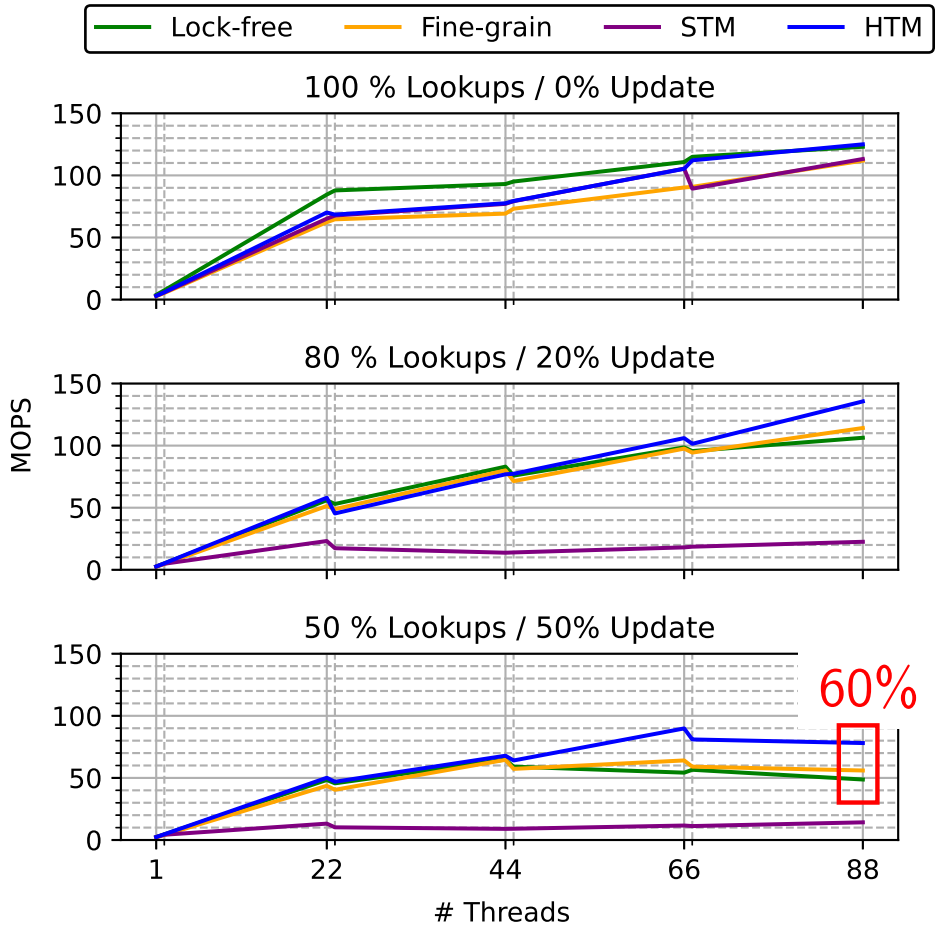
HTM, Locks & Lock-Free show comparable performance



Hash Table

Throughput

HTM performs better with larger # of thread

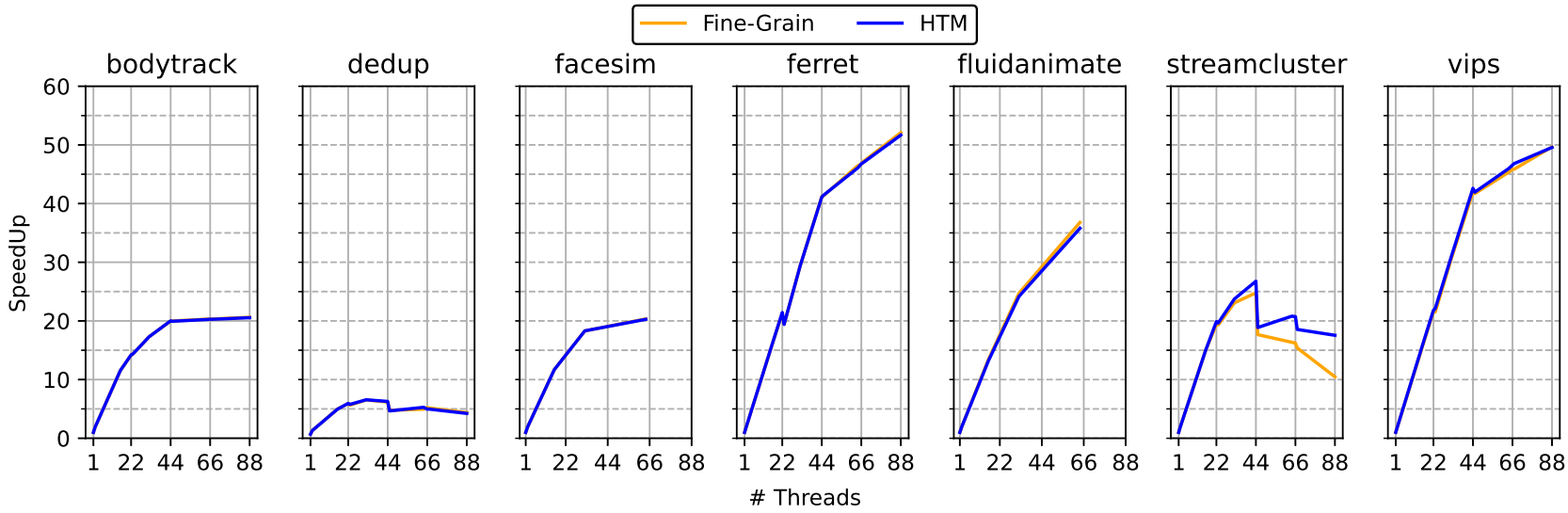


HTM ease-of-use

- How real workloads behave under HTM rather than lock
- PARSEC 3.0 most used parallel benchmark suite
- Only benchmarks that have a significant number of locks
- We use lock elision through GLIBC Linux library

HTM ease-of-use

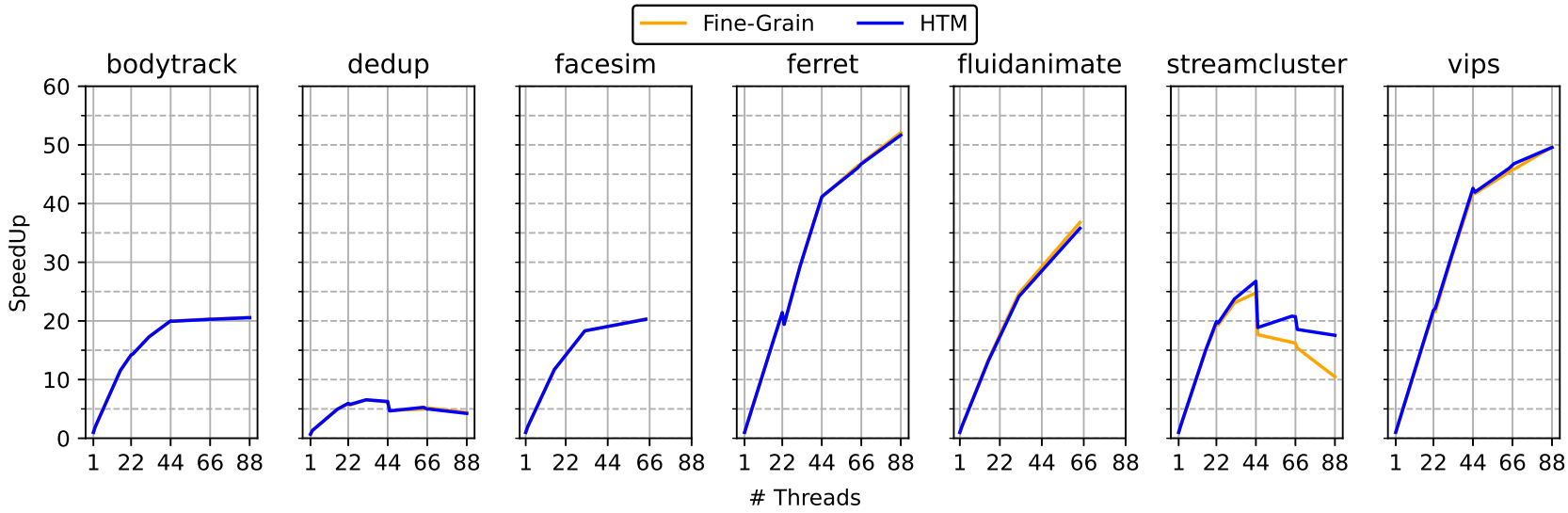
Performance



HTM ease-of-use

Performance

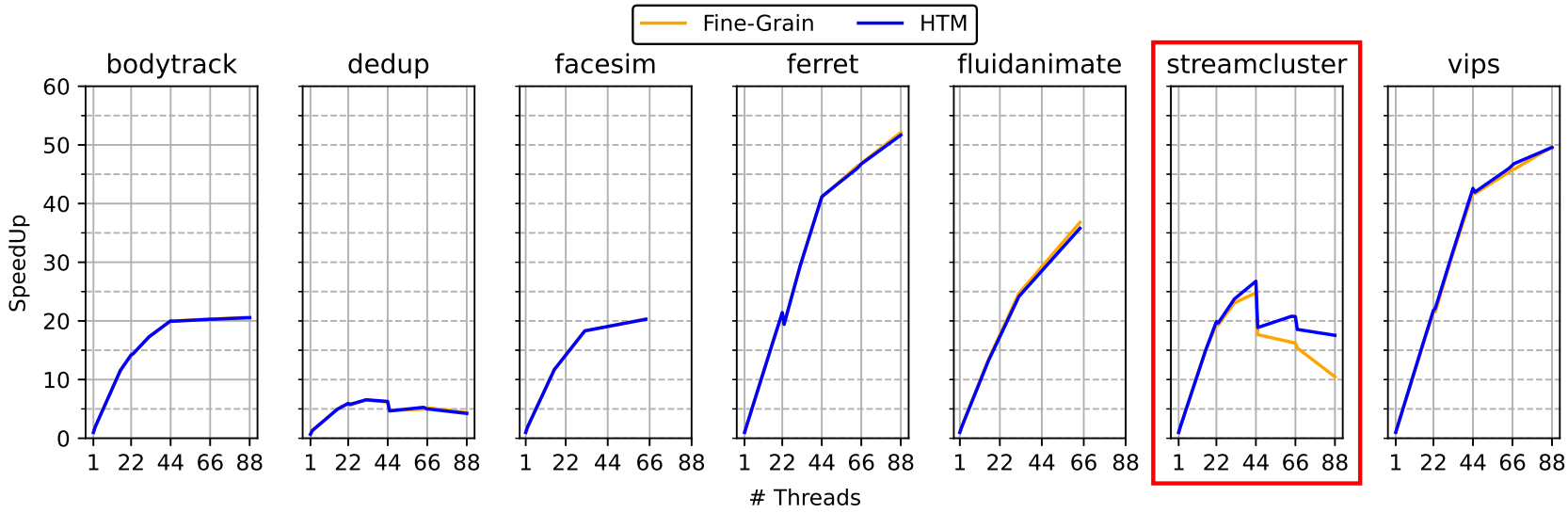
There is no performance difference in 6/7



