UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Contention-Aware Scheduling for SMT Multicore Processors

Author:

Josué Feliu Pérez

Advisors:

Julio Sahuquillo Borrás

Salvador V. Petit Martí

València, February 22, 2017

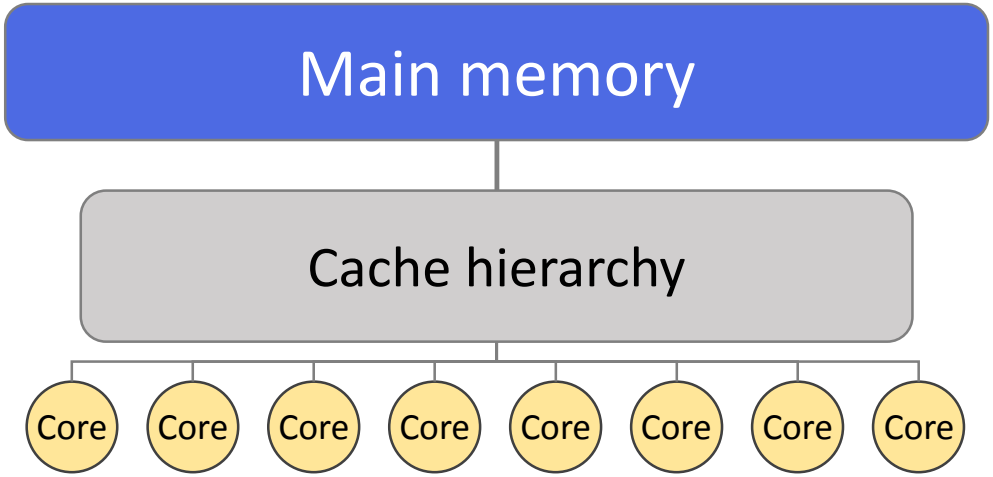
Introduction

- Multicore processors are the common implementation

Introduction

- Multicore processors are the common implementation

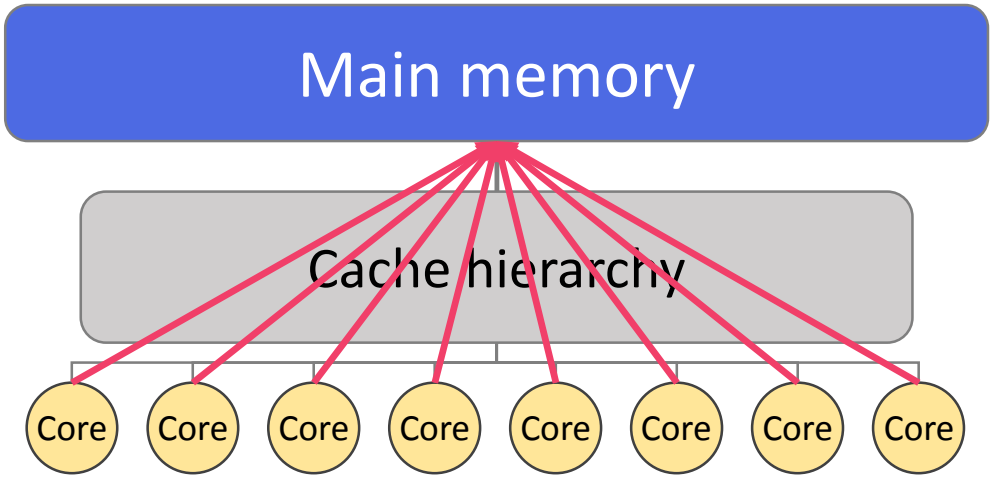
- Memory bandwidth rises as a known performance bottleneck



Introduction

- Multicore processors are the common implementation

- Memory bandwidth rises as a known performance bottleneck

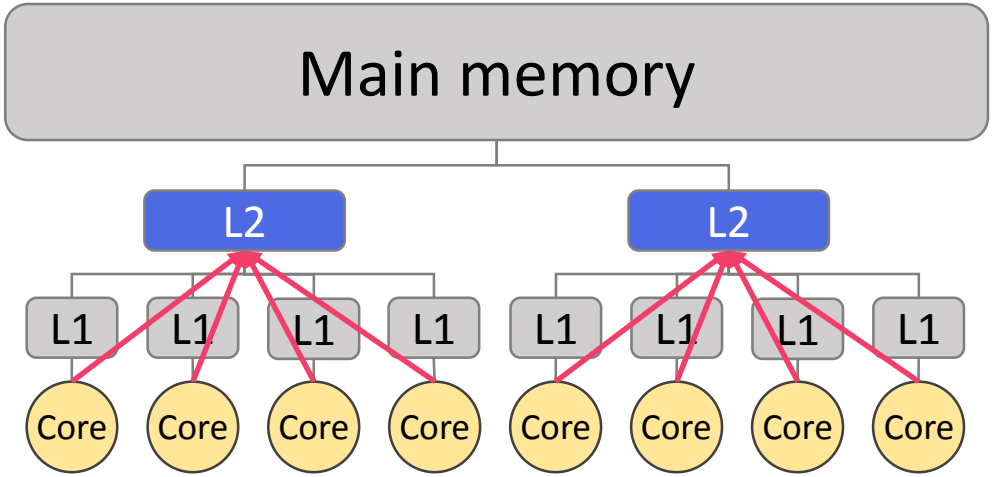


Introduction

- Multicore processors are the common implementation

- Memory bandwidth rises as a known performance bottleneck

Caches also suffers bandwidth contention

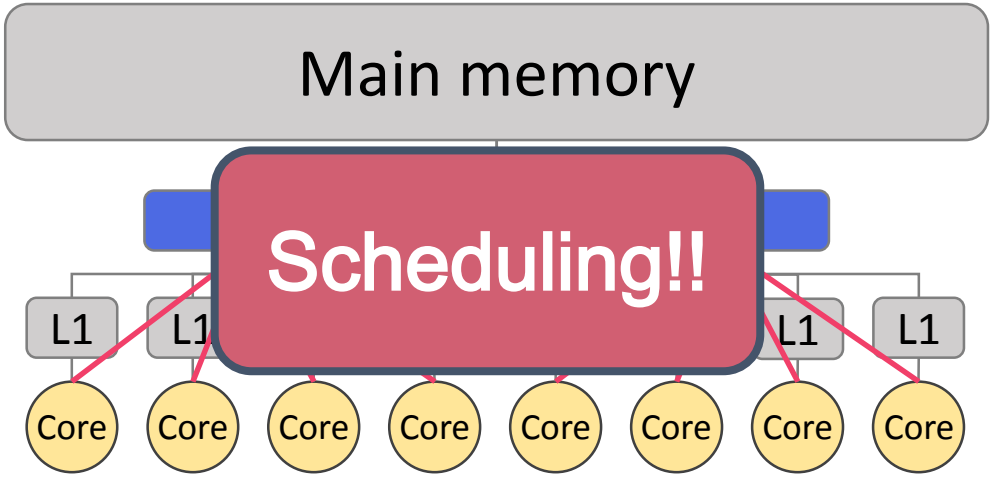


Introduction

- Multicore processors are the common implementation

- Memory bandwidth rises as a known performance bottleneck

Caches also suffers bandwidth contention

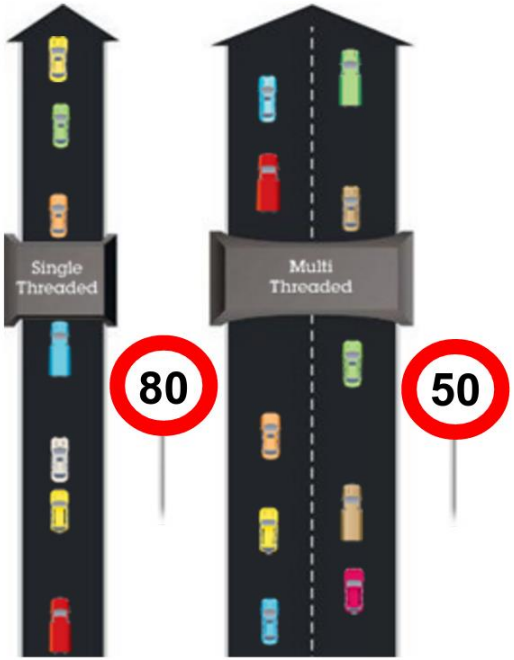


Introduction

- Simultaneous multithreading (SMT)

Introduction

- Simultaneous multithreading (SMT) improves processor throughput
  - Exploit instruction-level and thread-level parallelism

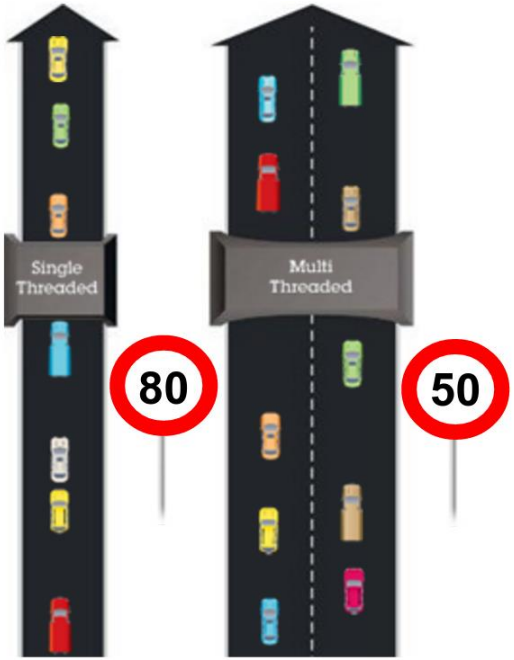


*Which approach is designed for the highest volume** of traffic? Which road is faster?*

Two lanes at 50 carry 25% more volume if traffic density per lane is equal

Introduction

- Simultaneous multithreading (SMT) improves processor throughput
  - Exploit instruction-level and thread-level parallelism
- Key resources are shared

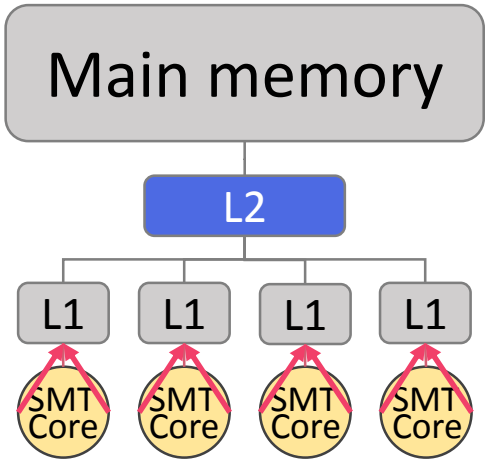


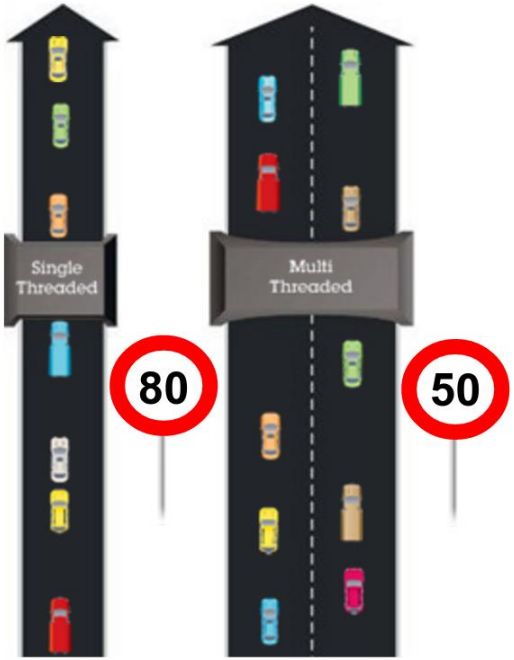
*Which approach is designed for the highest volume** of traffic? Which road is faster?*

Two lanes at 50 carry 25% more volume if traffic density per lane is equal

Introduction

- Simultaneous multithreading (SMT) improves processor throughput
  - Exploit instruction-level and thread-level parallelism
- Key resources are shared
  - L1 caches



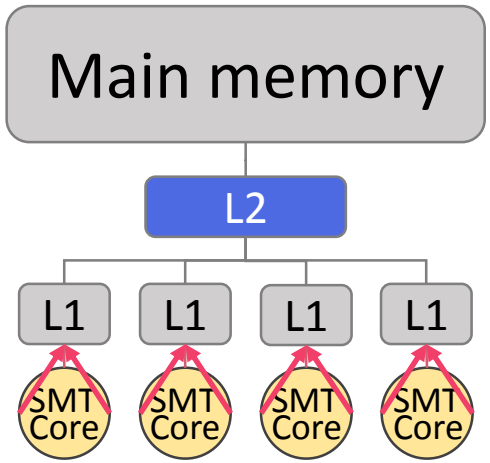


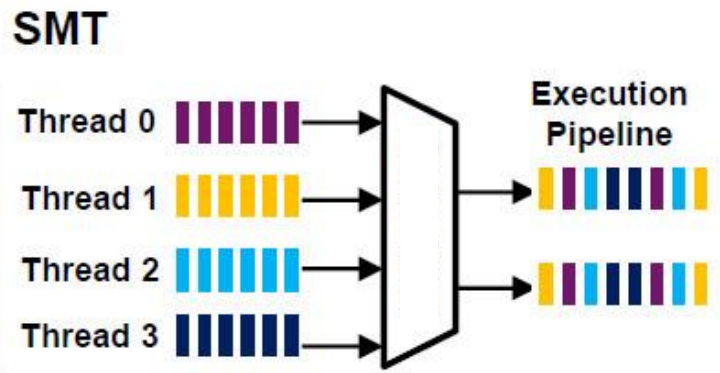
*Which approach is designed for the highest volume** of traffic? Which road is faster?*

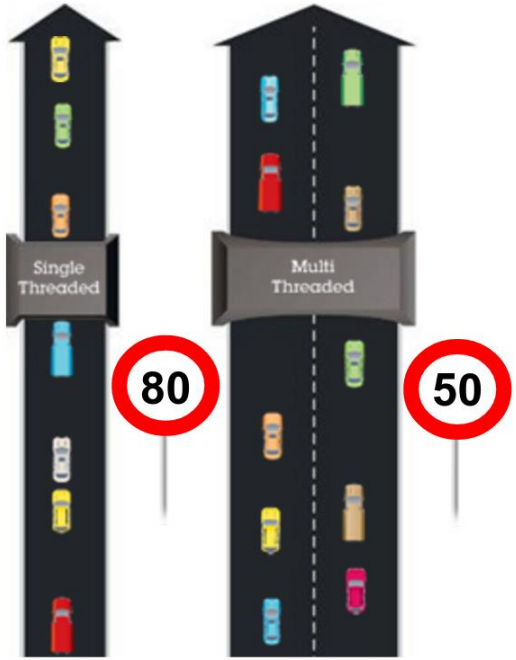
***Two lanes at 50 carry 25% more volume if traffic density per lane is equal*

Introduction

- Simultaneous multithreading (SMT) improves processor throughput
  - Exploit instruction-level and thread-level parallelism
- Key resources are shared
  - L1 caches, instructions queues, execution units, ROB, etc.





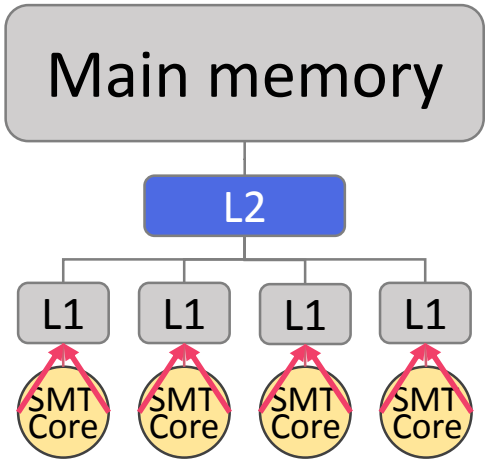


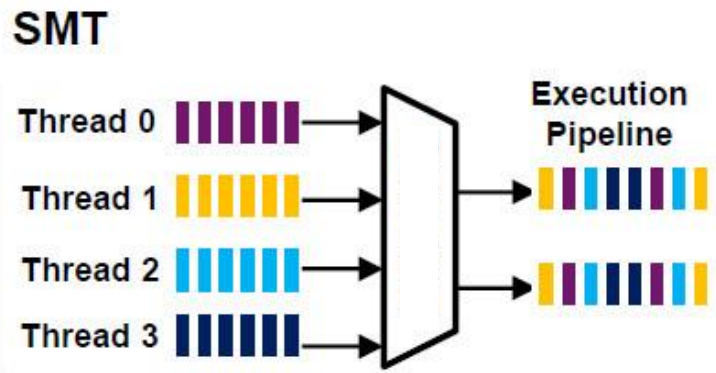
*Which approach is designed for the highest volume** of traffic? Which road is faster?*

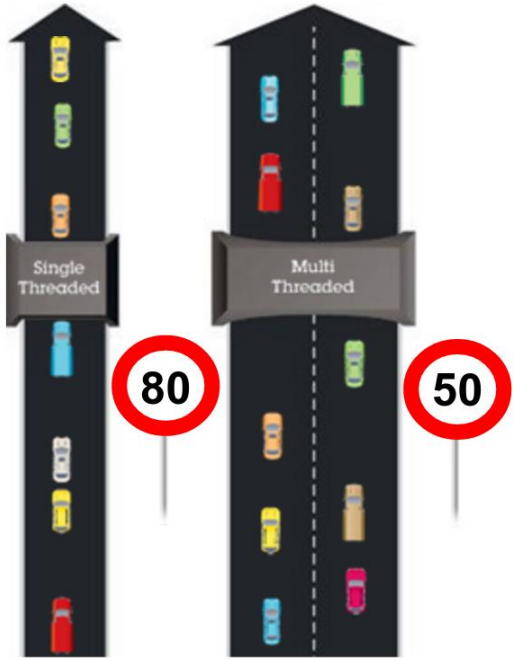
*** Two lanes at 50 carry 25% more volume if traffic density per lane is equal*