HTM ease-of-use

Performance

In one of benchmarks HTM works better



Conclusion Remarks

- STM is lagging behind other mechanisms
- But HTM scales better than lock-free and lock-based
- HTM comparable to highly-optimized fine-grained locking

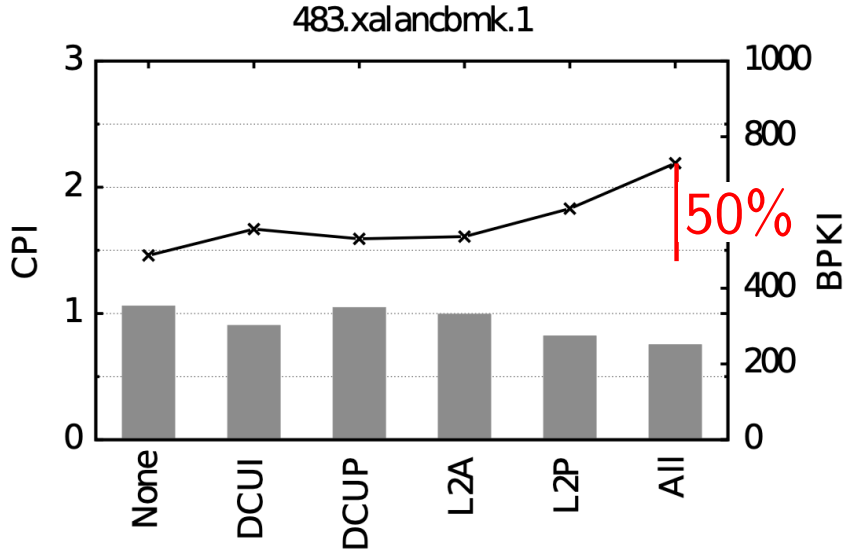
Introduction

- Hardware prefetching is critical for performance
- Accurate in most of the benchmarks

This work was done during an internship at University of Murcia

Introduction

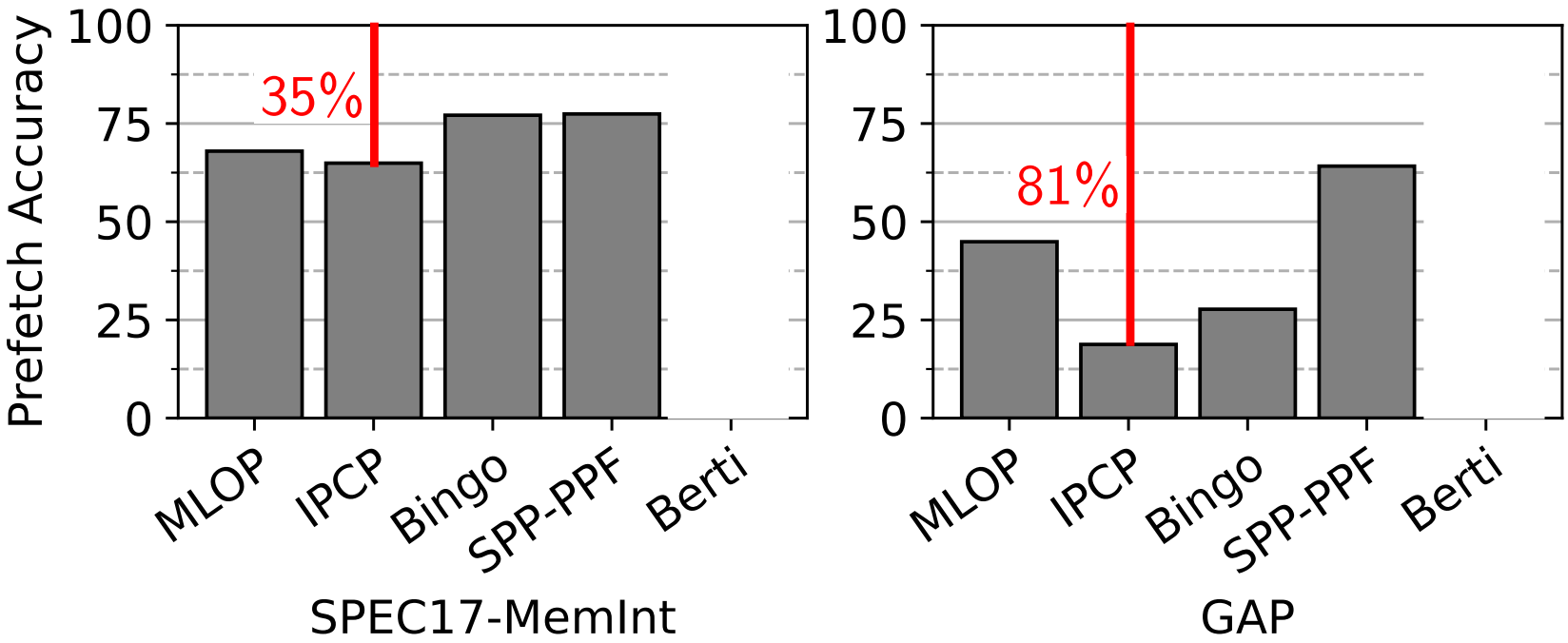
- Hardware prefetching is critical for performance
- Accurate in most of the benchmarks
- ...but not all of them



This work was done during an internship at University of Murcia

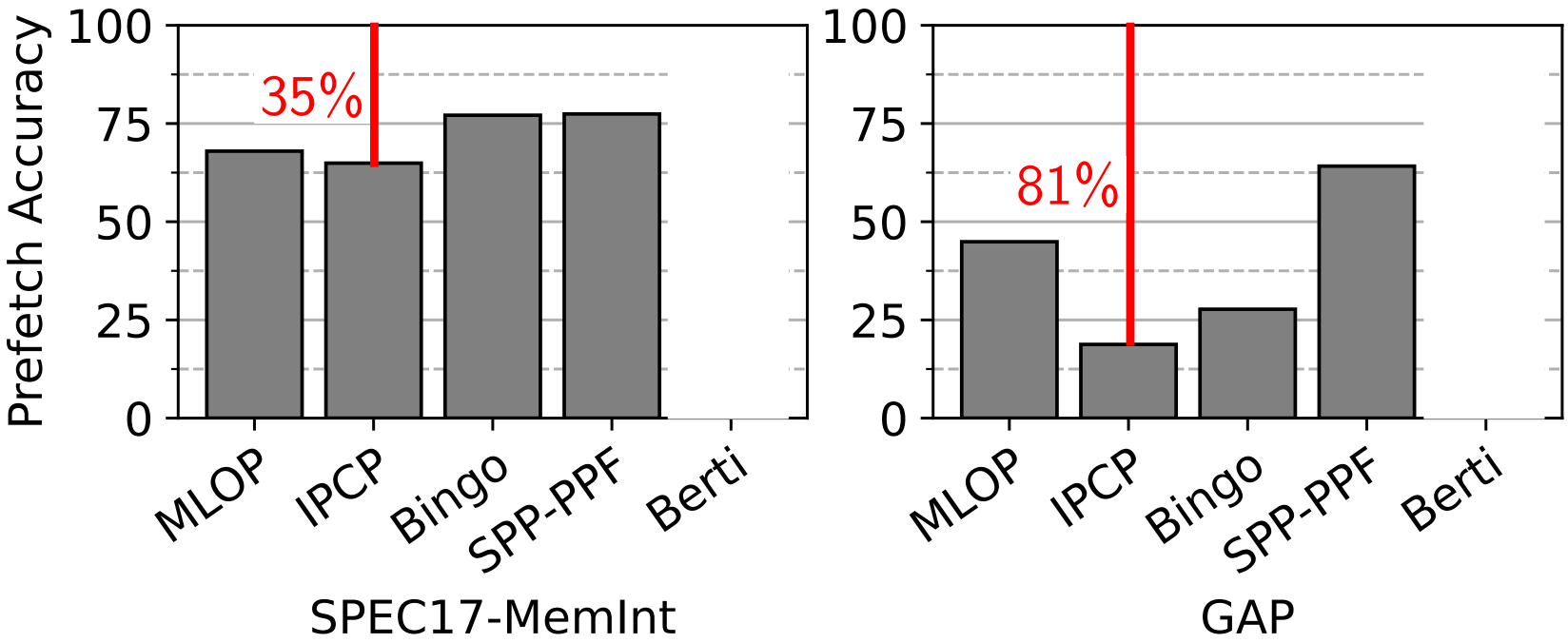
Introduction

- Most of the state-of-the-art hw prefetchers target L2/LLC
- ...but they lack accuracy



Introduction

- Most of the state-of-the-art hw prefetchers target L2/LLC
- ...but they lack accuracy
- **We propose Berti a new L1D prefetch that push the limits of single-thread performance and high accuracy**



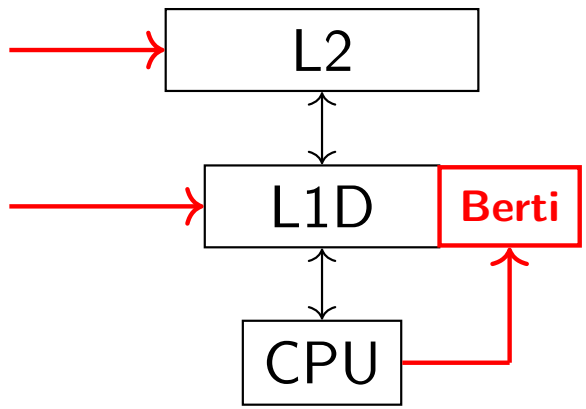
Introduction

Key aspects

- Sited at L1D
- Learns patterns by local deltas
- Strong case for accuracy

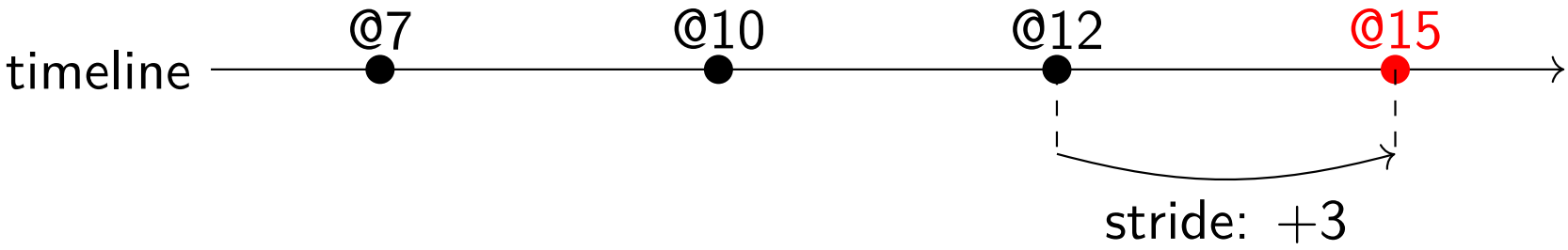
Motivation: Why a L1D Prefetcher?

- Orchestrates prefetch requests across L1D/L2
- Instruction Pointer (IP) and unfiltered memory references
- Virtual addresses: cross-page prefetching



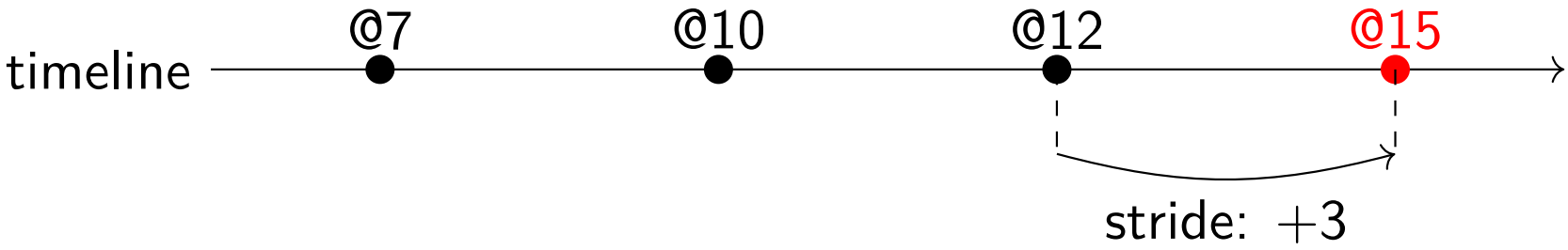
Motivation: Why a new delta prefetcher?

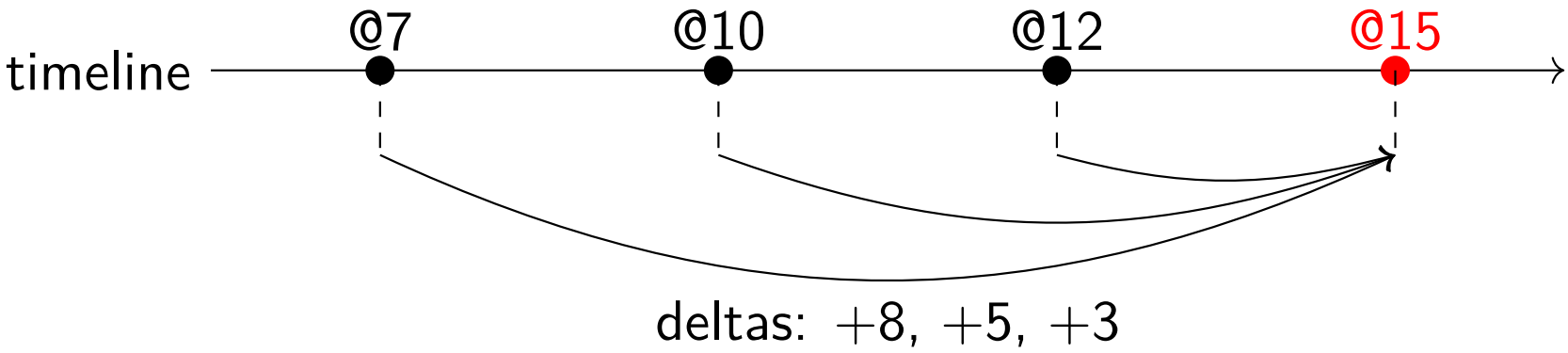
Delta vs. Stride



Motivation: Why a new delta prefetcher?

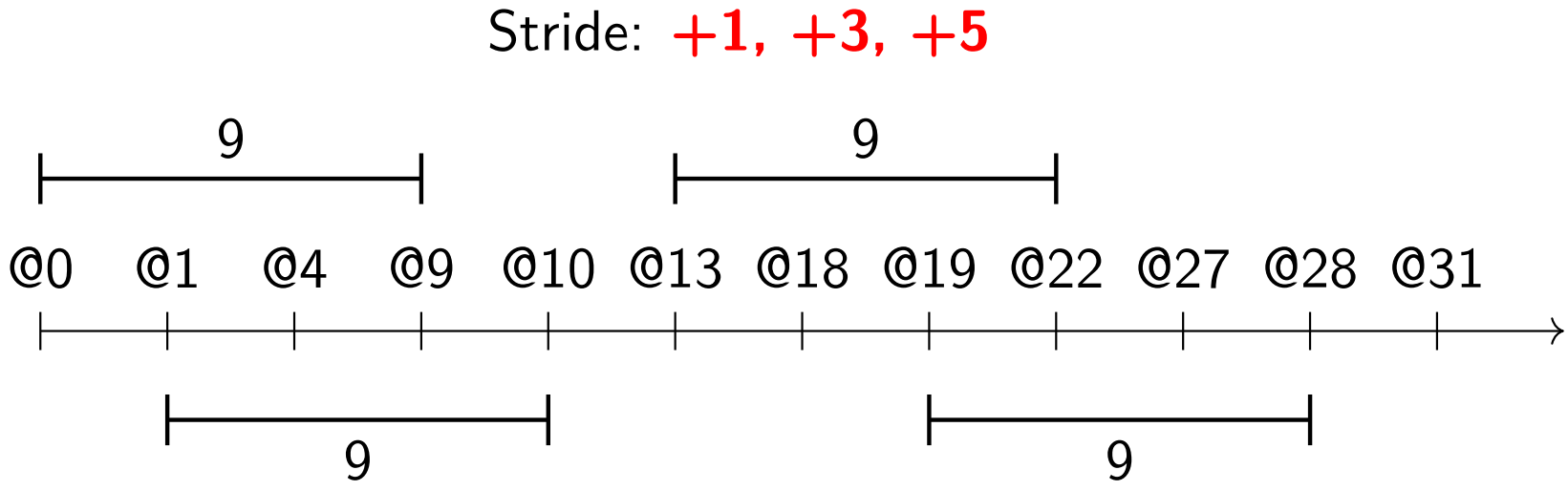
Delta vs. Stride





Motivation: Why a new delta prefetcher?

Why Deltas?

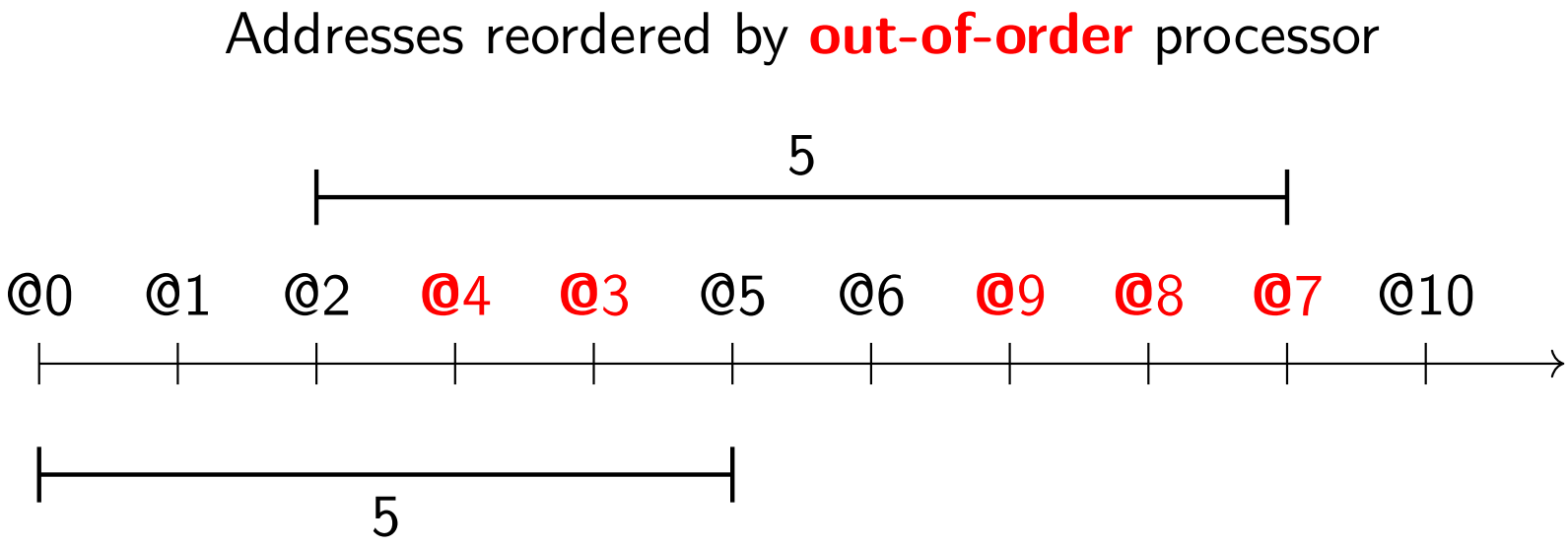


With which delta should I prefetch?

delta = 1 + 3 + 5 = 9 → always hit

Motivation: Why a new delta prefetcher?