Introduction

- Simultaneous multithreading (SMT) improves processor throughput
  - Exploit instruction-level and thread-level parallelism
- Key resources are shared
  - L1 caches, instructions queues, execution units, ROB, etc.
- Performance depends on processes interference





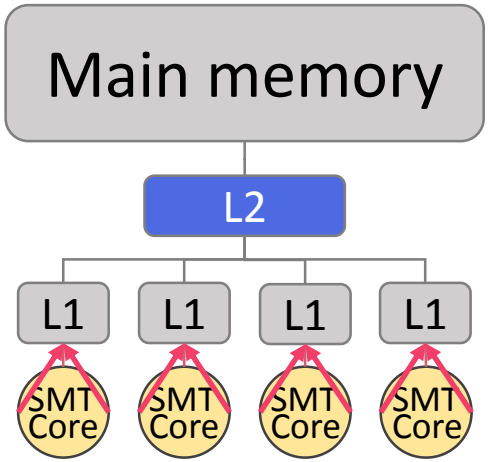


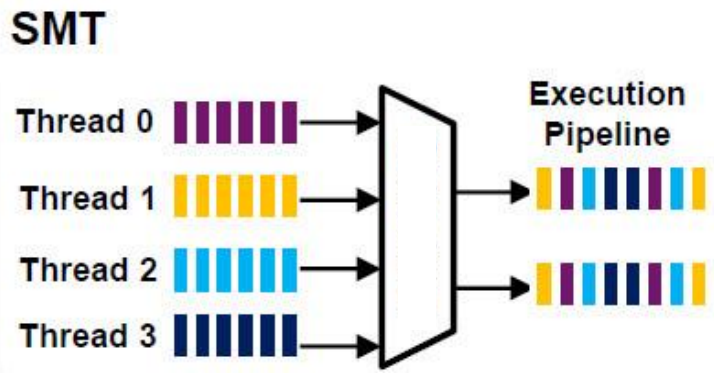
*Which approach is designed for the highest volume** of traffic? Which road is faster?*

*** Two lanes at 50 carry 25% more volume if traffic density per lane is equal*

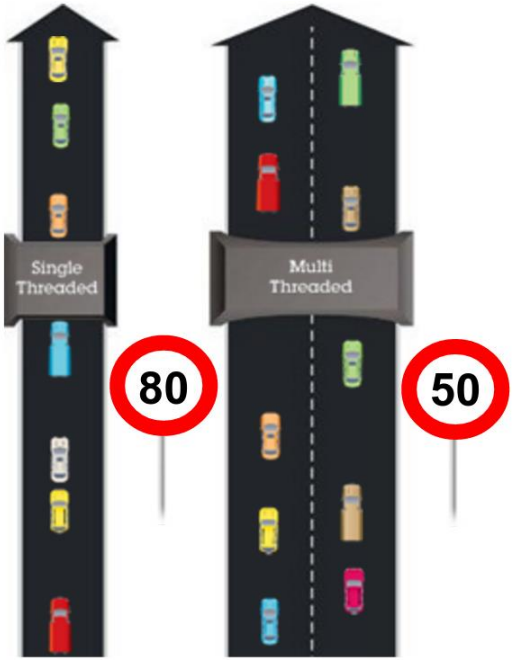
Introduction

- Simultaneous multithreading (SMT) improves processor throughput
  - Exploit instruction-level and thread-level parallelism
- Key resources are shared
  - L1 caches, instructions queues, execution units, ROB, etc.
- Performance depends on processes interference





Scheduling!!



*Which approach is designed for the highest volume** of traffic? Which road is faster?*

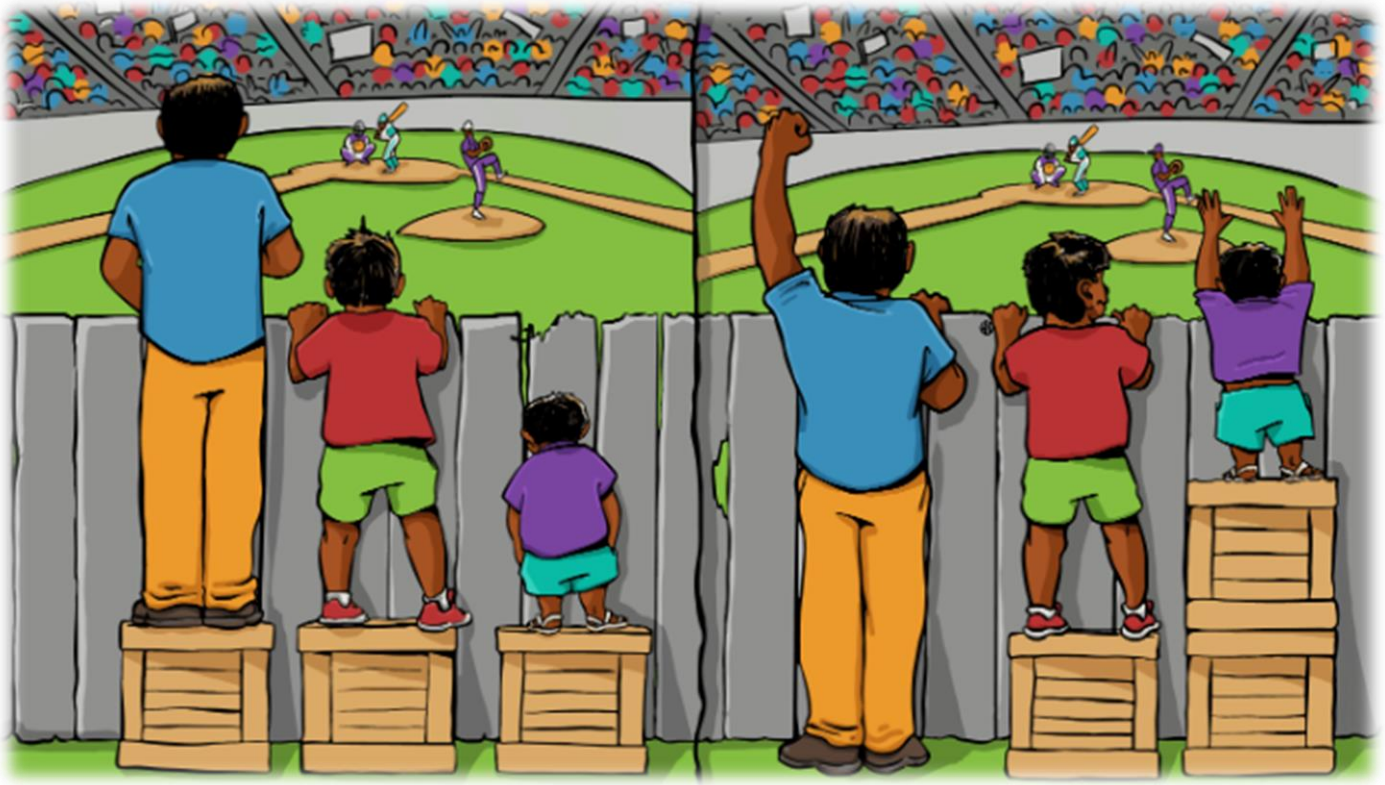
*** Two lanes at 50 carry 25% more volume if traffic density per lane is equal*

Introduction

- Sharing resources affects fairness

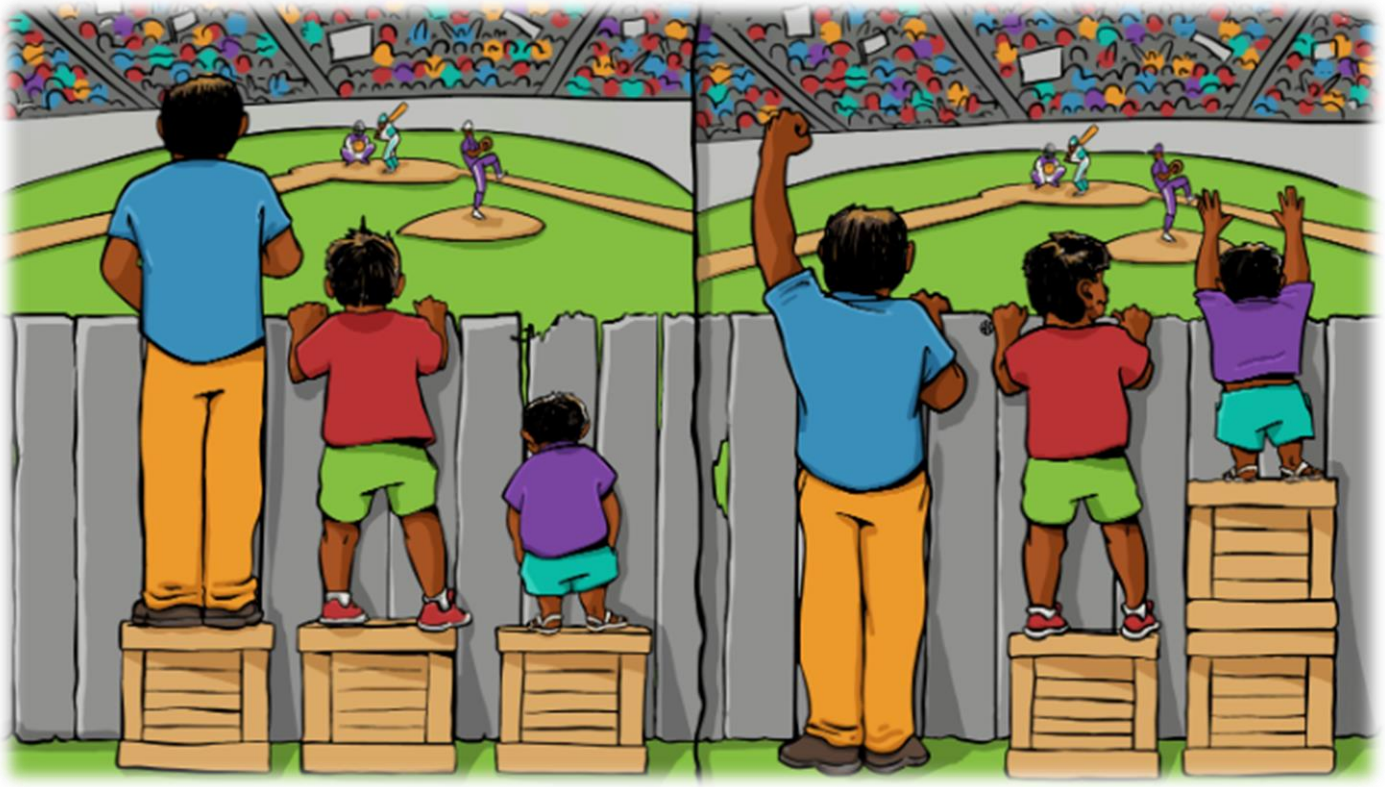
Introduction

- Sharing resources affects fairness
- Fairly sharing a resource is challenging



Introduction

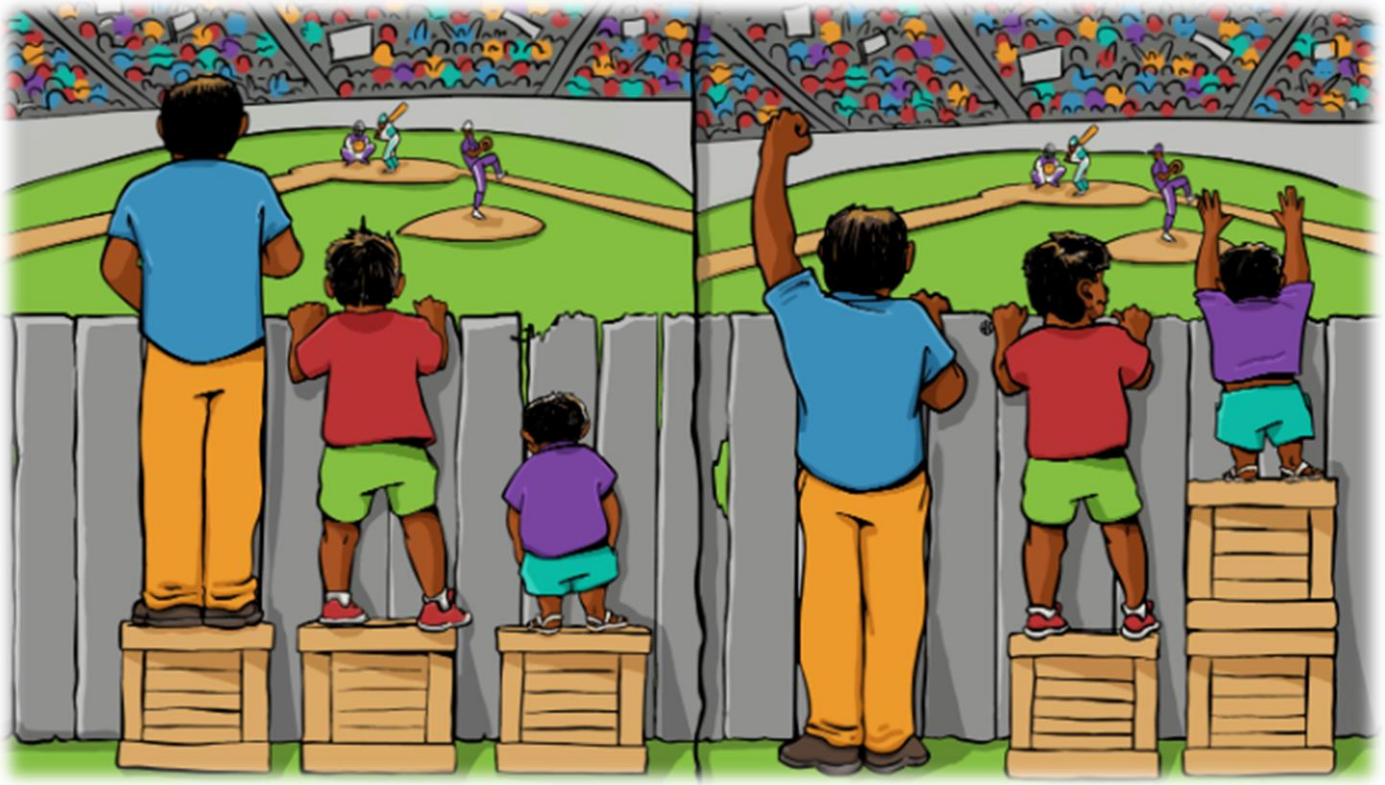
- Sharing resources affects fairness
- Fairly sharing a resource is challenging
- Unfairness has negative effects on the system behavior



Introduction

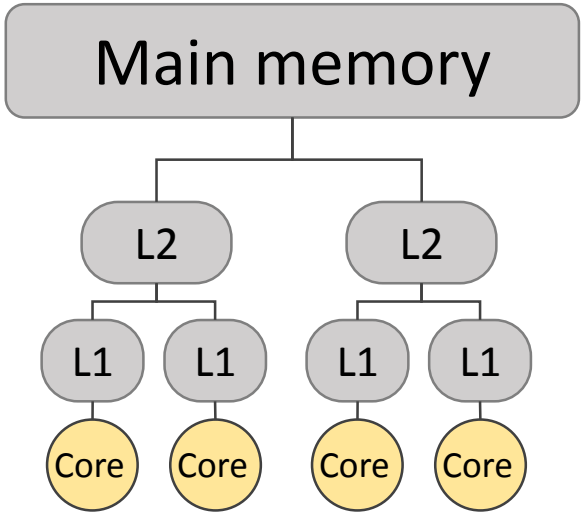
- Sharing resources affects fairness
- Fairly sharing a resource is challenging
- Unfairness has negative effects on the system behavior

Scheduling!!