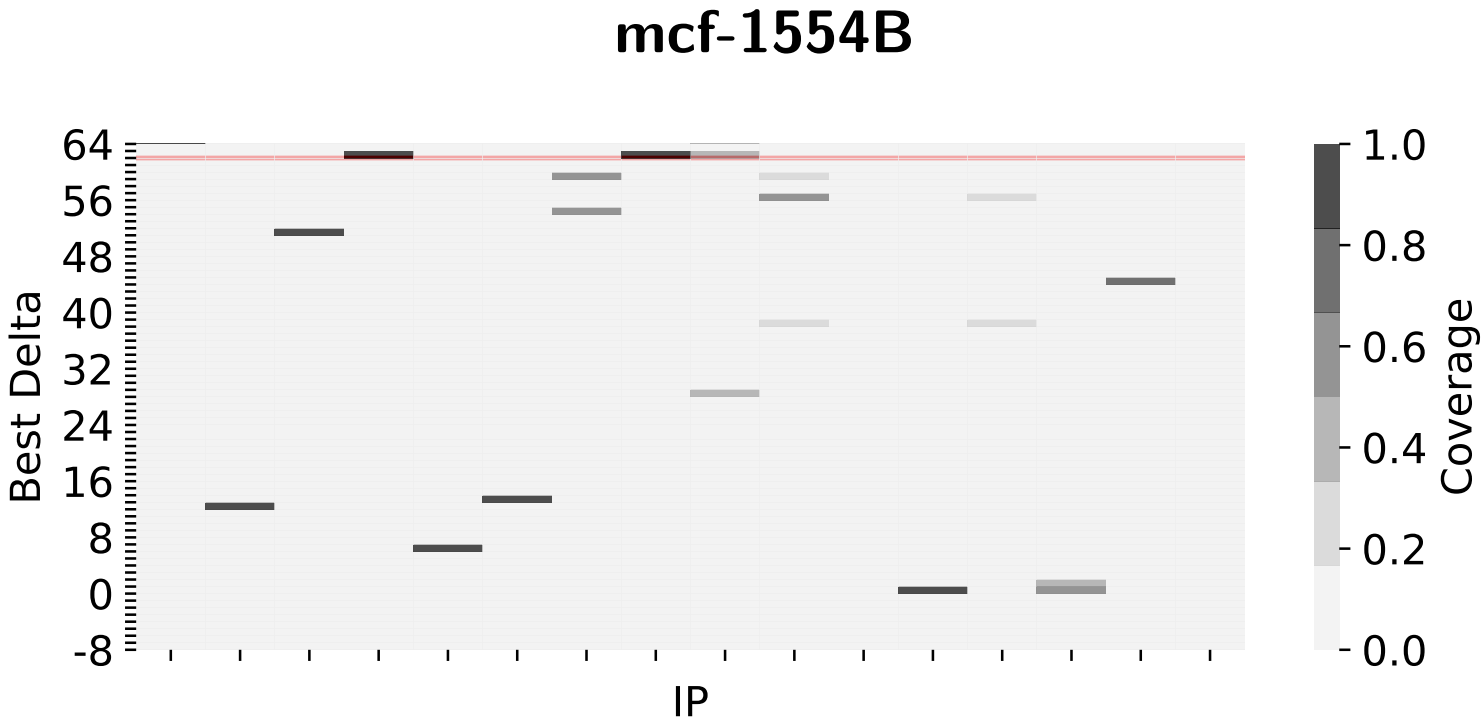
Out-of-order processors



Stride prefetch requires specific order
We can prefetch with delta = 5, for example

Motivation: Why a new delta prefetcher?

Why local Delta?

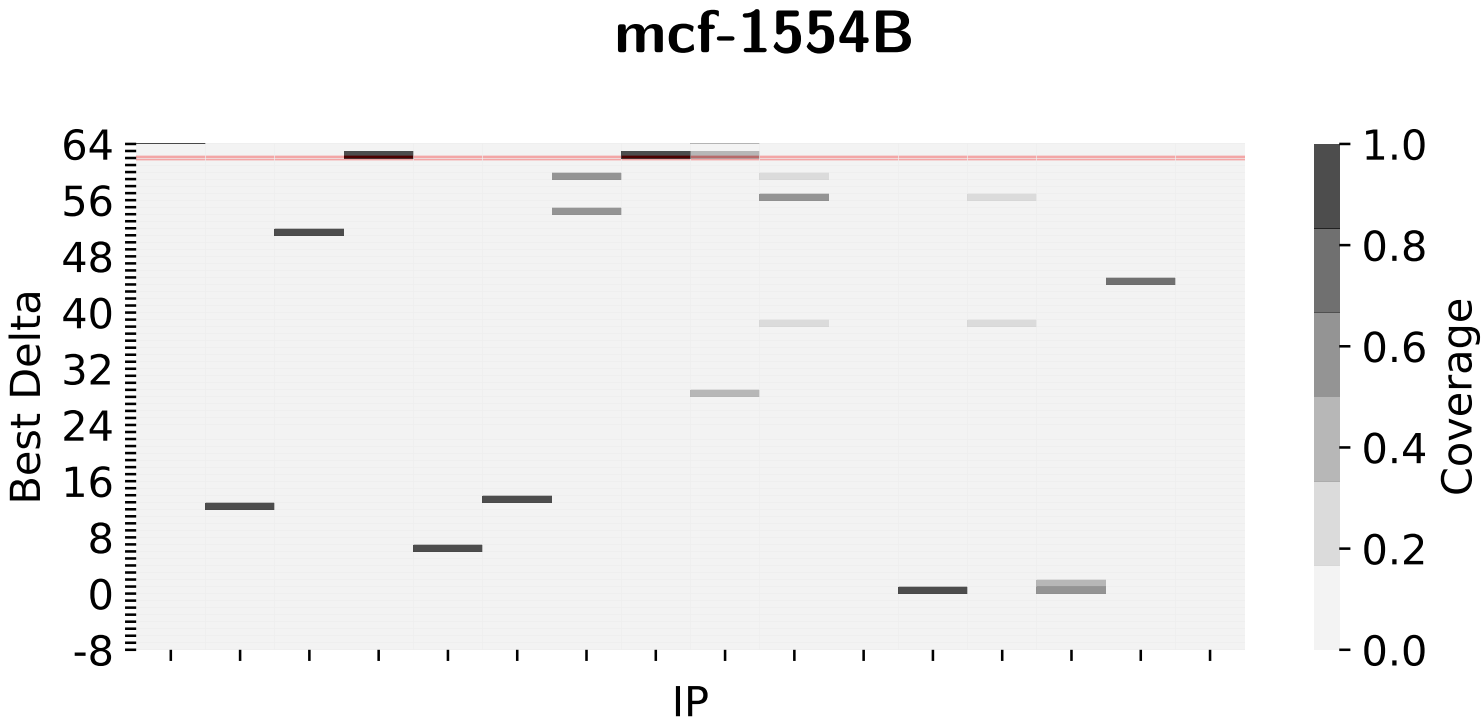


- **Red line**: best delta by BOP[1]
- **Gray lines**: per-IP local deltas

[1] Winner of 2nd Data Prefetching Championship (DPC2)

Motivation: Why a new delta prefetcher?

Why local Delta?



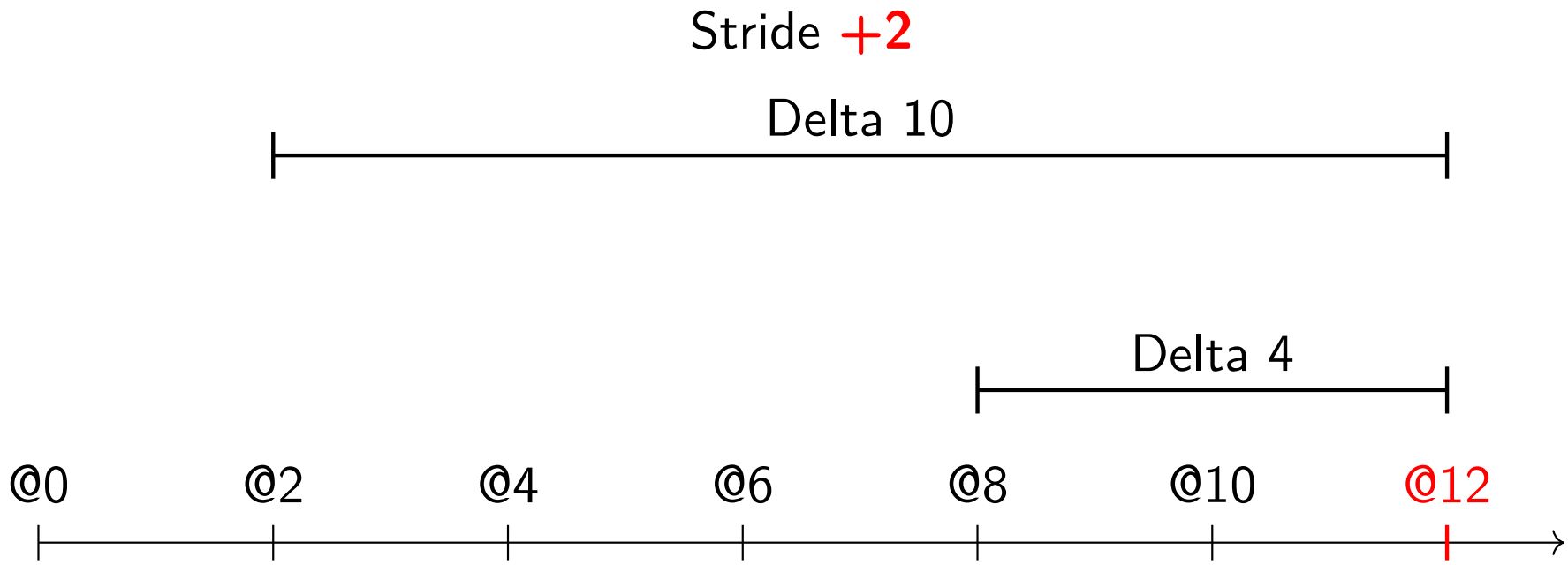
- **Red line**: best delta by I
- **Gray lines**: per-IP local

Every IP can have its own deltas!

[1] Winner of 2nd Data Prefetching Championship (DPC2)

Motivation: Why a new delta prefetcher?

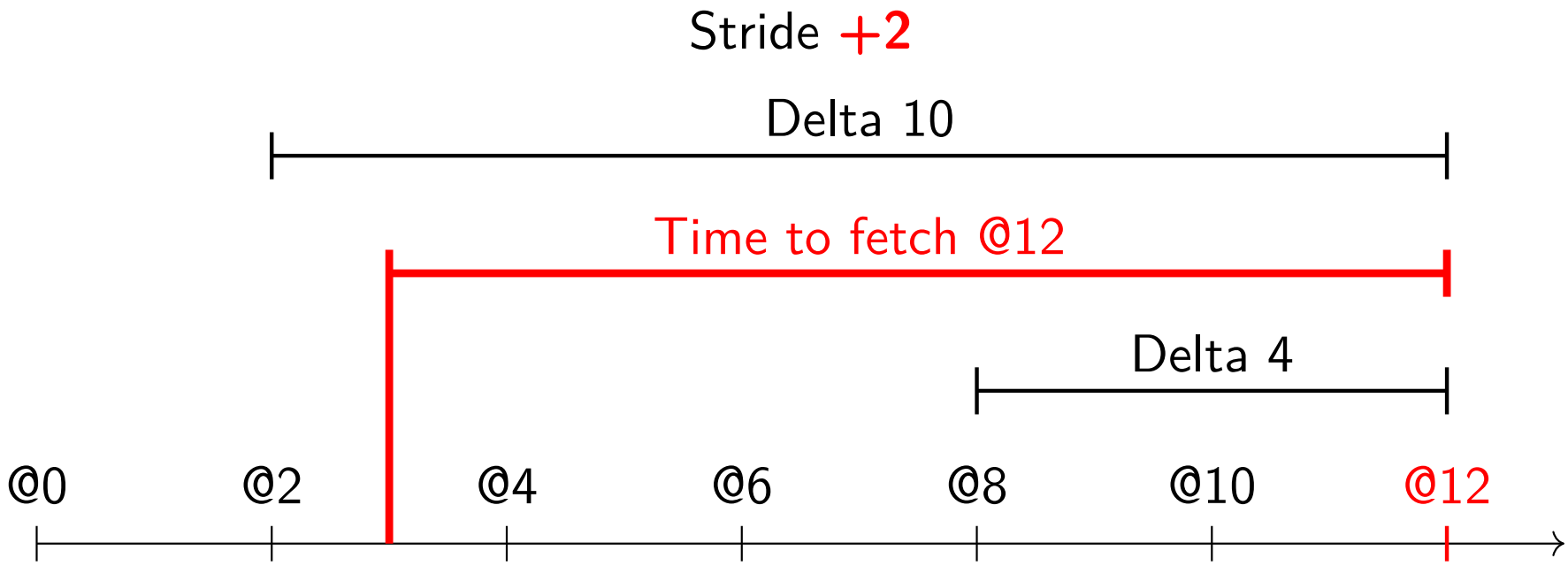
How far in advance should I prefetch?



How far in advance should I prefetch address 12?

Motivation: Why a new delta prefetcher?

How far in advance should I prefetch?



How far in advance should I prefetch address 12?
Depends on its latency

Berti

Training

1. Measure fetch latency
2. Learn timely and accurate deltas
3. Compute coverage of deltas

History table

IP	@	Time
A	2	0
A	5	30
B	10	50

Table of deltas

IP	Delta	Coverage	Destination

Berti

Training

1. Measure fetch latency
2. Learn timely and accurate deltas
3. Compute coverage of deltas

History table

IP	@	Time
A	2	0
A	5	30
B	10	50
A	12	70

Lat: 60

Table of deltas

IP	Delta	Coverage	Destination

Berti

Training

1. Measure fetch latency
2. Learn timely and accurate deltas
3. Compute coverage of deltas

History table

IP	@	Time
A	2	0
A	5	30
B	10	50
A	12	70

+10

Table of deltas

IP	Delta	Coverage	Destination

Berti

Training

1. Measure fetch latency
2. Learn timely and accurate deltas
3. Compute coverage of deltas

History table

IP	@	Time
A	2	0
A	5	30
B	10	50
A	12	70

Table of deltas

IP	Delta	Coverage	Destination
A	+10	1/1 (100%)	

Berti

Training

1. Measure fetch latency
2. Learn timely and accurate deltas
3. Compute coverage of deltas

Lat: 90

History table

IP	@	Time
A	2	0
A	5	30
B	10	50
A	12	70
A	15	140

Table of deltas

IP	Delta	Coverage	Destination
A	+10	1/1 (100%)	

Berti

Training

1. Measure fetch latency
2. Learn timely and accurate deltas
3. Compute coverage of deltas

History table

IP	@	Time
A	2	0
A	5	30
B	10	50
A	12	70
A	15	140

+13
+10

Table of deltas

IP	Delta	Coverage	Destination
A	+10	1/1 (100%)	

Berti

Training

1. Measure fetch latency
2. Learn timely and accurate deltas
3. Compute coverage of deltas

History table

IP	@	Time
A	2	0
A	5	30
B	10	50
A	12	70
A	15	140

Table of deltas

IP	Delta	Coverage	Destination
A	+10	2/2 (100%)	
A	+13	1/2 (50%)	

Berti

Issuing prefetch requests

1. Select deltas
2. Orchestration

History table

IP	@	Time
A	2	0
A	5	30
B	10	50
A	12	70
A	15	140

Table of deltas

IP	Delta	Coverage	Destination
A	+10	2/2 (100%)	
A	+13	1/2 (50%)	