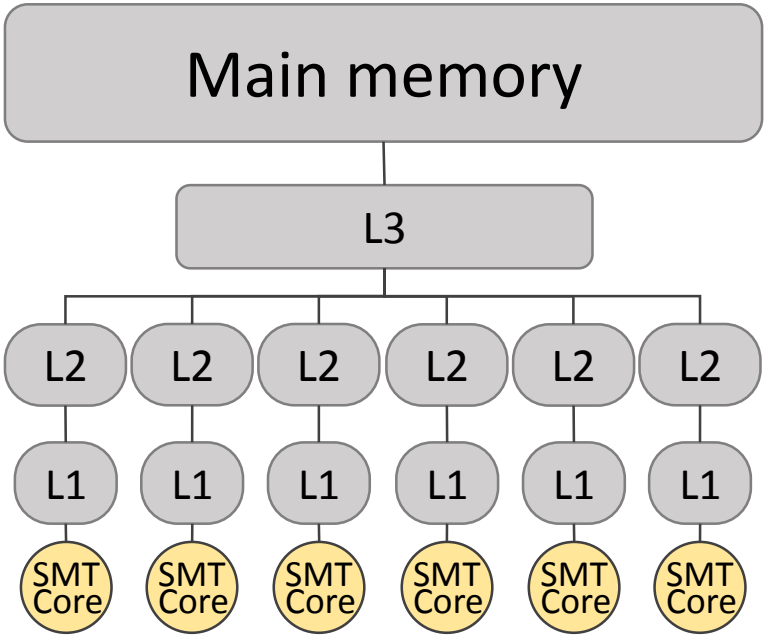
Methodology

- Experiments on real systems
  - Intel Xeon X3320



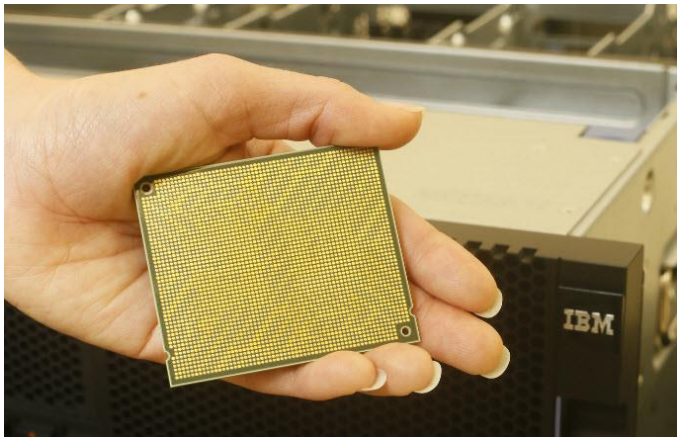
Methodology

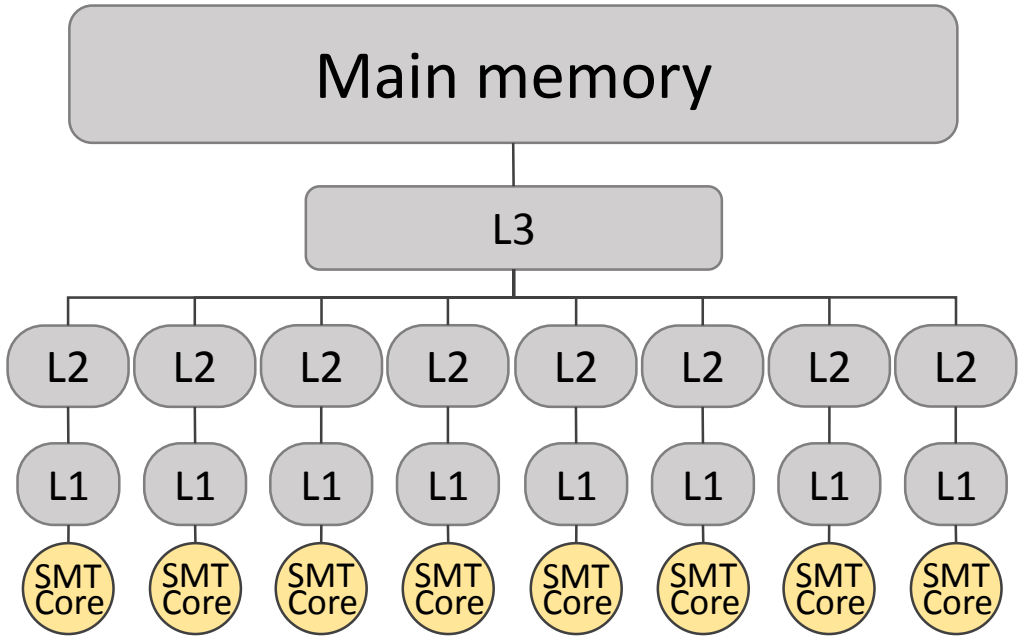
- Experiments on real systems
  - Intel Xeon X3320
  - Intel Xeon E5645



Methodology

- Experiments on real systems
  - Intel Xeon X3320
  - Intel Xeon E5645
  - IBM POWER8





Methodology

- Experiments on real systems
  - Intel Xeon X3320
  - Intel Xeon E5645
  - IBM POWER8
- Software process schedulers
  - Prototyped as a user-level applications in Linux
  - Guided by performance counters (*libfpm* library)
    - Processor specific events

Methodology

- Experiments on real systems
  - Intel Xeon X3320
  - Intel Xeon E5645
  - IBM POWER8
- Software process schedulers
  - Prototyped as a user-level applications in Linux
  - Guided by performance counters (*libfpm* library)
    - Processor specific events
- Multiprogram workloads with SPEC CPU2006 benchmarks

Outline

MAIN CONTRIBUTIONS

I. Bandwidth-Aware Scheduling on Multicores

II. Bandwidth-Aware Scheduling on SMT Multicores

III. Progress-Aware Scheduling on SMT Multicores

IV. Symbiotic Job Scheduling on the IBM POWER8

Outline

I. Bandwidth-Aware Scheduling on Multicores
   I. Performance degradation analysis
   II. Memory-hierarchy bandwidth-aware scheduling
   III. Experimental evaluation (I)
   IV. IPC-degradation memory-hierarchy bandwidth-aware scheduling
   V. Experimental evaluation (II)

II. Bandwidth-Aware Scheduling on SMT Multicores

III. Progress-Aware Scheduling on SMT Multicores

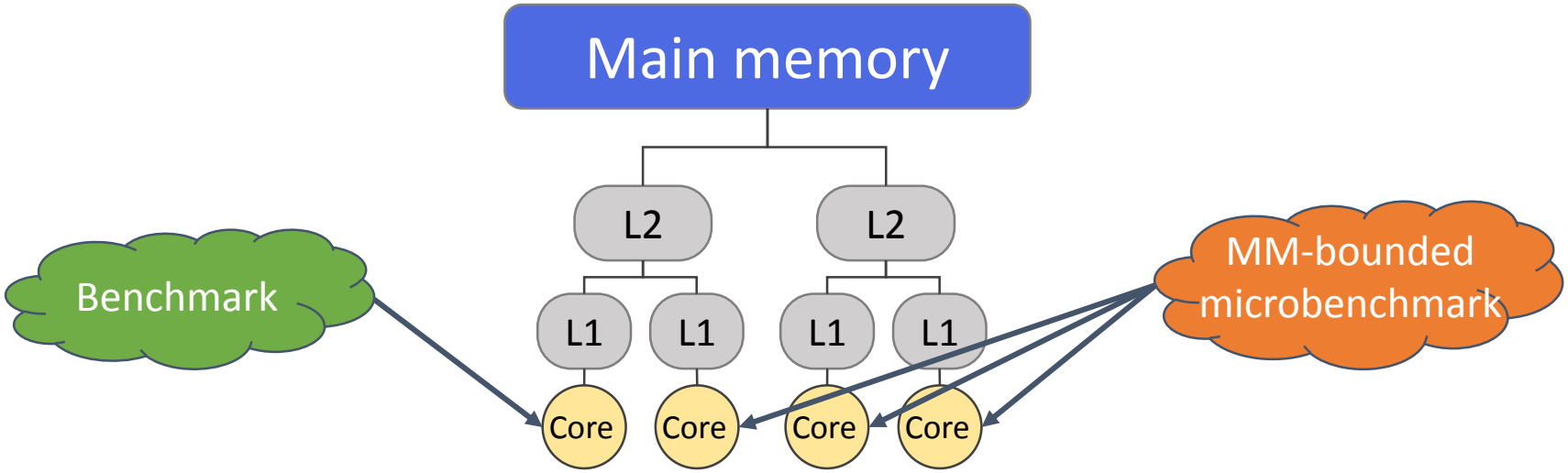
IV. Symbiotic Job Scheduling on the IBM POWER8

Performance degradation analysis

Main memory bandwidth contention

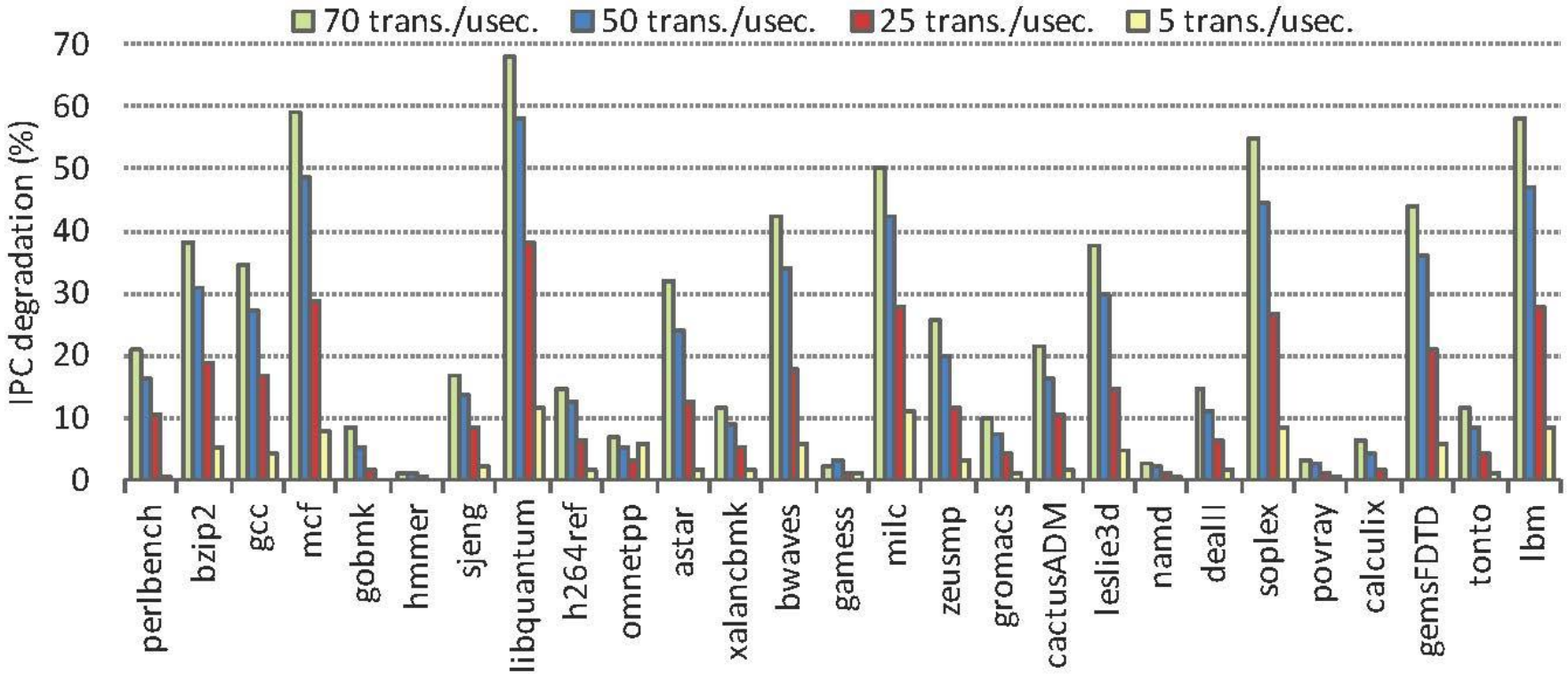
Experiment:

- Benchmark + 3 microbenchmarks with different TR_{MM}



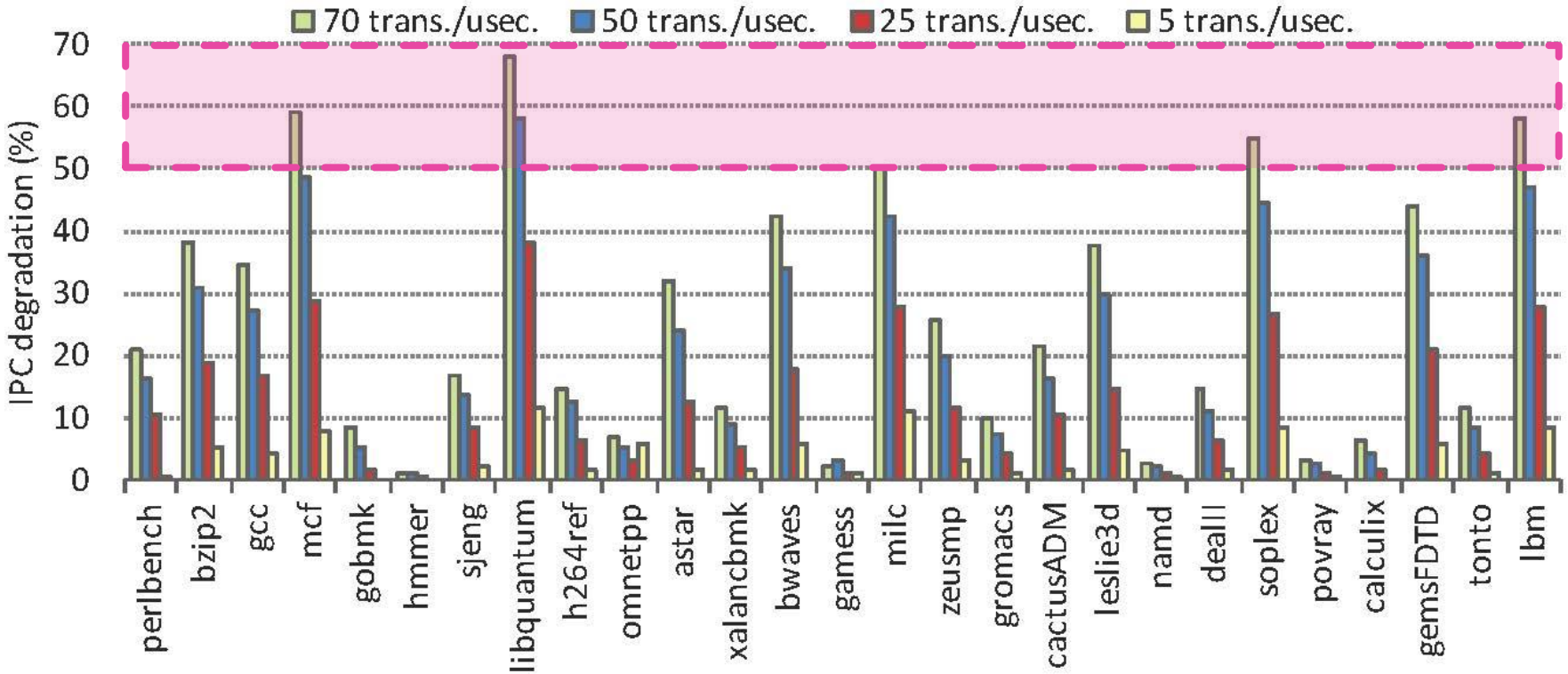
Performance degradation analysis

Main memory bandwidth contention



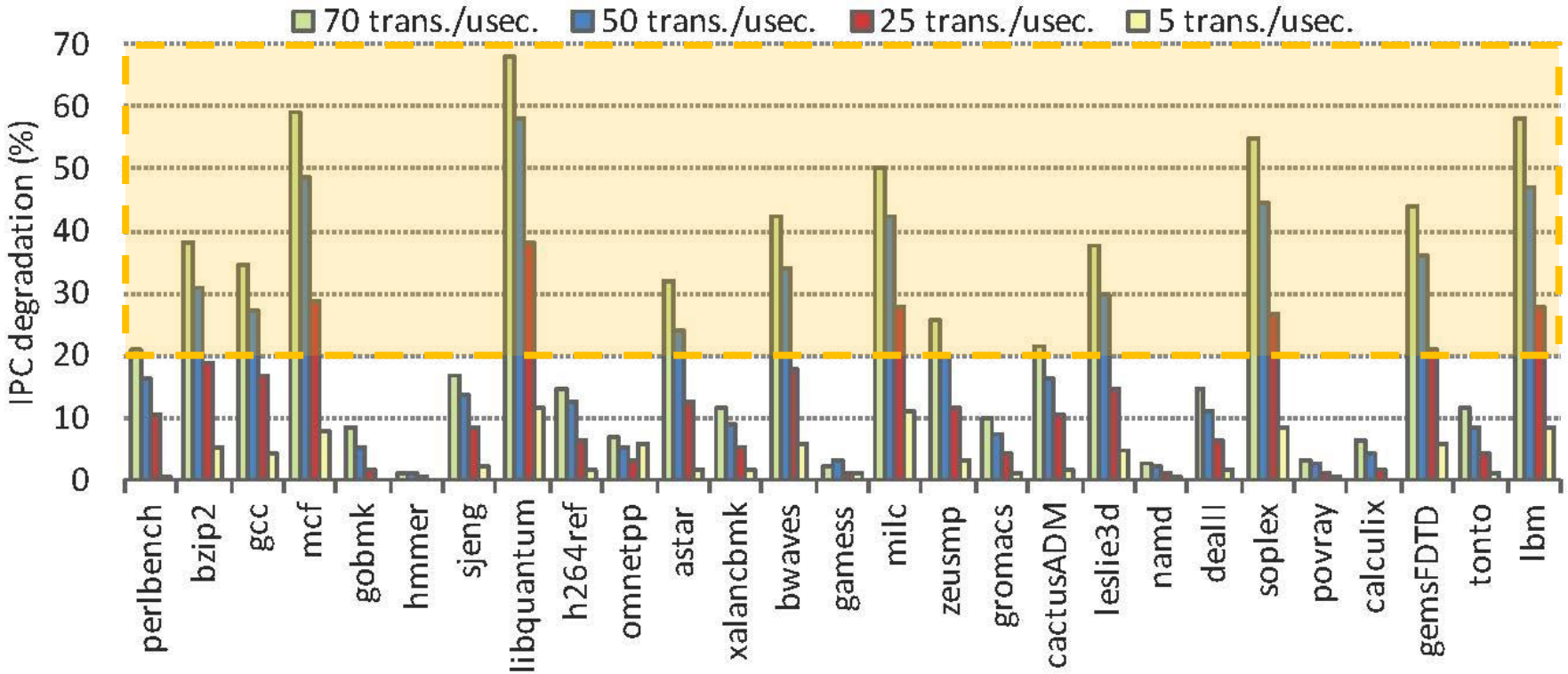
Performance degradation analysis

Main memory bandwidth contention



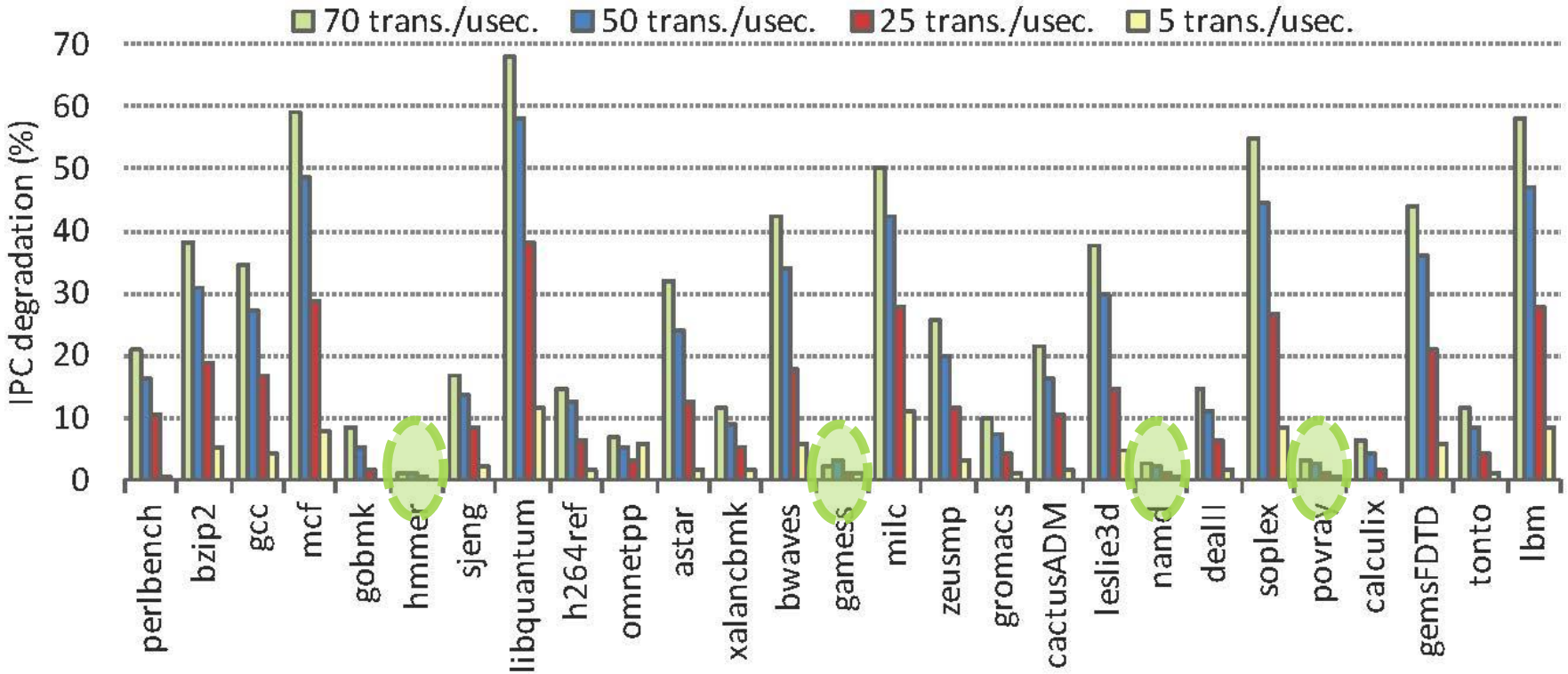
Performance degradation analysis

Main memory bandwidth contention



Performance degradation analysis

Main memory bandwidth contention

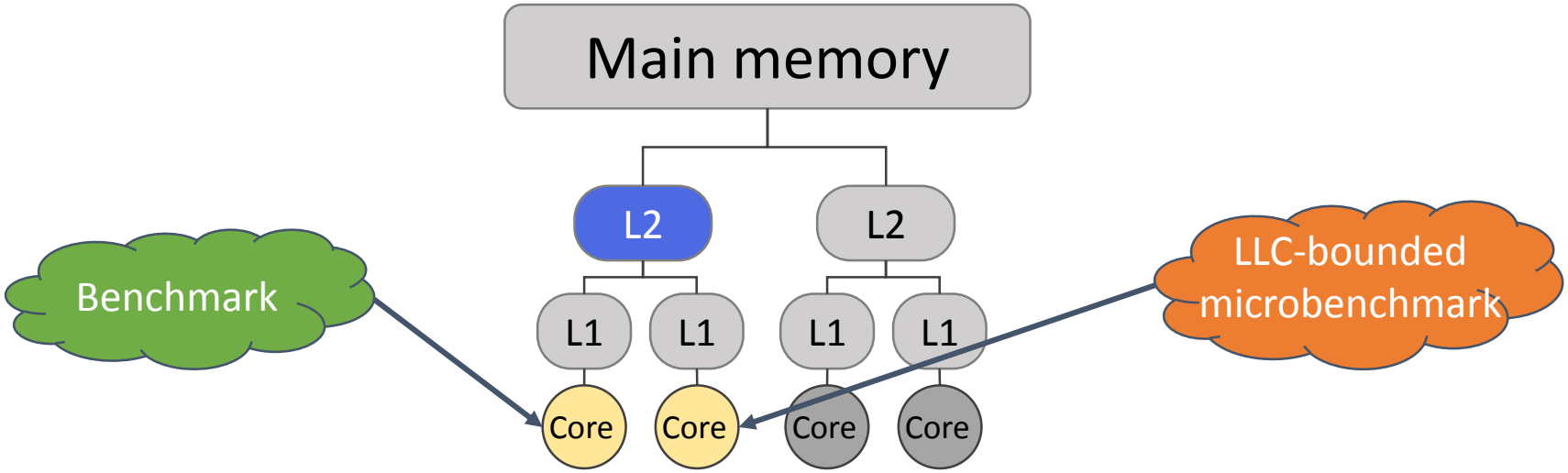


Performance degradation analysis

L2 bandwidth contention

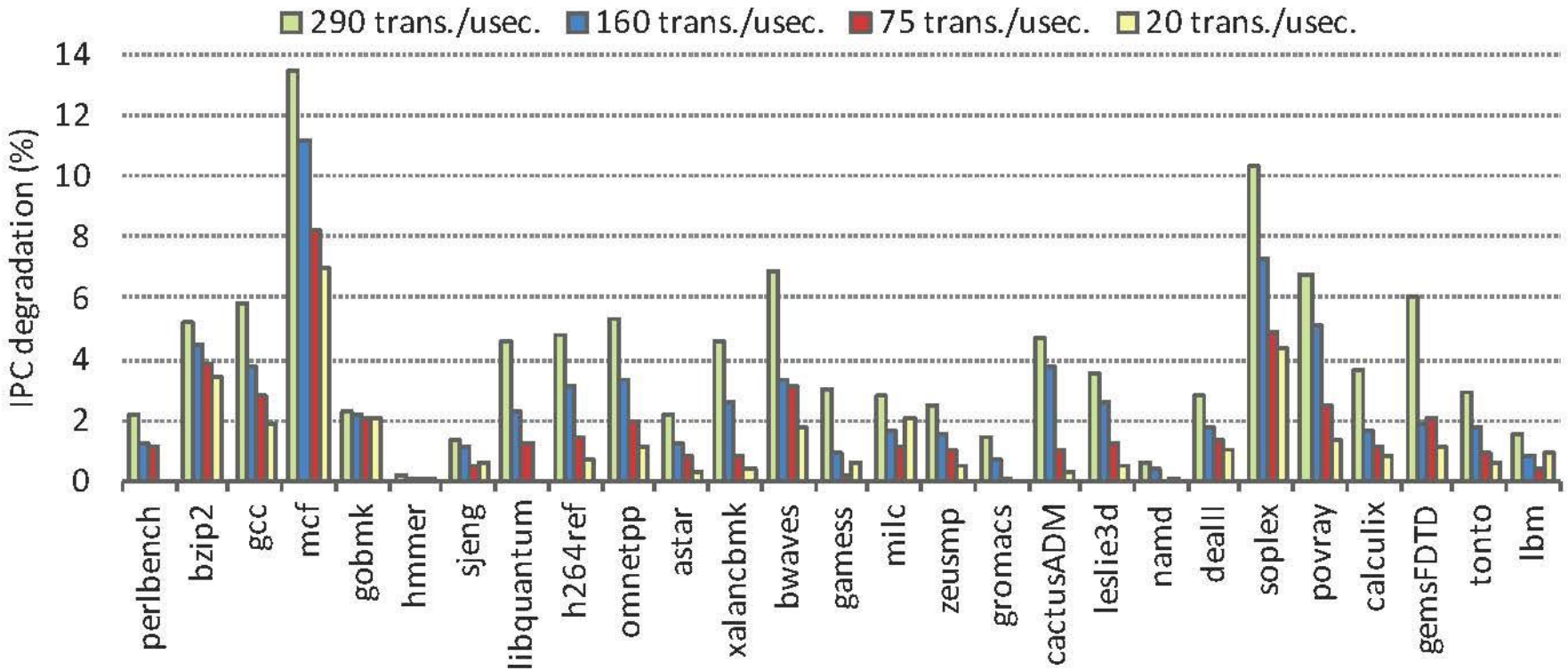
Experiment:

- Benchmark + 1 microbenchmarks with different TR_{L2}



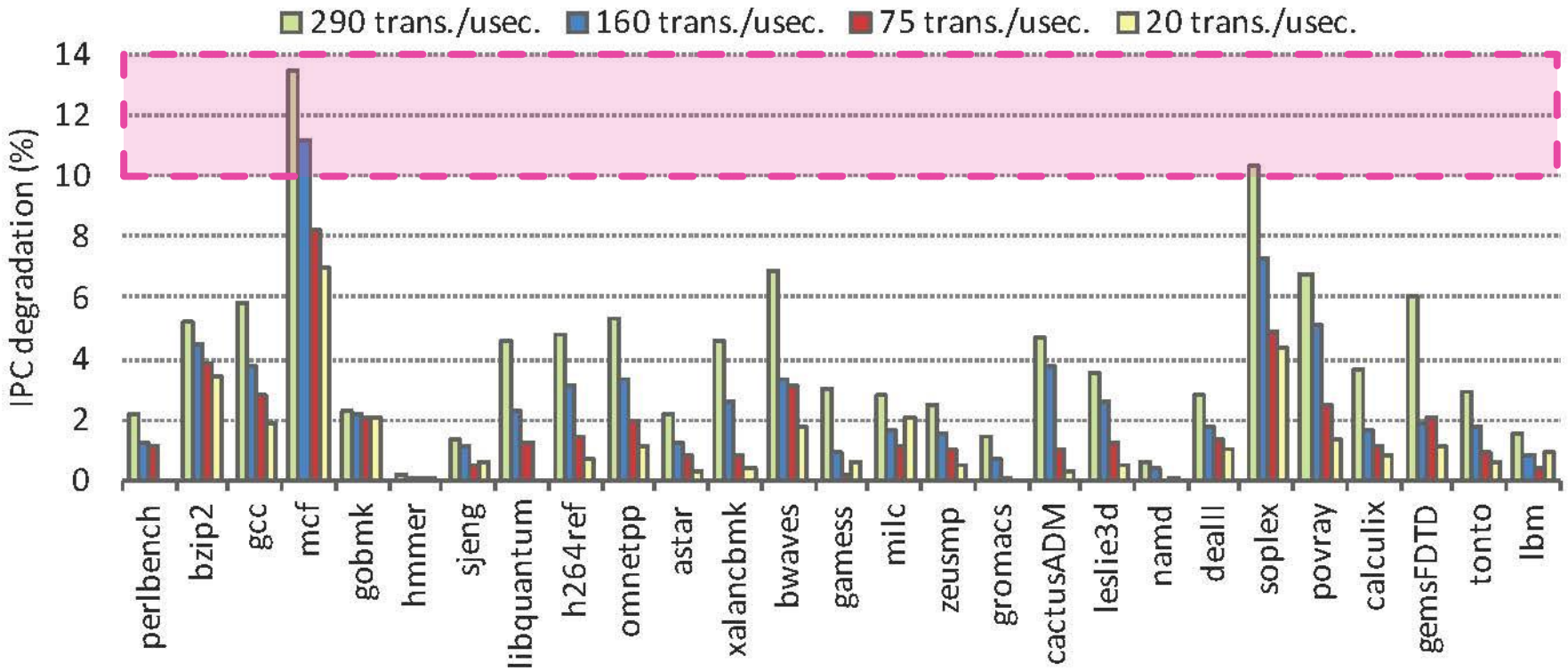
Performance degradation analysis

L2 bandwidth contention



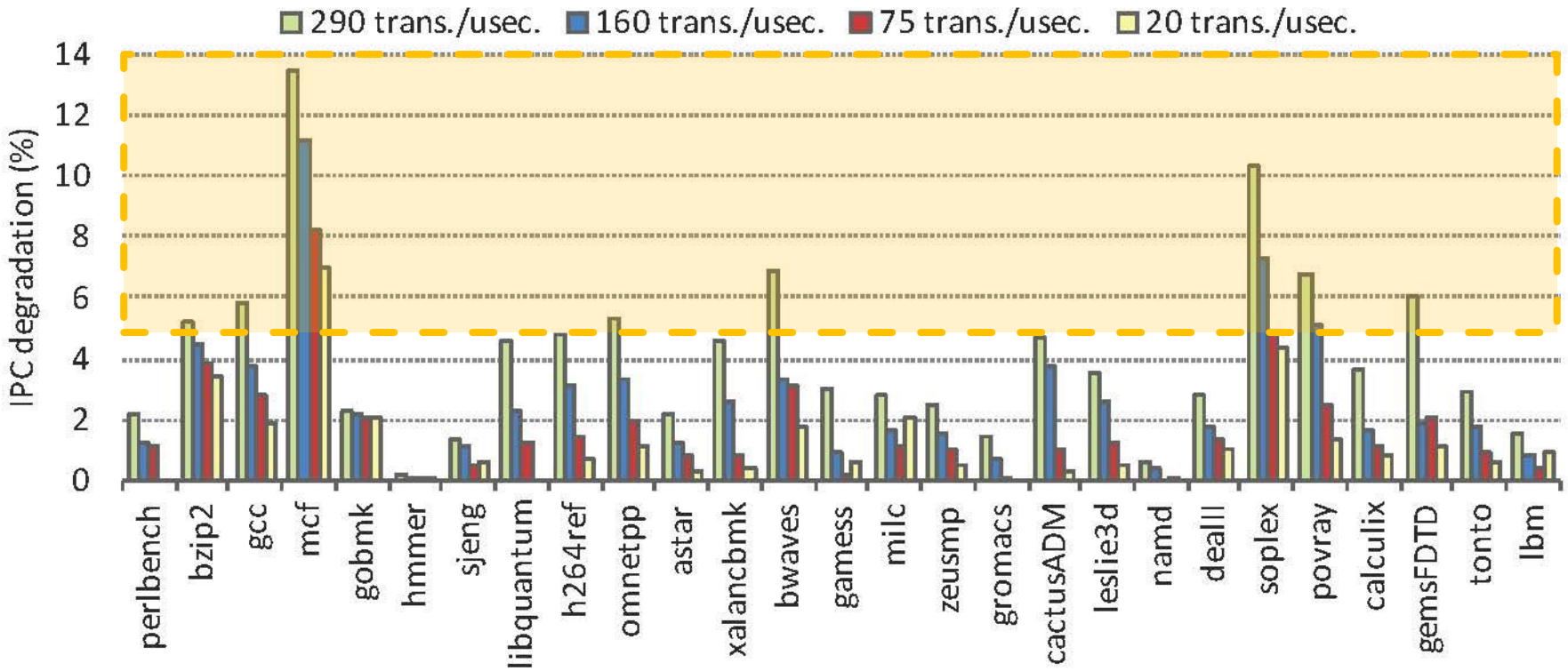
Performance degradation analysis

L2 bandwidth contention



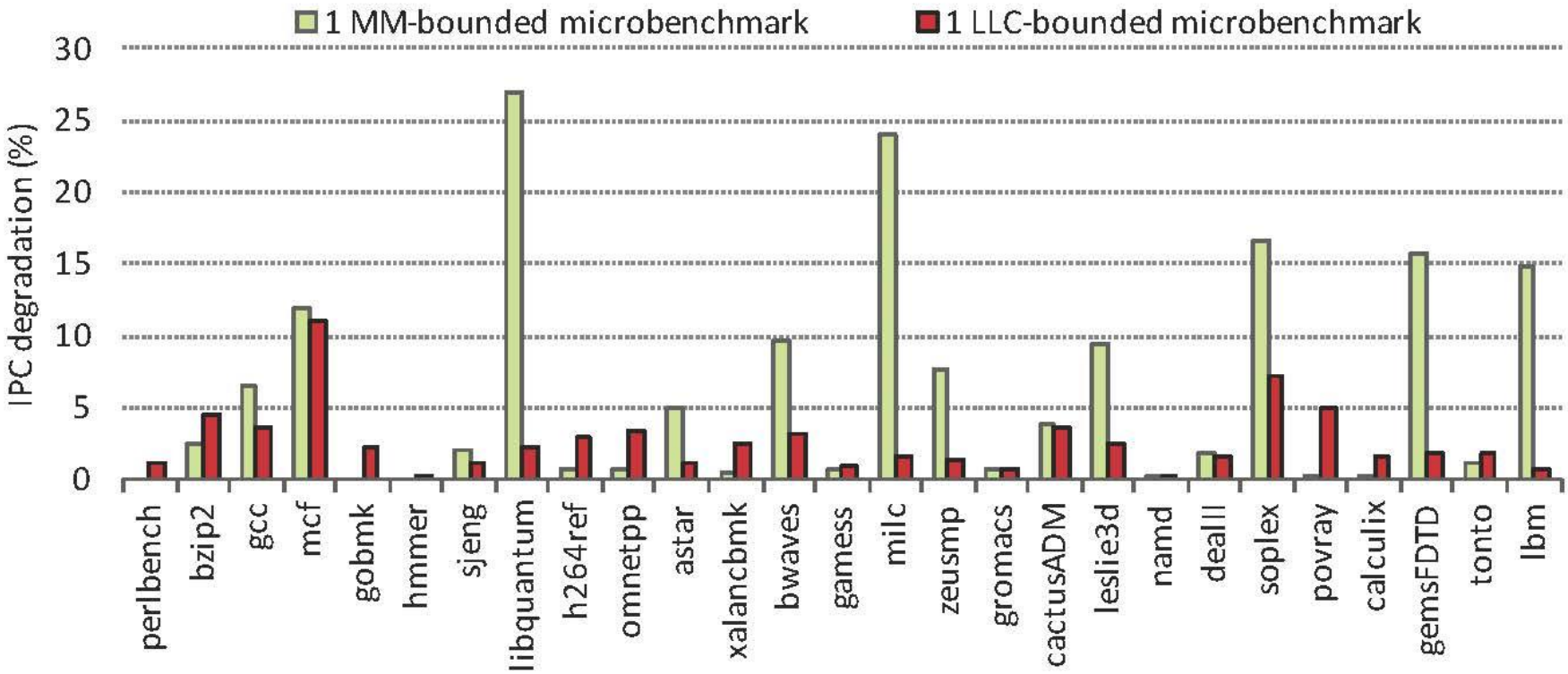
Performance degradation analysis

L2 bandwidth contention



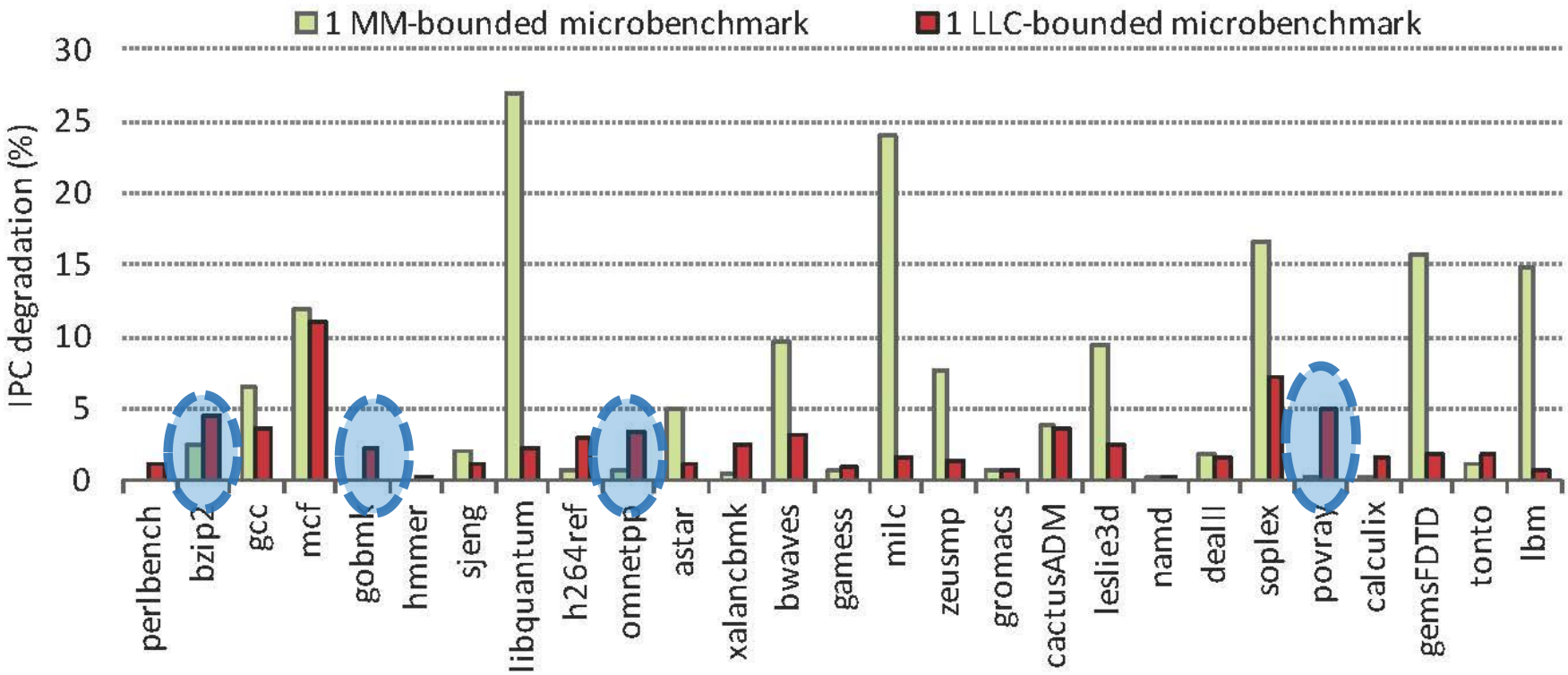
Performance degradation analysis

L2 bandwidth contention



Performance degradation analysis

L2 bandwidth contention



Higher performance degradation due to LLC bandwidth contention

Memory-hierarchy bandwidth-aware scheduler

Baseline

Main memory bandwidth-aware scheduler (state-of-the-art)

D. Xu, C. Wu, and P.-C. Yew, “On mitigating memory bandwidth contention through bandwidth-aware scheduling”, at *PACT 2010*

Memory-hierarchy bandwidth-aware scheduler

Baseline

Main memory bandwidth-aware scheduler (state-of-the-art)

D. Xu, C. Wu, and P.-C. Yew, “On mitigating memory bandwidth contention through bandwidth-aware scheduling”, at *PACT 2010*

- Distributes the memory requests over the workload execution time to minimize bandwidth contention

Memory-hierarchy bandwidth-aware scheduler

Baseline

Main memory bandwidth-aware scheduler (state-of-the-art)

D. Xu, C. Wu, and P.-C. Yew, "On mitigating memory bandwidth contention through bandwidth-aware scheduling", at *PACT 2010*

- Distributes the memory requests over the workload execution time to minimize bandwidth contention

$$IABW = \frac{\sum_{p=0}^{P} AvgTR_{MM}^{p} * T^{p}}{\frac{\sum_{p=0}^{P} T^{p}}{\#\ cores}}$$

Overall memory transactions of process *p*

Ideal execution time of the workload (without interference)

Ideal Average Bandwidth

Memory-hierarchy bandwidth-aware scheduler

Baseline

Main memory bandwidth-aware scheduler (state-of-the-art)

D. Xu, C. Wu, and P.-C. Yew, "On mitigating memory bandwidth contention through bandwidth-aware scheduling", at *PACT 2010*

- Distributes the memory requests over the workload execution time to minimize bandwidth contention

$$IABW = \frac{\sum_{p=0}^{P} AvgTR_{MM}^{p} * T^{p}}{\frac{\sum_{p=0}^{P} T^{p}}{\#\ cores}}$$

Ideal Average Bandwidth

Overall memory transactions of process *p*

Requires preliminary information of the processes

Ideal execution time of the workload (without interference)

Memory-hierarchy bandwidth-aware scheduler

Baseline

Main memory bandwidth-aware scheduler (state-of-the-art)

D. Xu, C. Wu, and P.-C. Yew, "On mitigating memory bandwidth contention through bandwidth-aware scheduling", at *PACT 2010*

- Distributes the memory requests over the workload execution time to minimize bandwidth contention

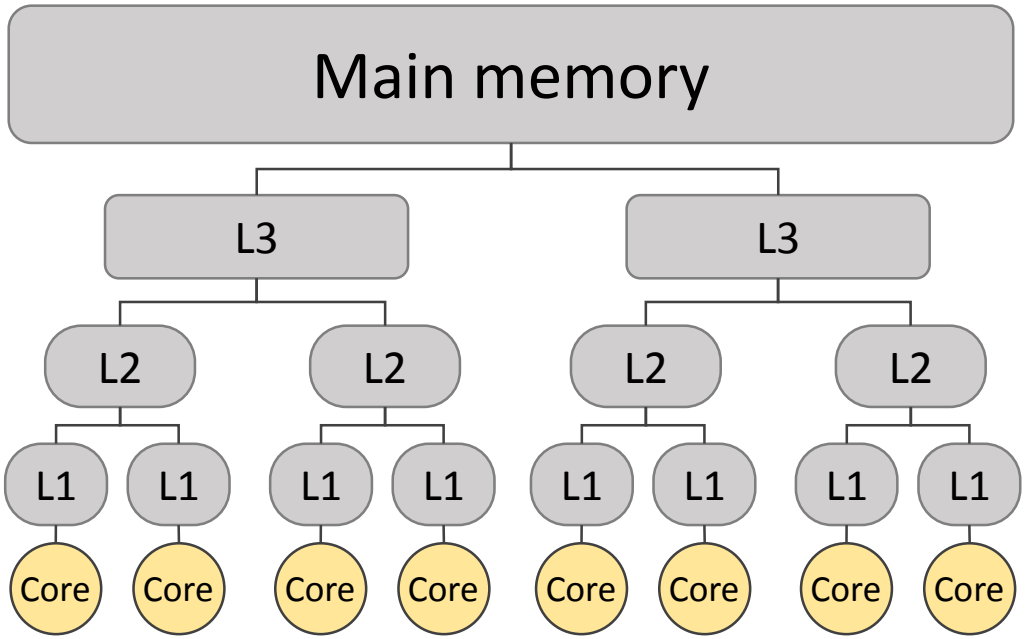
$$IABW = \frac{\sum_{p=0}^{P} AvgTR_{MM}^{p} * T^{p}}{\frac{\sum_{p=0}^{P} T^{p}}{\#\ cores}}$$

BW_{Remain} =IABW
Select the process P_head at the queue head
BW_{Remain} − =$TR_{MM}^{P\_head}$, CPU_{Remain} = #cores−1
while $\mathrm{CPU}_{Remain} > 0$ **do**
  select the process P that maximizes
  $\mathrm{FITNESS(p)} = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - TR_{MM}^{P}\right|}$
  BW_{Remain} − = TR_{MM}^{P}, CPU_{Remain} − −
end while

Quantifies the gap between the predicted TR_{MM} and the available TR_{MM} per still unallocated core

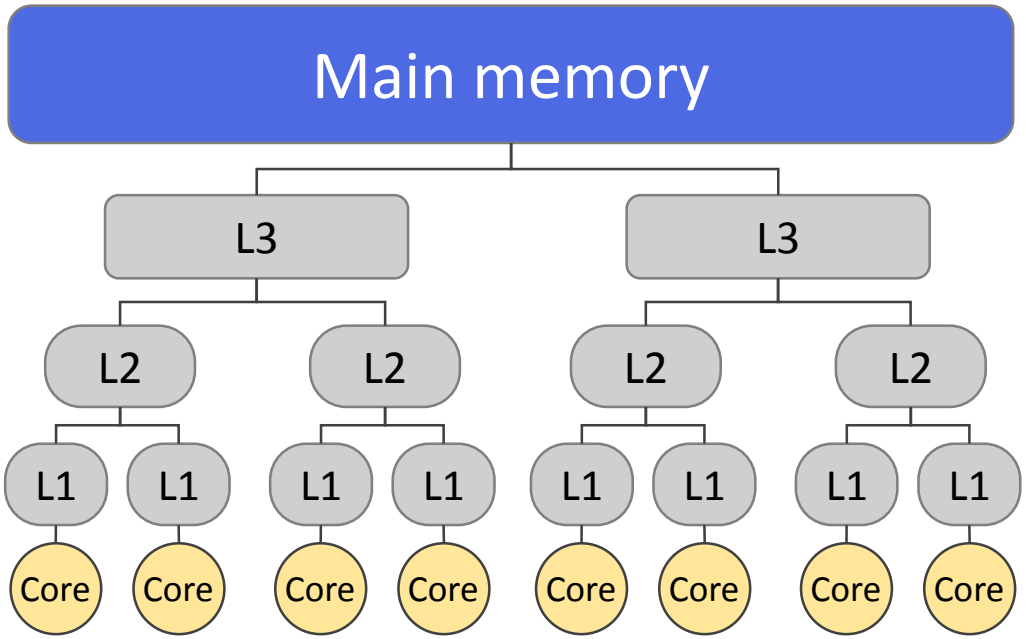
Memory-hierarchy bandwidth-aware scheduler

- Addresses bandwidth contention on the entire memory hierarchy



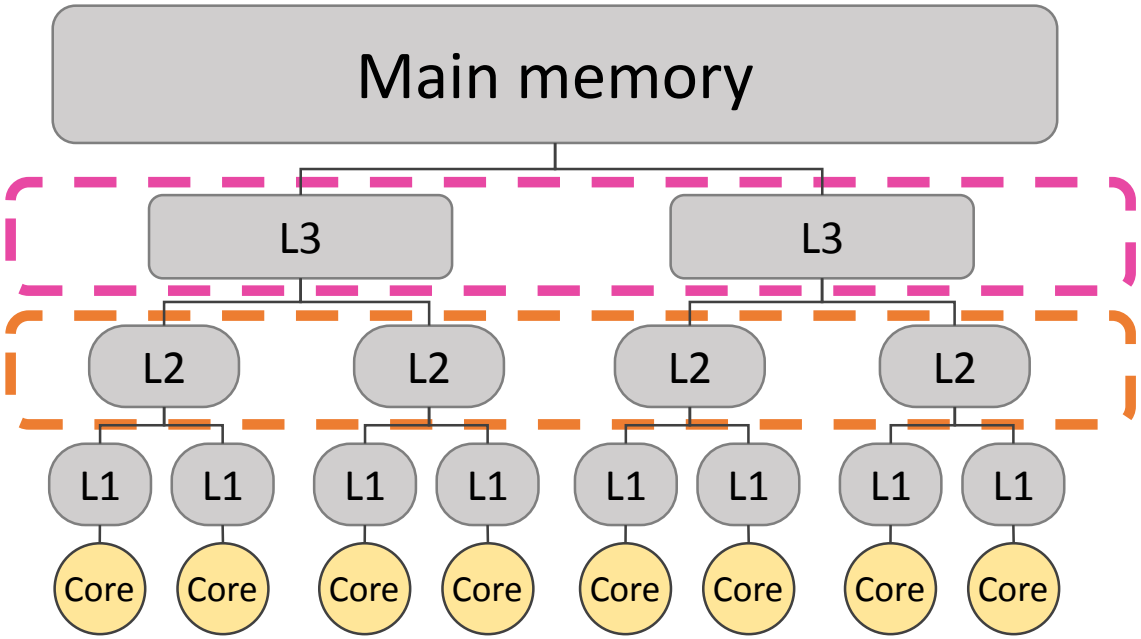
Memory-hierarchy bandwidth-aware scheduler

- Addresses bandwidth contention on the entire memory hierarchy
  i. Process selection to approach the IABW



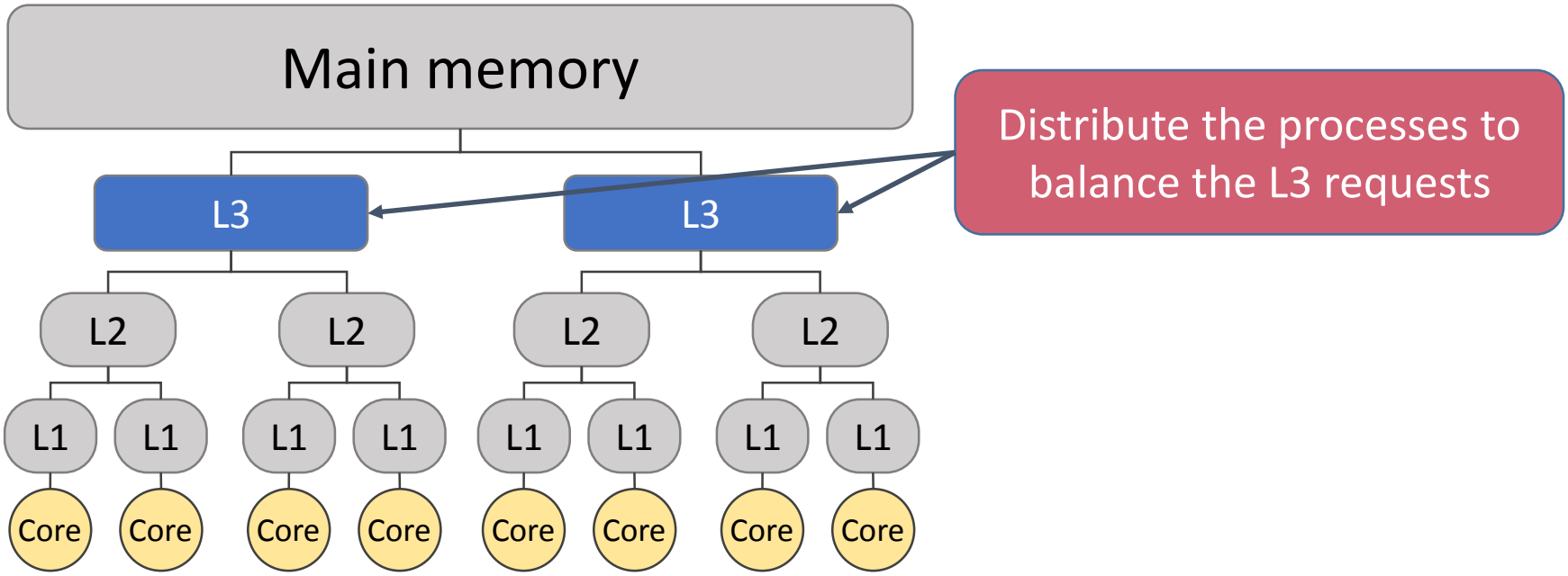
Memory-hierarchy bandwidth-aware scheduler

- Addresses bandwidth contention on the entire memory hierarchy
  i. Process selection to approach the IABW
  ii. Process allocation to balance the cache requests over the caches in that level
    - Minimize contention
    - *n* steps (*n* = levels with multiple shared caches)