Berti

Issuing prefetch requests

1. Select deltas
2. Orchestration

History table

IP	@	Time
A	2	0
A	5	30
B	10	50
A	12	70
A	15	140

Table of deltas

IP	Delta	Coverage	Destination
A	+10	2/2 (100%)	L1D
A	+13	1/2 (50%)	L2

Coverage

$> \mathbf{65\%} \rightarrow L1D$

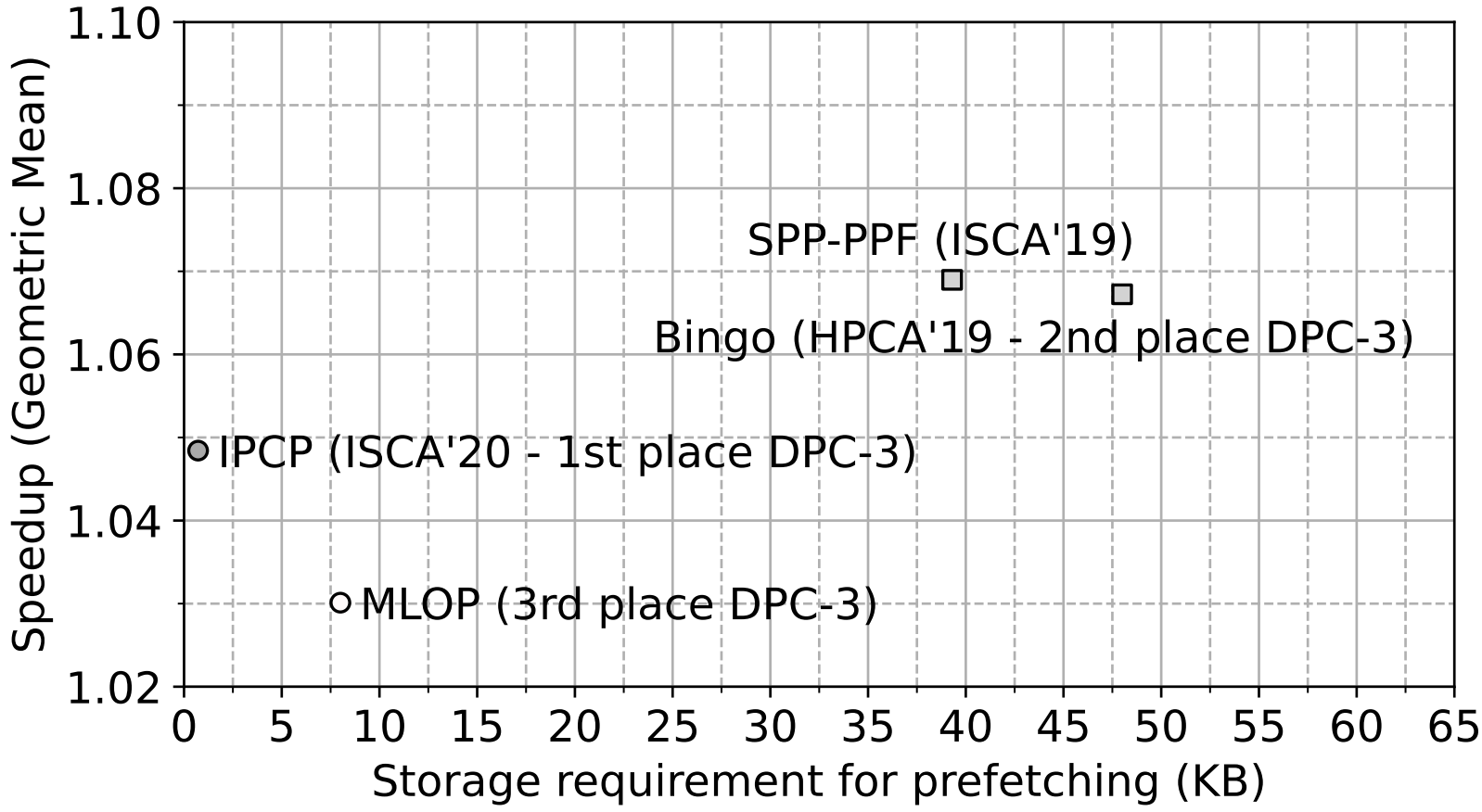
$> \mathbf{35\%} \rightarrow L2$

Evaluation

- ChampSim (Intel Sunny Cove)
- Normalized to IP-Stride (Intel L1D)
- SPEC CPU & GAP traces

Evaluation

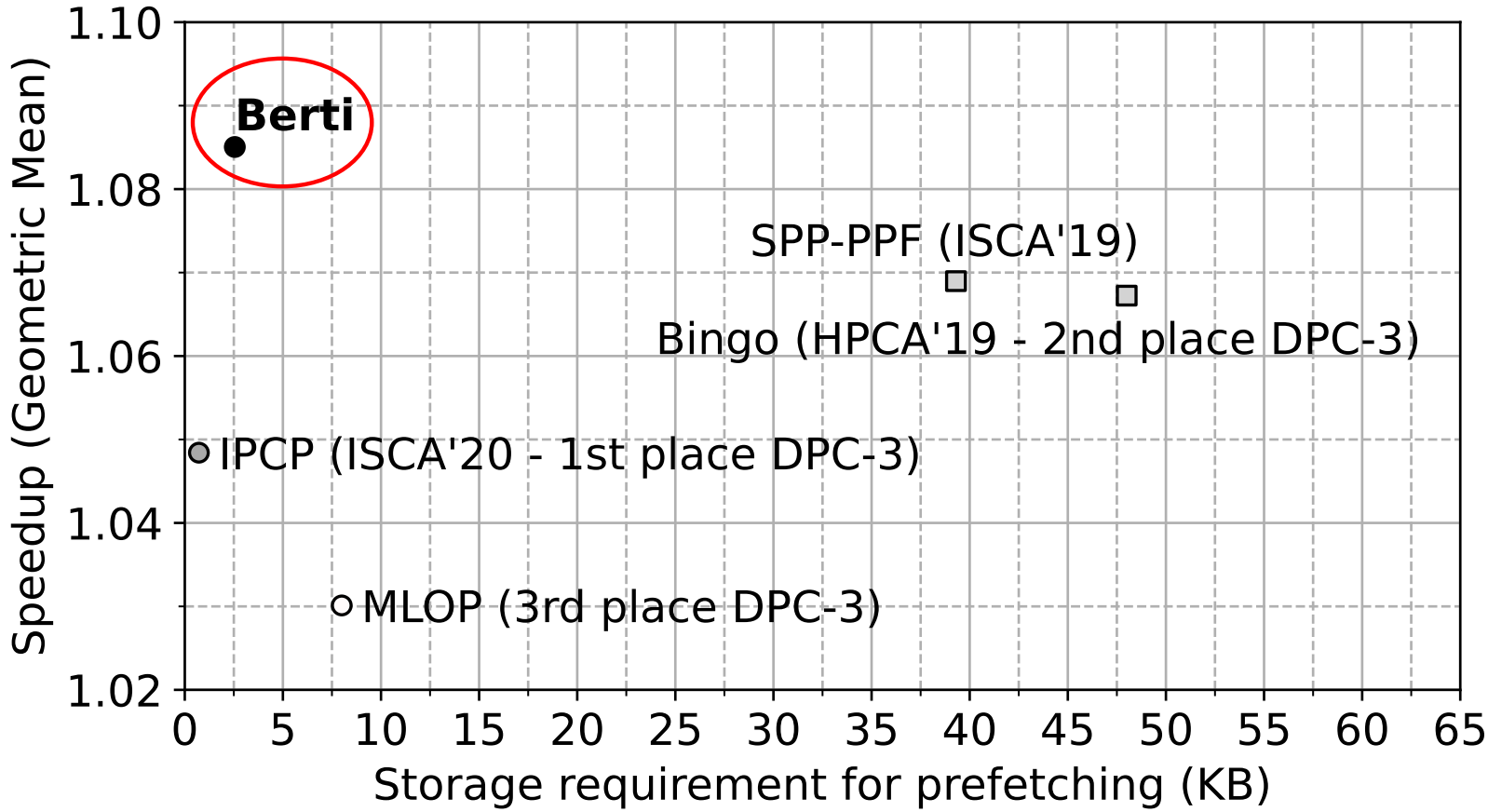
Single-thread performance



DPC-3 = 3rd Data Prefetching Championship

Evaluation

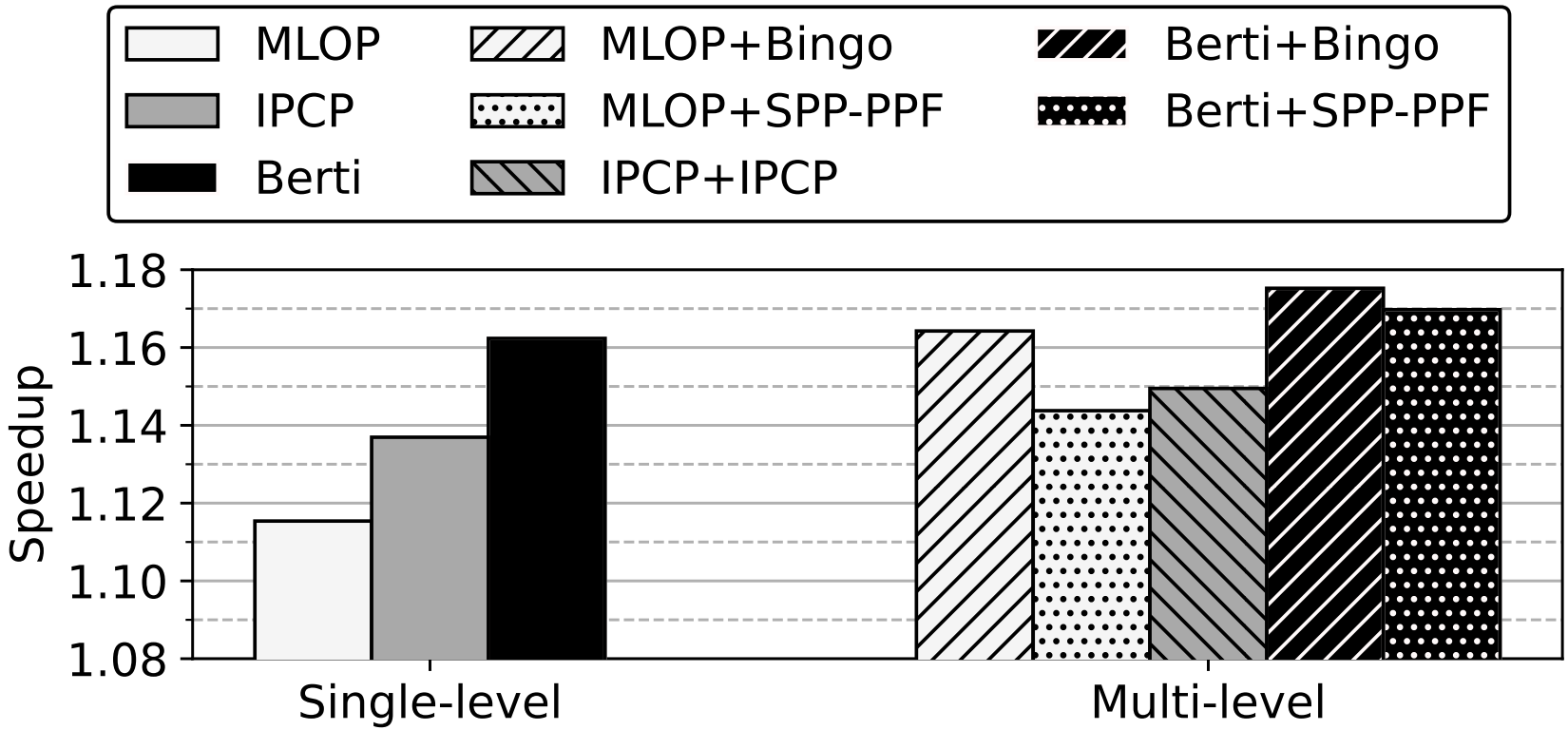
Single-thread performance



DPC-3 = 3rd Data Prefetching Championship

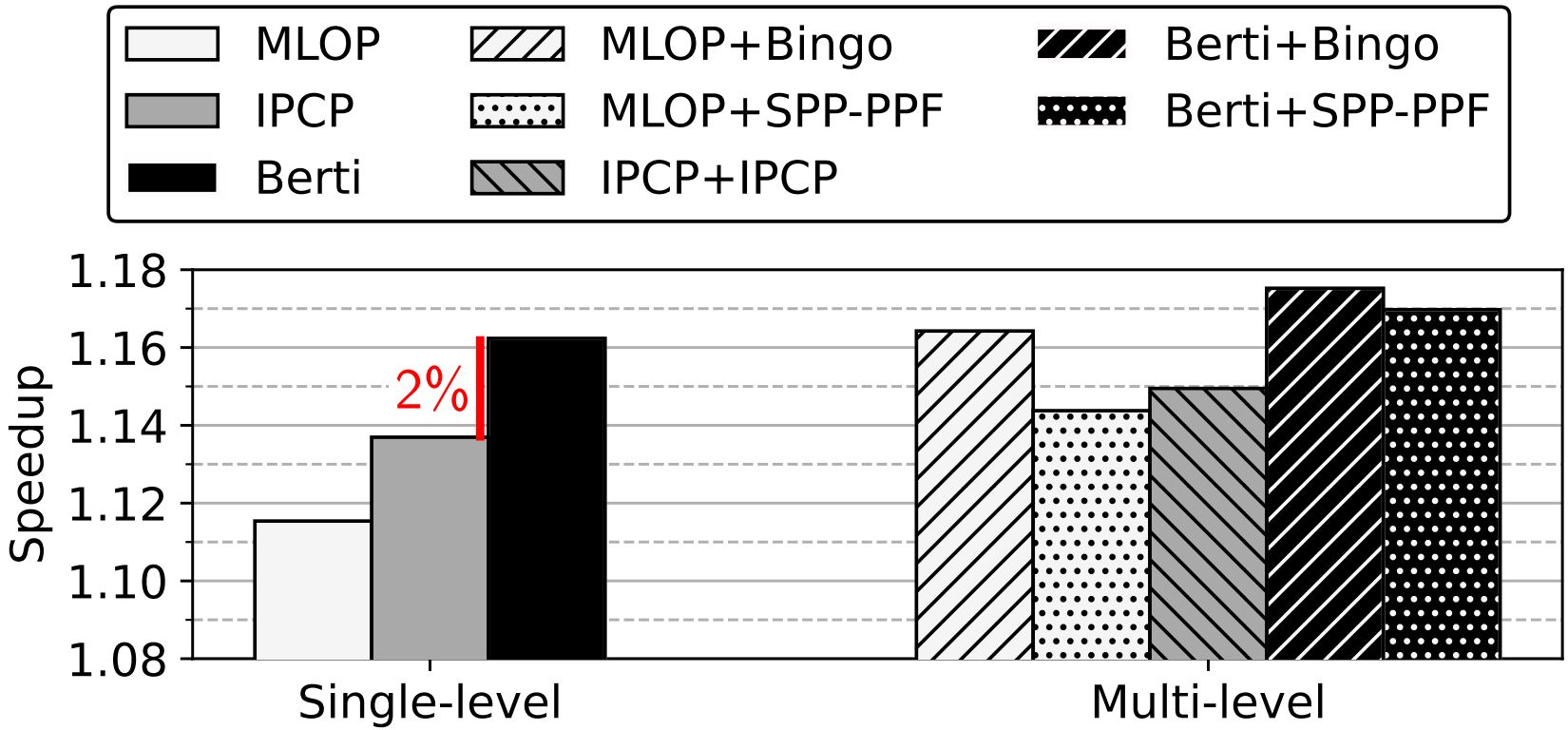
Evaluation

Multicore performance



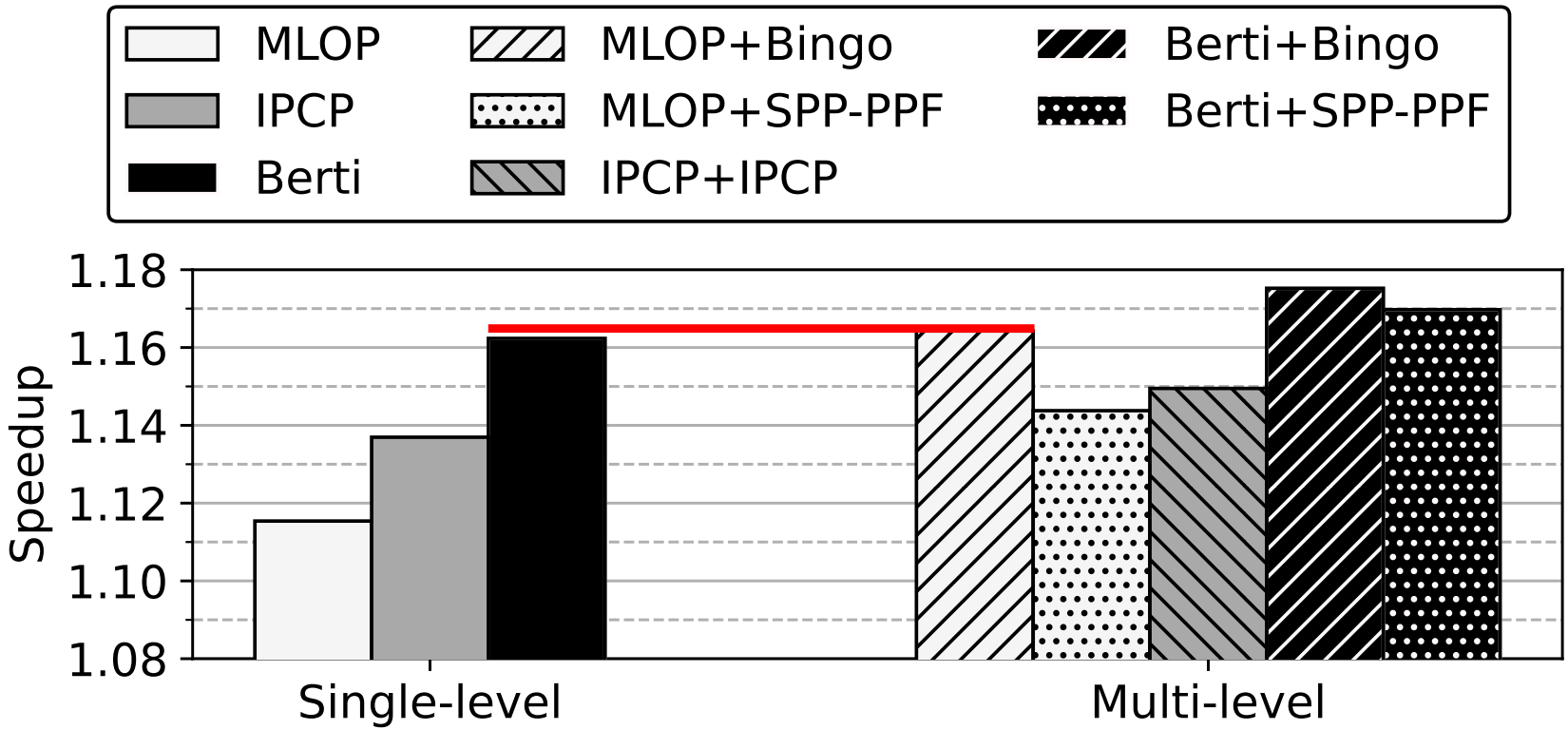
Evaluation

Multicore performance



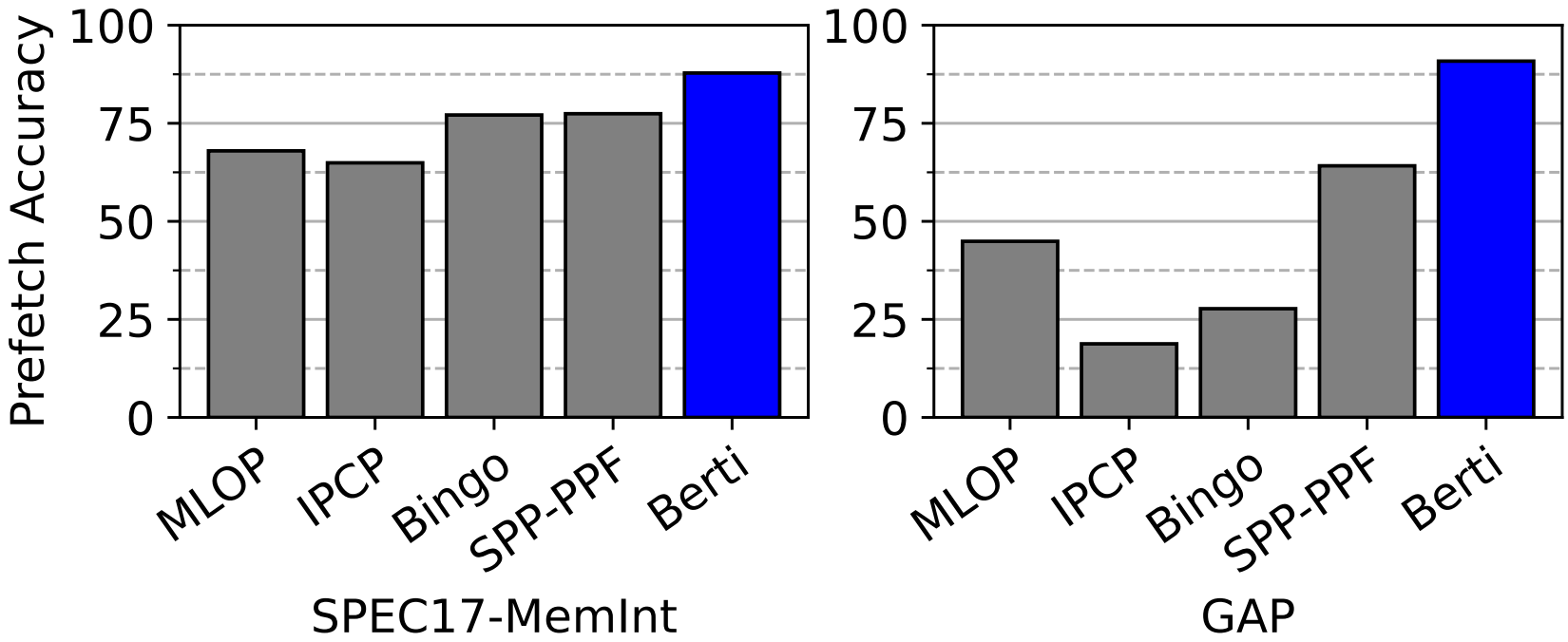
Evaluation

Multicore performance



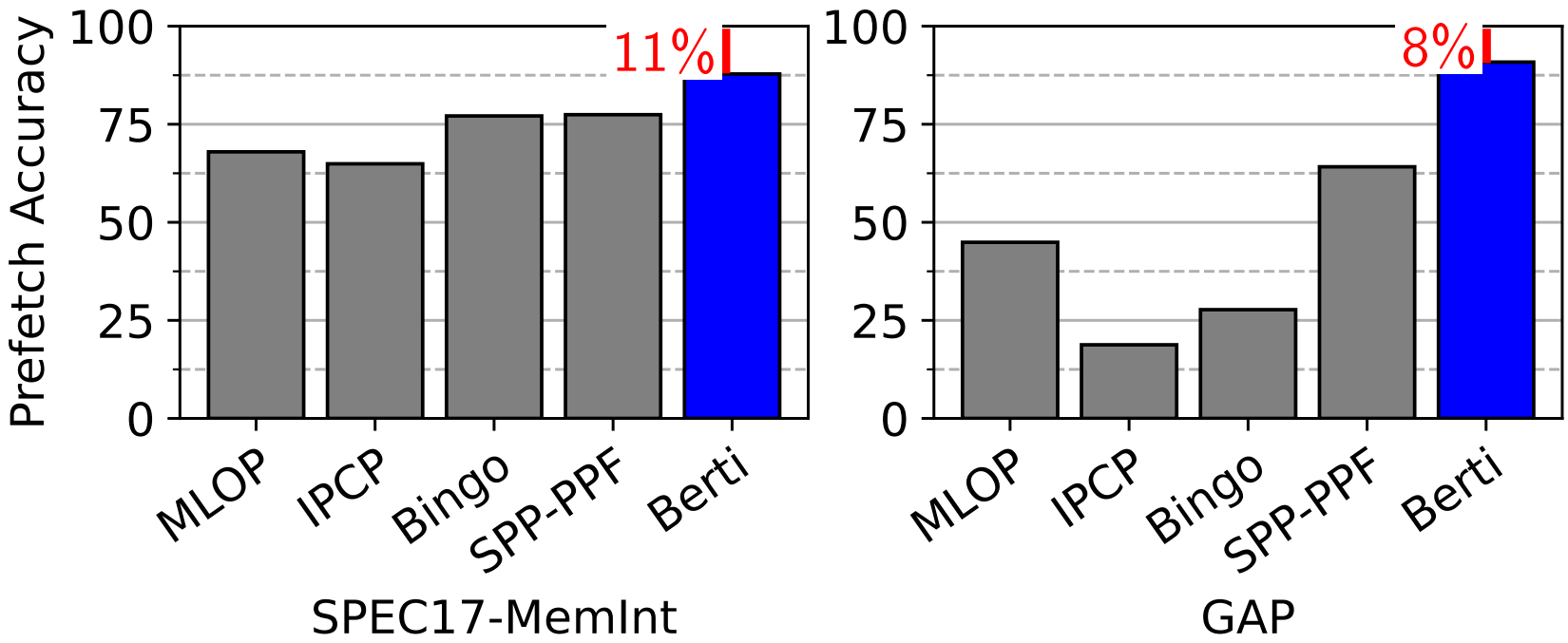
Evaluation

Accuracy



Evaluation

Accuracy



Conclusion Remarks

- **Accurate** and **timely local delta L1D** prefetcher
- Learns the best **deltas** to prefetch
- Outperforms state-of-the-art prefetchers by 3.5%
- Only 2.55KiB of storage overhead

- With more cores in CMPs we need more efficient shared resources

- With more cores in CMPs we need more efficient shared resources
- **We evaluated a real system memory hierarchy**

- With more cores in CMPs we need more efficient shared resources
- **We evaluated a real system memory hierarchy**
  - Utilization of SLLC is uneven
  - Hardware prefetching boost performance

- With more cores in CMPs we need more efficient shared resources
- **We evaluated a real system memory hierarchy**
  - Utilization of SLLC is uneven
  - Hardware prefetching boost performance
- **We propose a new mechanism for SLLC and BW management (Balancer)**

- With more cores in CMPs we need more efficient shared resources
- **We evaluated a real system memory hierarchy**
  - Utilization of SLLC is uneven
  - Hardware prefetching boost performance
- **We propose a new mechanism for SLLC and BW management (Balancer)**
  - Dynamically manages the amount of SLLC and DRAM BW
  - It can improve performance and/or fairness

- With more cores in CMPs we need more efficient shared resources
- **We evaluated a real system memory hierarchy**
  - Utilization of SLLC is uneven
  - Hardware prefetching boost performance
- **We propose a new mechanism for SLLC and BW management (Balancer)**
  - Dynamically manages the amount of SLLC and DRAM BW
  - It can improve performance and/or fairness
- **More cores →high multi-threading application → complex synchronization**

- With more cores in CMPs we need more efficient shared resources
- **We evaluated a real system memory hierarchy**
  - Utilization of SLLC is uneven
  - Hardware prefetching boost performance
- **We propose a new mechanism for SLLC and BW management (Balancer)**
  - Dynamically manages the amount of SLLC and DRAM BW
  - It can improve performance and/or fairness
- **More cores →high multi-threading application → complex synchronization**

- With more cores in CMPs we need more efficient shared resources
- **We evaluated a real system memory hierarchy**
  - Utilization of SLLC is uneven
  - Hardware prefetching boost performance
- **We propose a new mechanism for SLLC and BW management (Balancer)**
  - Dynamically manages the amount of SLLC and DRAM BW
  - It can improve performance and/or fairness
- **More cores →high multi-threading application → complex synchronization**
  - Classical sync. and HTM have similar performance
  - HTM scales better with high-contention & number of threads

- With more cores in CMPs we need more efficient shared resources
- **We evaluated a real system memory hierarchy**
  - Utilization of SLLC is uneven
  - Hardware prefetching boost performance
- **We propose a new mechanism for SLLC and BW management (Balancer)**