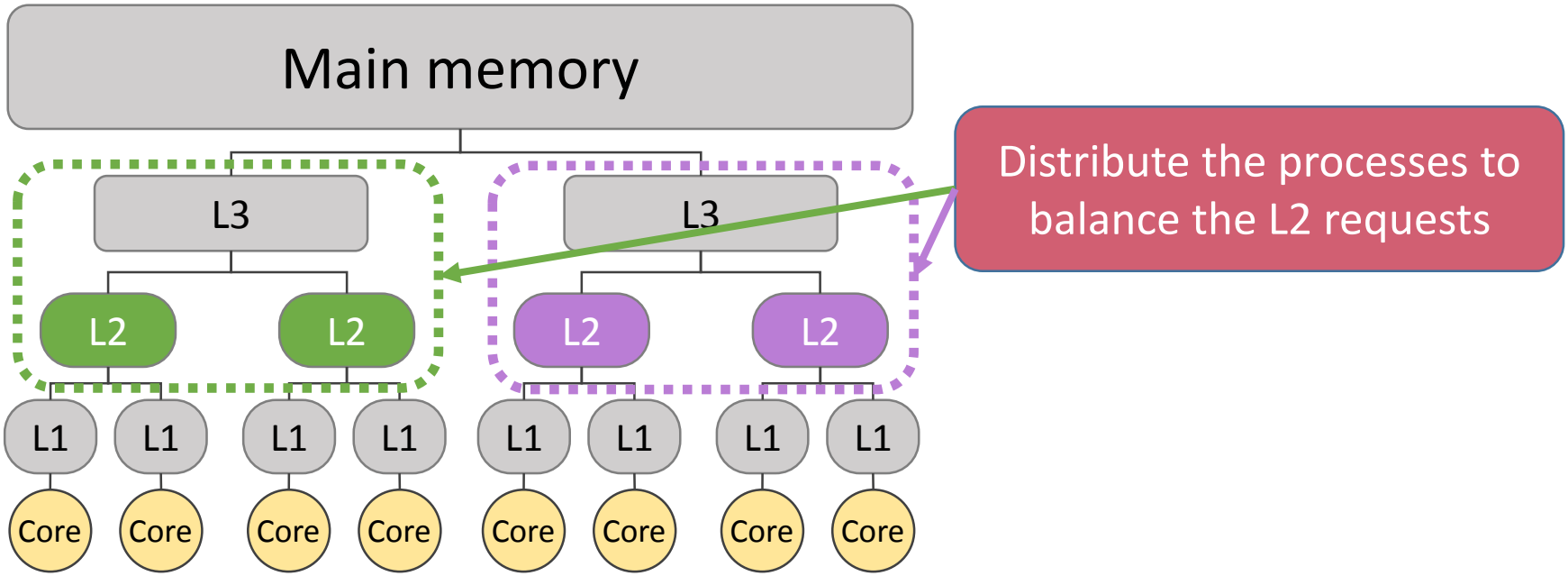
Memory-hierarchy bandwidth-aware scheduler

- Addresses bandwidth contention on the entire memory hierarchy
  i. Process selection to approach the IABW
  ii. Process allocation to balance the cache requests over the caches in that level
    - Minimize contention
    - *n* steps (*n* = levels with multiple shared caches)



Memory-hierarchy bandwidth-aware scheduler

- Addresses bandwidth contention on the entire memory hierarchy
  - i. Process selection to approach the IABW
  - ii. Process allocation to balance the cache requests over the caches in that level
    - Minimize contention
    - *n* steps (*n* = levels with multiple shared caches)



Memory-hierarchy bandwidth-aware scheduler

Algorithm 2 Memory-hierarchy bandwidth-aware scheduler

Require: Benchmarks submitted with execution time and TR_{MM} in stand-alone execution

1: $IABW = \frac{\sum_{p=0}^{P} (TR_{MM}^{p}) * T^{p}}{\frac{\sum_{p=0}^{P} T^{p}}{\#cores}}$
2: **while** there are unfinished jobs **do**
3: Block the executing processes and place them at the queue tail
4: **for** each process P executed in the last quantum **do**
5: **for** each cache level L **do**
6: Update TR for process P in cache level L
7: **end for**
8: **end for**
9: BW_{Remain} =IABW
10: Select the process P_head at the queue head
11: $BW_{Remain} - = TR_{MM}^{P\_head}$, CPU_{Remain} = #cores−1
12: **while** $CPU_{Remain} > 0$ **do**
13: select the process P that maximizes
14: FITNESS(p) = $\frac{1}{\left| \frac{BW_{Remain}}{CPU_{Remain}} - TR_{MM}^{P} \right|}$
15: $BW_{Remain} - = TR_{MM}^{P}$, $CPU_{Remain} - -$
16: **end while**
17: **for** each level i in the cache-hierarchy with shared caches beginning from the LLC **do**
18: AVG_TR(L_i)=$\frac{\sum TR_{L(i)}}{\#Caches \ \ at \ \ Li}$
19: **for** each cache in level L_i **do**
20: BW_{Remain} = AVG_TR(L_i), CPU_{Remain} = # cores sharing the cache
21: **while** $CPU_{Remain} > 0$ **do**
22: From the remaining processes selected to share the immediately lower memory level, select the process P that maximizes
23: FITNESS(p) = $\frac{1}{\left| \frac{BW_{Remain}}{CPU_{Remain}} - TR_{Li}^{P} \right|}$
24: $BW_{Remain} - = TR_{Li}^{P}$, $CPU_{Remain} - -$
25: **end while**
26: **end for**
27: **end for**
28: Unblock the processes, and allocate them in the chosen core
29: Sleep during the quantum
30: **end while**

Process selection:
main memory bandwidth-aware

Memory-hierarchy bandwidth-aware scheduler

Algorithm 2 Memory-hierarchy bandwidth-aware scheduler

Require: Benchmarks submitted with execution time and TR_{MM} in stand-alone execution

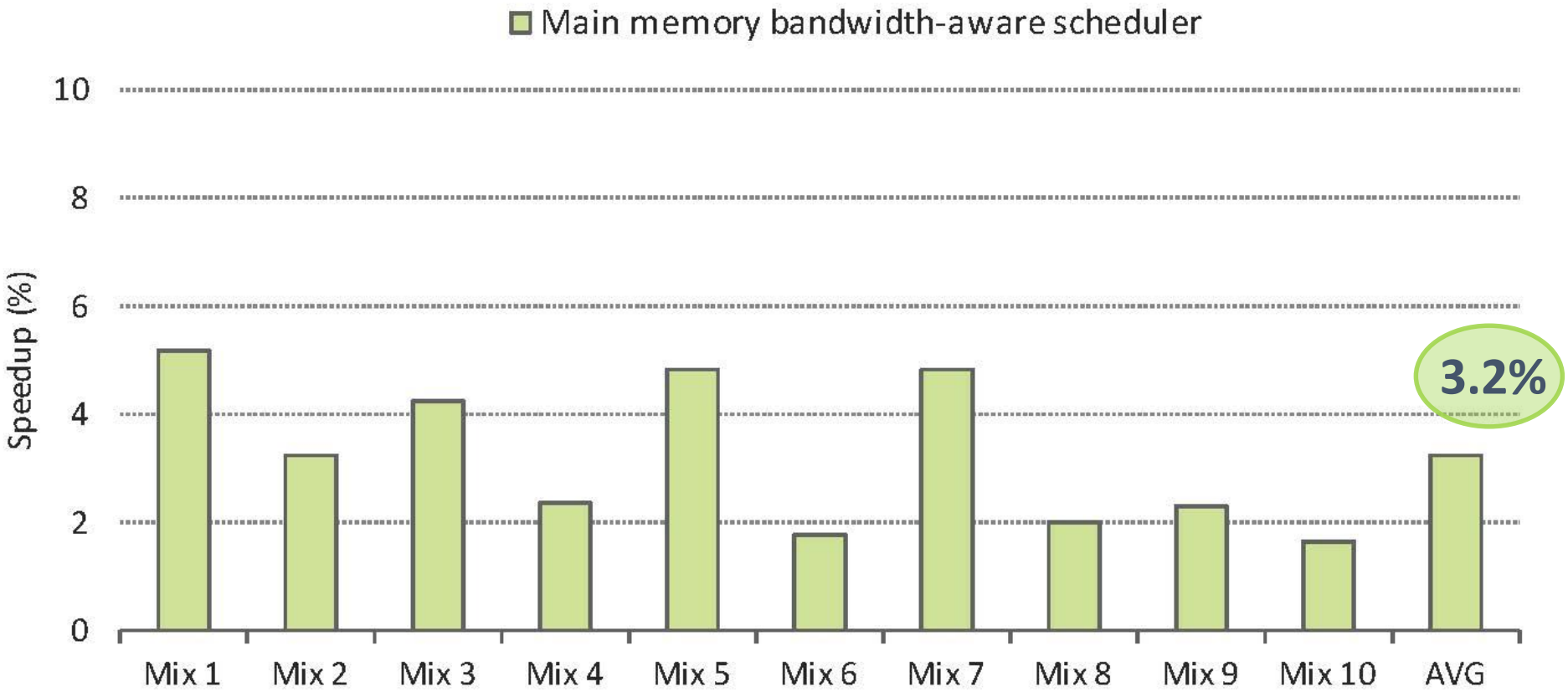
1: $IABW = \frac{\sum_{p=0}^{P} (TR_{MM}^{p}) * T^{p}}{\frac{\sum_{p=0}^{P} T^{p}}{\#cores}}$
2: **while** there are unfinished jobs **do**
3: Block the executing processes and place them at the queue tail
4: for each process P executed in the last quantum **do**
5: for each cache level L **do**
6: Update TR for process P in cache level L
7: **end for**
8: **end for**
9: BW_{Remain} =IABW
10: Select the process P_head at the queue head
11: $BW_{Remain} - = TR_{MM}^{P\_head}$, CPU_{Remain} = #cores−1
12: **while** $CPU_{Remain} > 0$ **do**
13: select the process P that maximizes
14: FITNESS(p) = $\frac{1}{\left| \frac{BW_{Remain}}{CPU_{Remain}} - TR_{MM}^{P} \right|}$
15: $BW_{Remain} - = TR_{MM}^{P}$, $CPU_{Remain} - -$
16: **end while**
17: for each level i in the cache-hierarchy with shared caches beginning from the LLC **do**
18: AVG_TR(L_i)=$\frac{\sum TR_{L(i)}}{\#Caches\ \ at\ \ Li}$
19: for each cache in level L_i **do**
20: BW_{Remain} = AVG_TR(L_i), CPU_{Remain} = # cores sharing the cache
21: **while** $CPU_{Remain} > 0$ **do**
22: From the remaining processes selected to share the immediately lower memory level, select the process P that maximizes
23: FITNESS(p) = $\frac{1}{\left| \frac{BW_{Remain}}{CPU_{Remain}} - TR_{Li}^{P} \right|}$
24: $BW_{Remain} - = TR_{Li}^{P}$, $CPU_{Remain} - -$
25: **end while**
26: **end for**
27: **end for**
28: Unblock the processes, and allocate them in the chosen core
29: Sleep during the quantum
30: **end while**

Process selection:
main memory bandwidth-aware

Process allocation:
cache bandwidth-aware

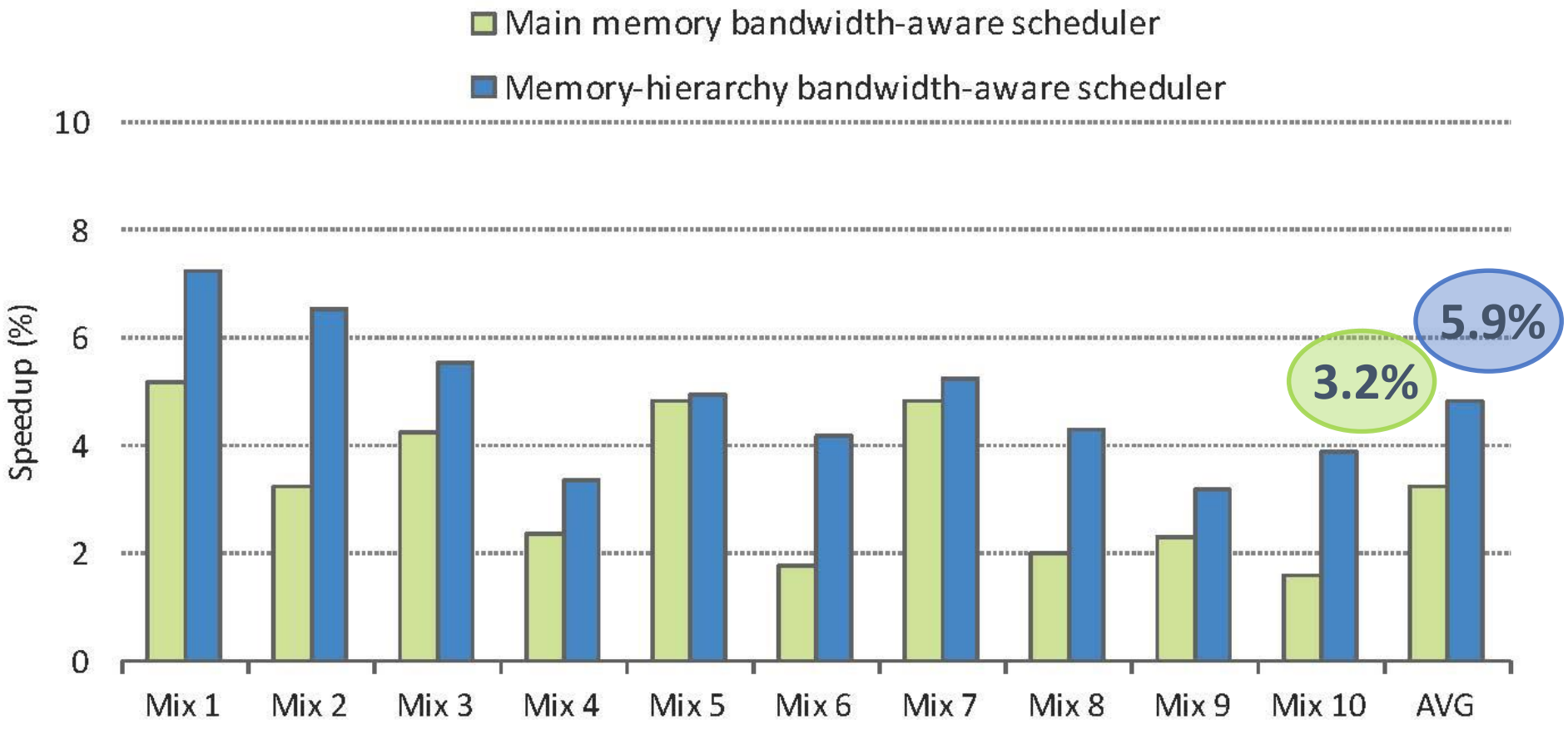
Experimental evaluation

Turnaround time speedup over Linux (I)



Experimental evaluation

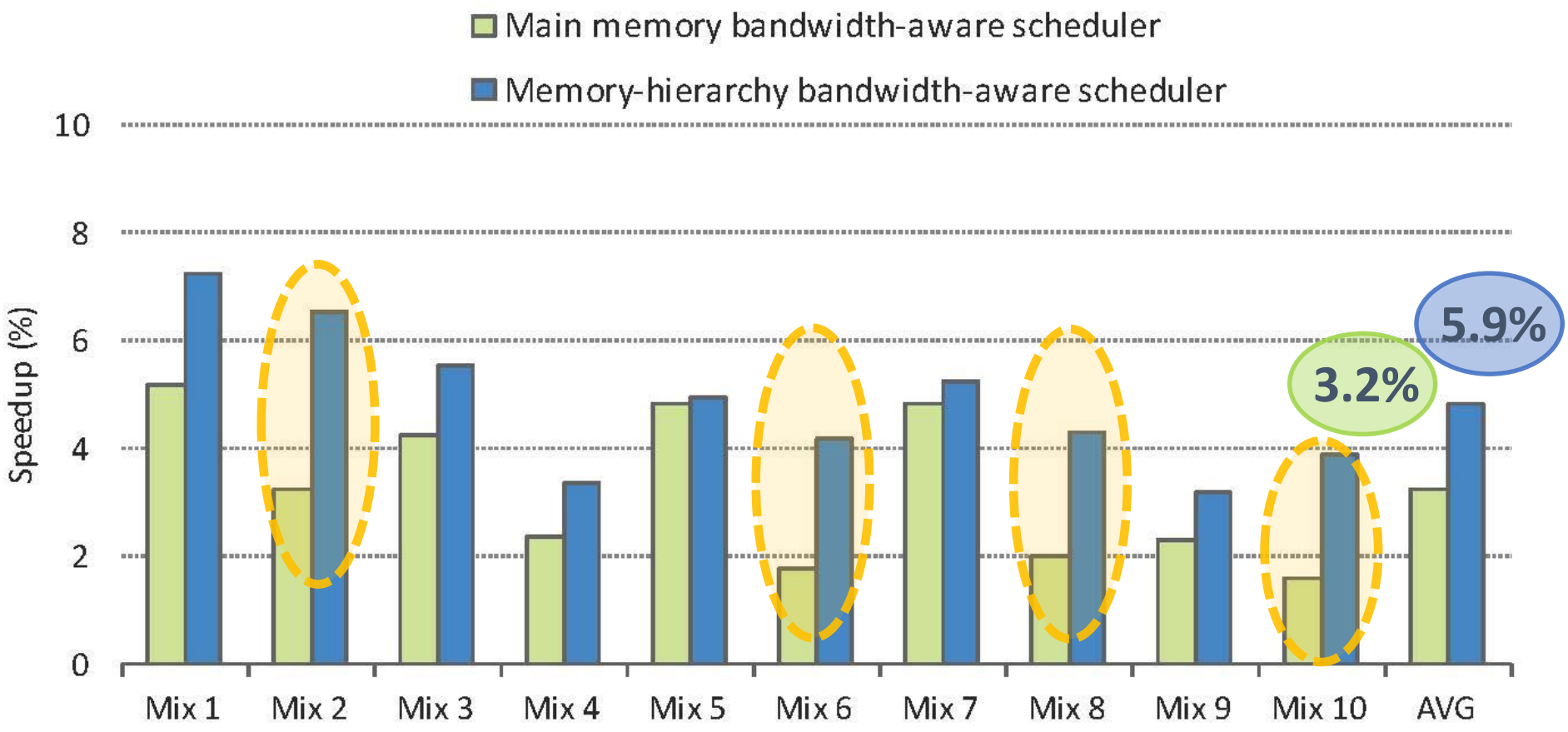
Turnaround time speedup over Linux (I)