  - Dynamically manages the amount of SLLC and DRAM BW
  - It can improve performance and/or fairness
- **More cores →high multi-threading application → complex synchronization**
  - Classical sync. and HTM have similar performance
  - HTM scales better with high-contention & number of threads
- **Hardware prefetchers are good, but sometimes inaccurate**

- With more cores in CMPs we need more efficient shared resources
- **We evaluated a real system memory hierarchy**
  - Utilization of SLLC is uneven
  - Hardware prefetching boost performance
- **We propose a new mechanism for SLLC and BW management (Balancer)**
  - Dynamically manages the amount of SLLC and DRAM BW
  - It can improve performance and/or fairness
- **More cores →high multi-threading application → complex synchronization**
  - Classical sync. and HTM have similar performance
  - HTM scales better with high-contention & number of threads
- **Hardware prefetchers are good, but sometimes inaccurate**
  - Identify memory patterns from deltas
  - High accuracy & performance

Future Work

- Many-cores/low-thread (Intel, AMD, Arm) $\rightarrow$ low-cores count/many-threads (IBM Power10)
- New control mechanism to control hardware prefetching
- Hardware prefetching: *TLB*, *irregular patterns*...

Contributions to high-performance memory hierarchies: program characterization, resource control, transactional synchronization, and hardware prefetching

Agustín Navarro Torres

Supervisors: Pablo Enrique Ibáñez Marín & Jesús Alastruey Benedé

April 19, 2023

Universidad Zaragoza 1542

Grupo de Investigación en Arquitectura de Computadores (gaZ)
Universidad Zaragoza 1542

Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

Instituto Universitario de Investigación de Ingeniería de Aragón
Universidad Zaragoza

Memory hierarchy characterization

- New benchmarks requires new characterization
- At the time only two papers about SPEC CPU2017 characterization[2][3]:
  - Use “old” Intel system (Haswell)
  - They do not study SLLC and hardware prefetching sensitivity
- First paper to study SLLC size sensitivity on a real system.

[2] A. Limaye et. al , ”A Workload Characterization of the SPEC CPU2017 Benchmark Suite,” 2018 ISPASS

[3] Reena Panda et. al ”Wait of a decade: Did spec cpu 2017 broaden the performance horizon?” 2018 HPCA

Balancer

- A lot of papers about cache partition on a real system
- No paper does this on an SLLC clustered system
- Only two papers manage SLLC and memory bandwidth at the same time[4][5]

[4] Shuang Chen et al. "Parties: Qos-aware resource partitioning for multiple interactive services." 2019 ASPLOS

[5] Jinsu Park et al. "Copart: Coordinated partitioning of last-level cache and memory bandwidth for fairness-aware workload consolidation on commodity servers." 2019 EuroSys

Synchronization

- Most of works focus on a single synchronization strategy and make a deep-dive into its scalability performance and issues.[6][7]
- Others works analyze performance of several synchronization strategies, but on low thread count[8]

[6] Timoteo M. Rico et al. "Energy consumption and scalability evaluation for software trans- actional memory on a real computing environment." 2015 SBAC-PAD

[7] Hugo Guiroux et al. "Multicore locks: The case is not closed yet." 2016 USENIX

[8] Richard M. Yoo et al. "Performance evaluation of Intel® transactional synchronization extensions for high-performance computing." 2013 SC