Experimental evaluation

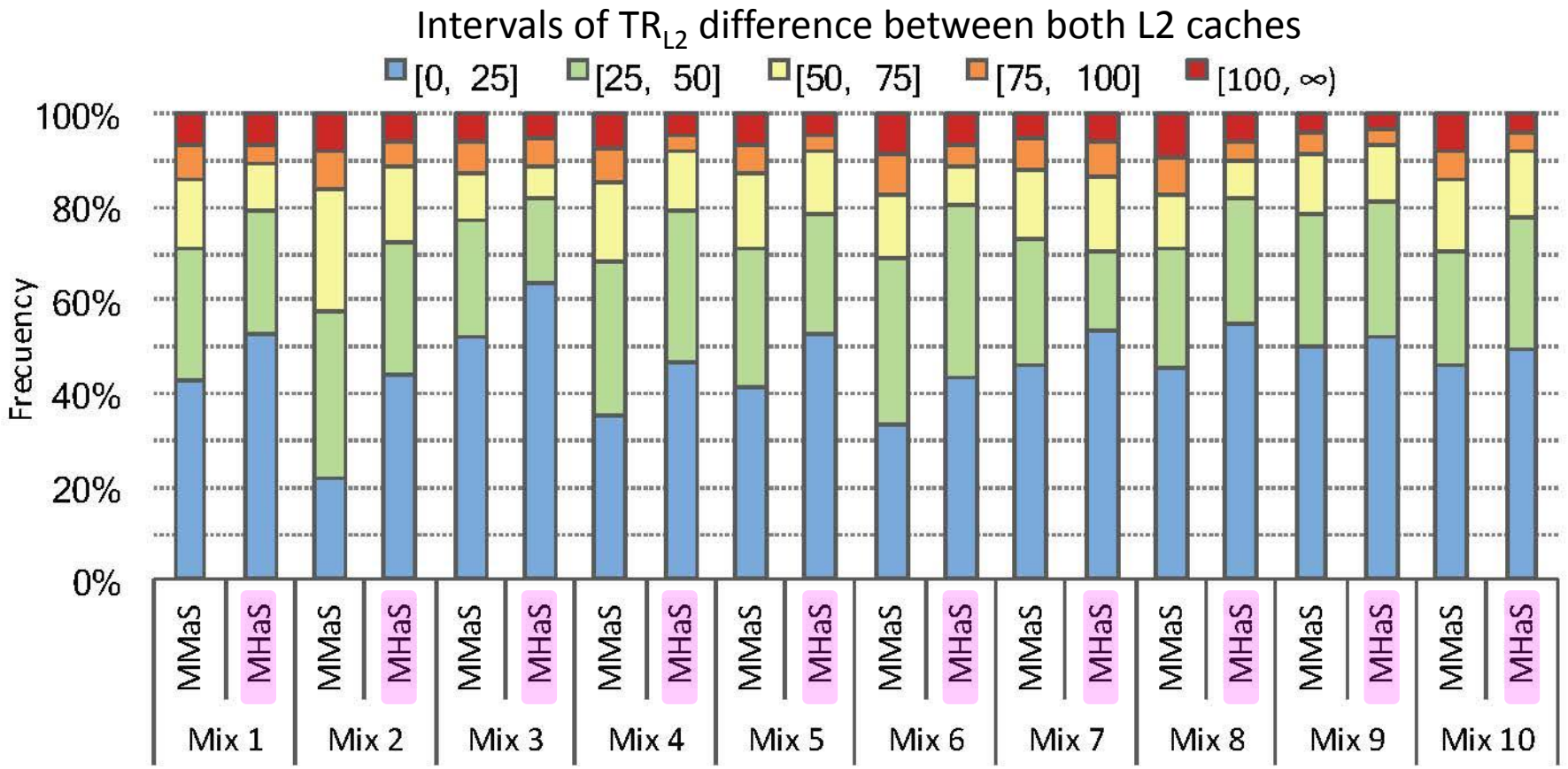
Turnaround time speedup over Linux (I)

The proposal doubles the speedup of the state-of-the-art in some workloads



Experimental evaluation

TR_{L2} difference (I) – Histogram



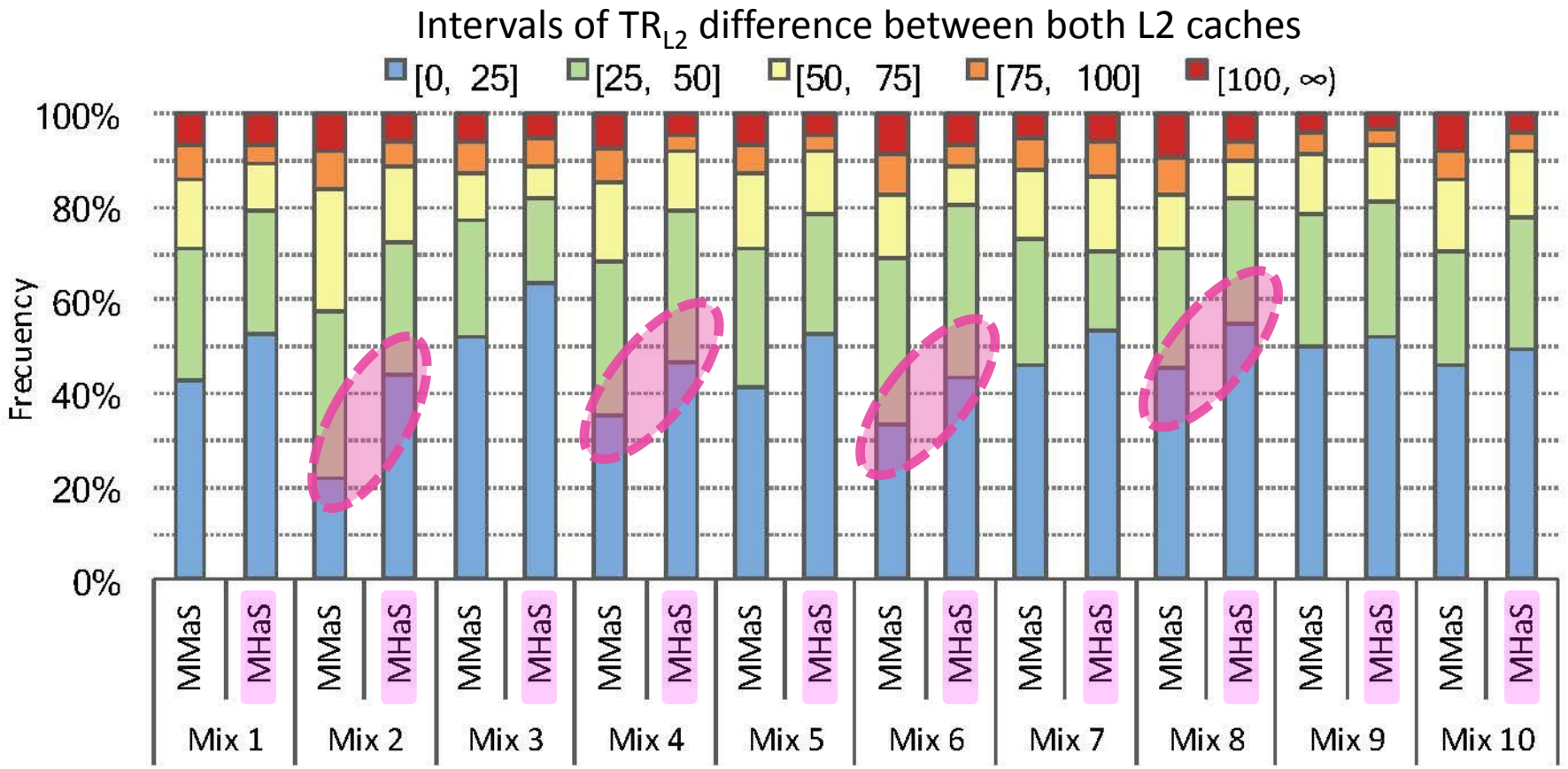
MMaS: main memory bandwidth-aware scheduler

MHaS: memory hierarchy bandwidth-aware scheduler

The higher the frequency of the lower intervals, the better the L2 requests are balanced

Experimental evaluation

TR_{L2} difference (I) – Histogram



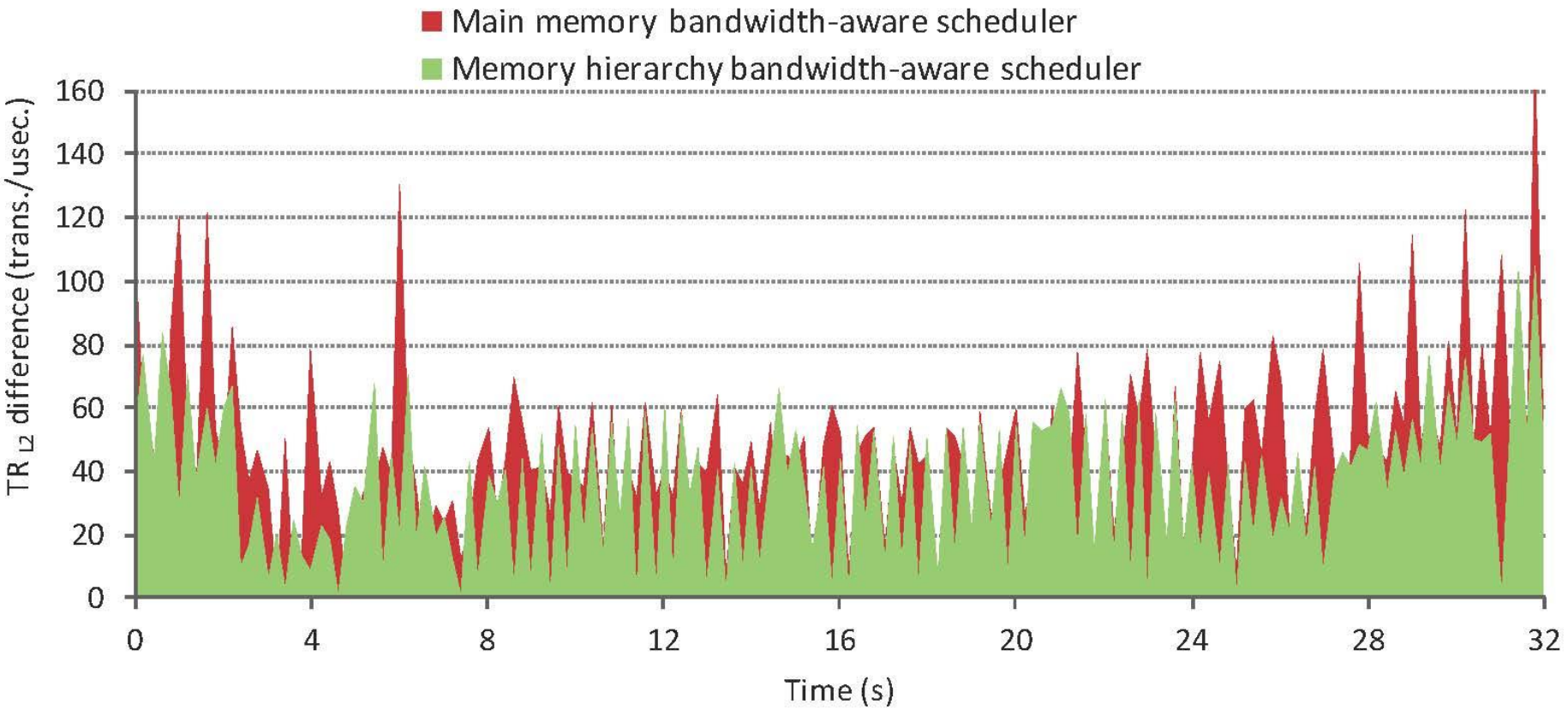
MMaS: main memory bandwidth-aware scheduler

MHaS: memory hierarchy bandwidth-aware scheduler

The higher the frequency of the lower intervals, the better the L2 requests are balanced

Experimental evaluation

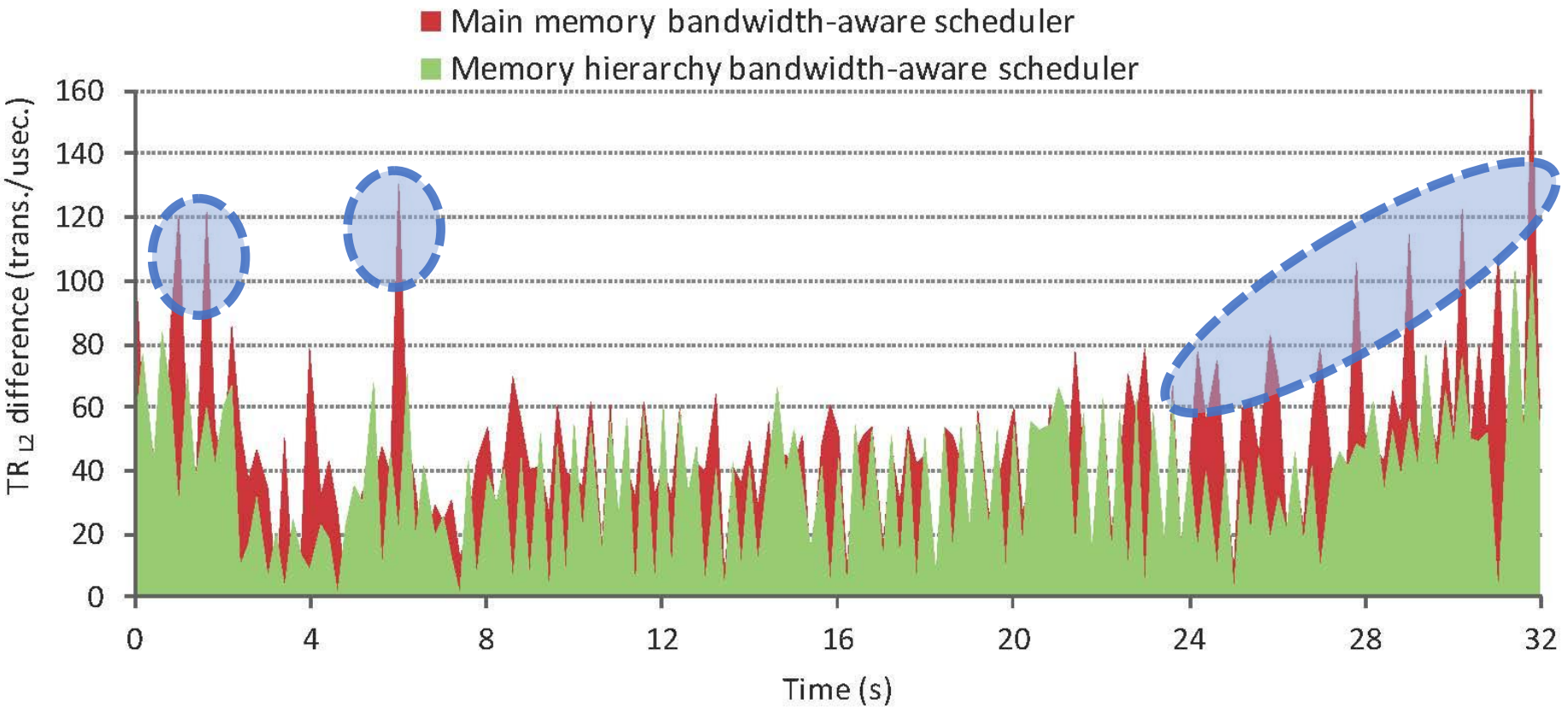
TR_{L2} difference (II) – Dynamic evolution



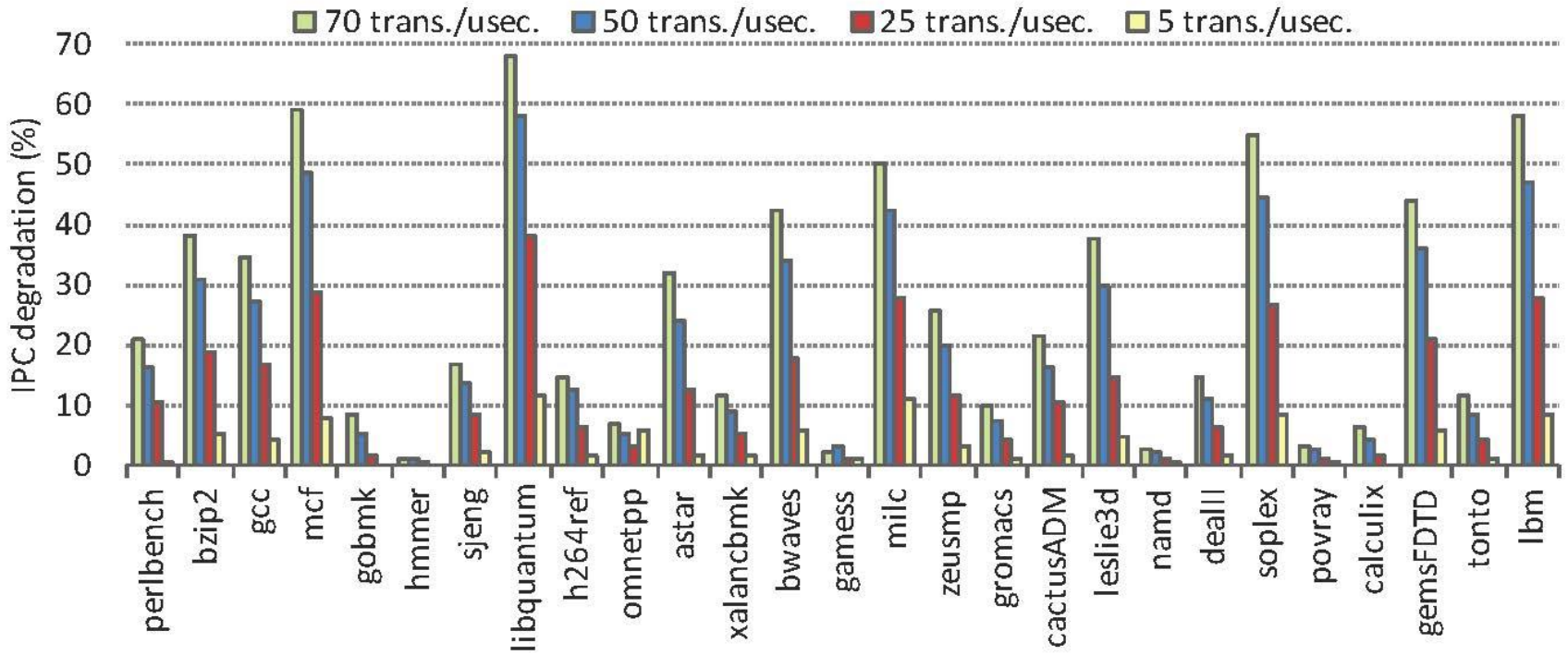
Experimental evaluation

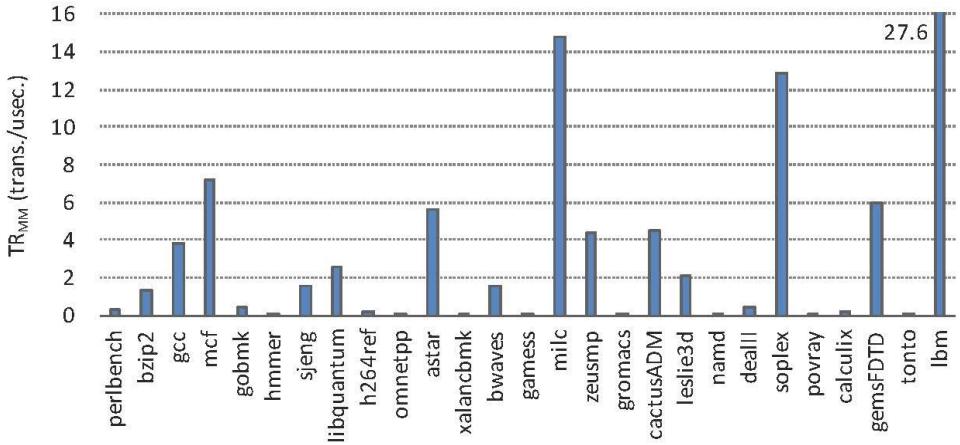
TR_{L2} difference (II) – Dynamic evolution

Peaks of TR_{L2} difference reduced in frequency and height



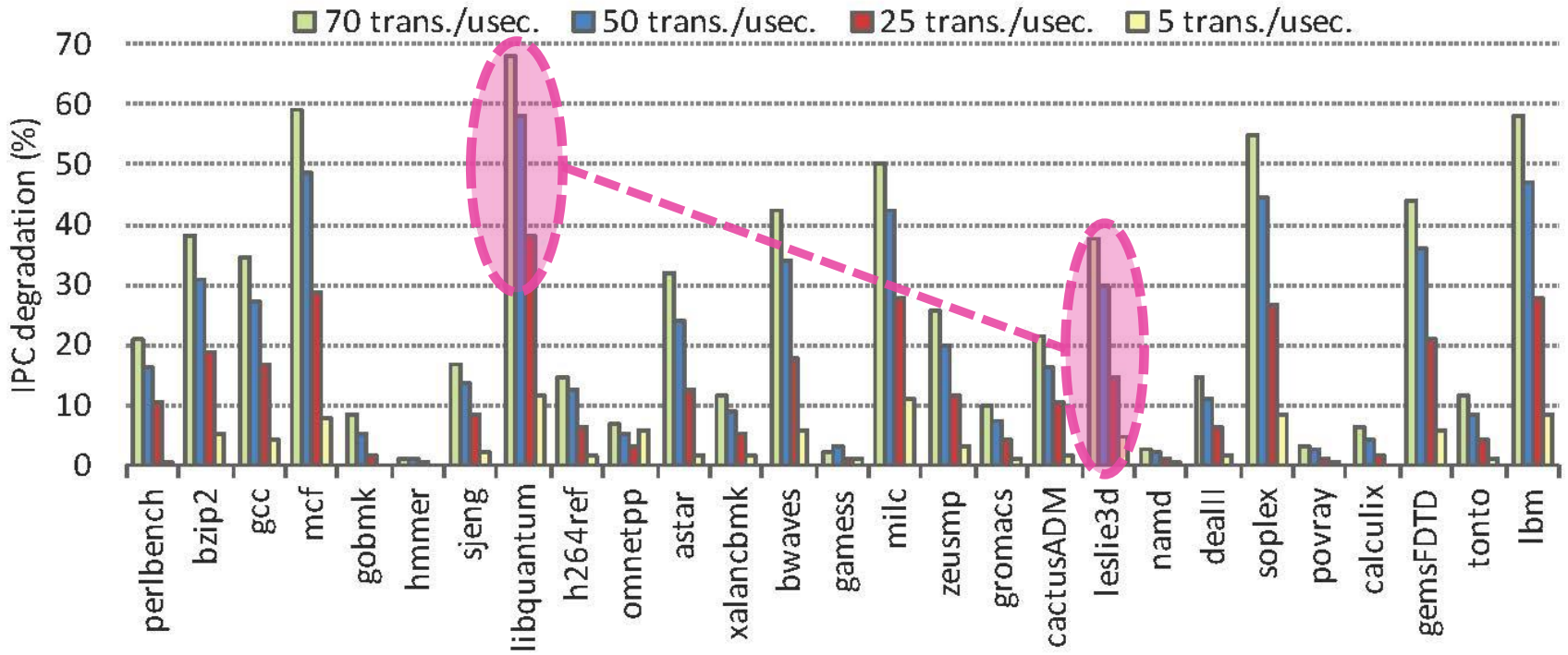
IPC-degradation memory-hierarchy bandwidth-aware scheduler

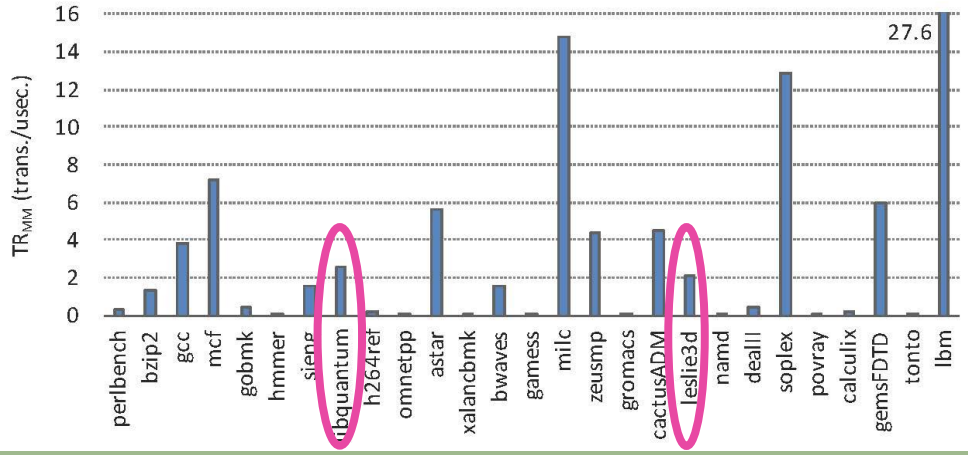




Applications with similar bandwidth suffer different performance degradation

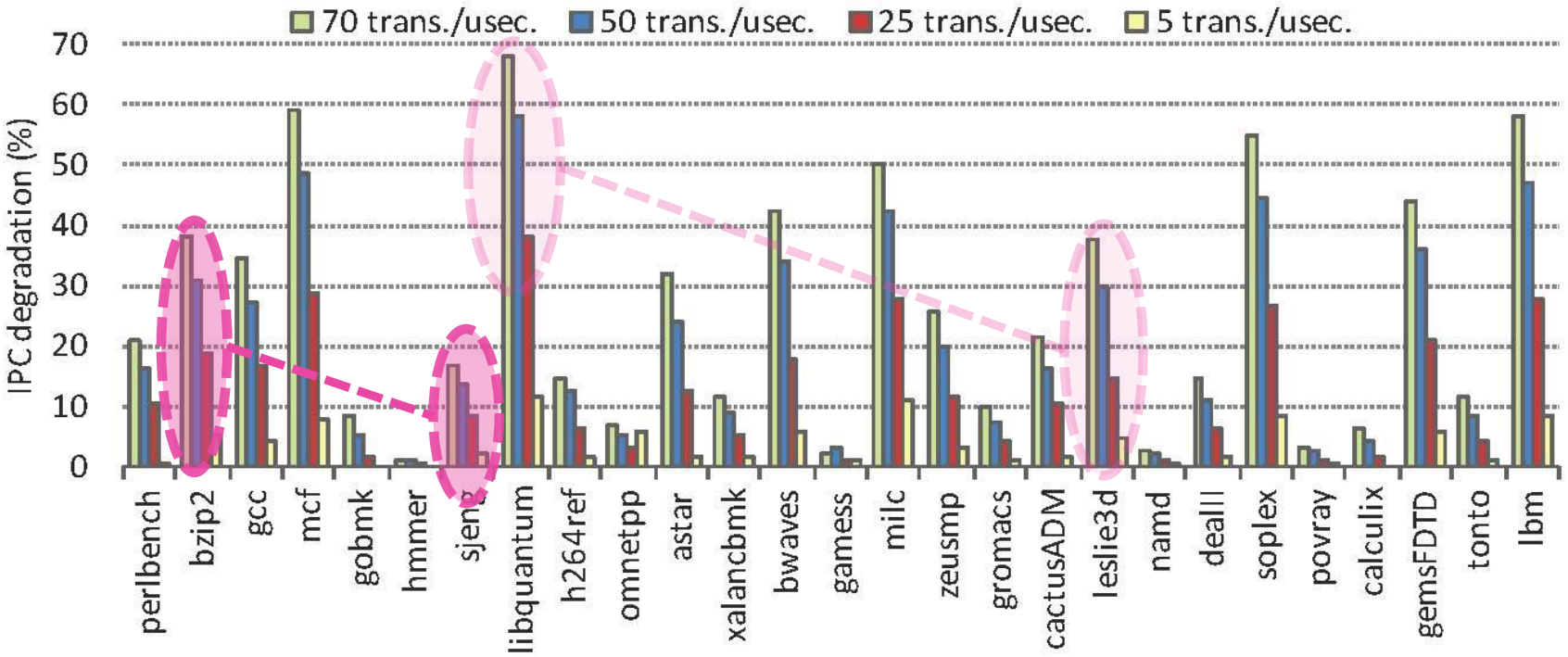
IPC-degradation memory-hierarchy bandwidth-aware scheduler

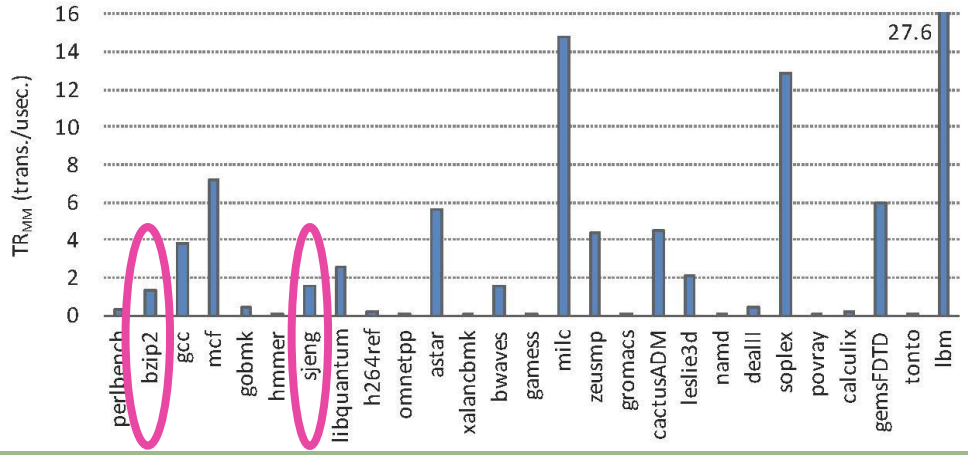




Applications with similar bandwidth suffer different performance degradation

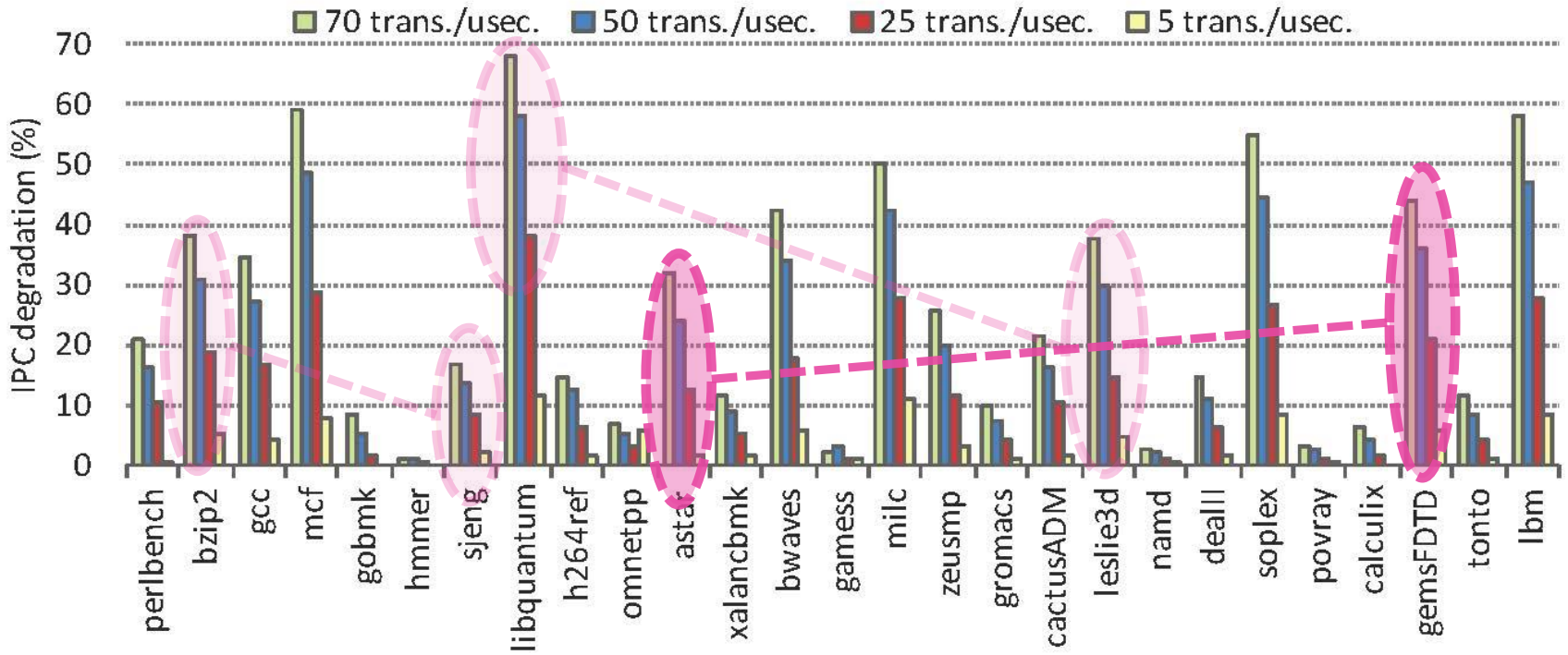
IPC-degradation memory-hierarchy bandwidth-aware scheduler

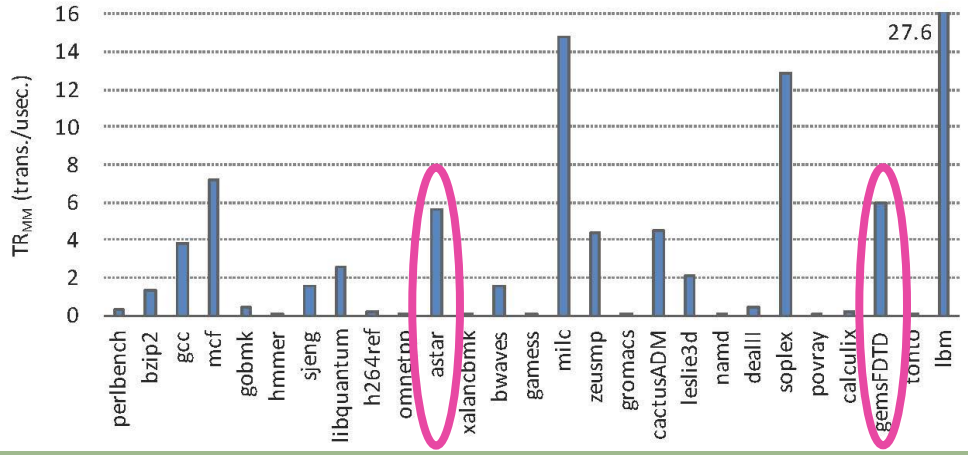




Applications with similar bandwidth suffer different performance degradation

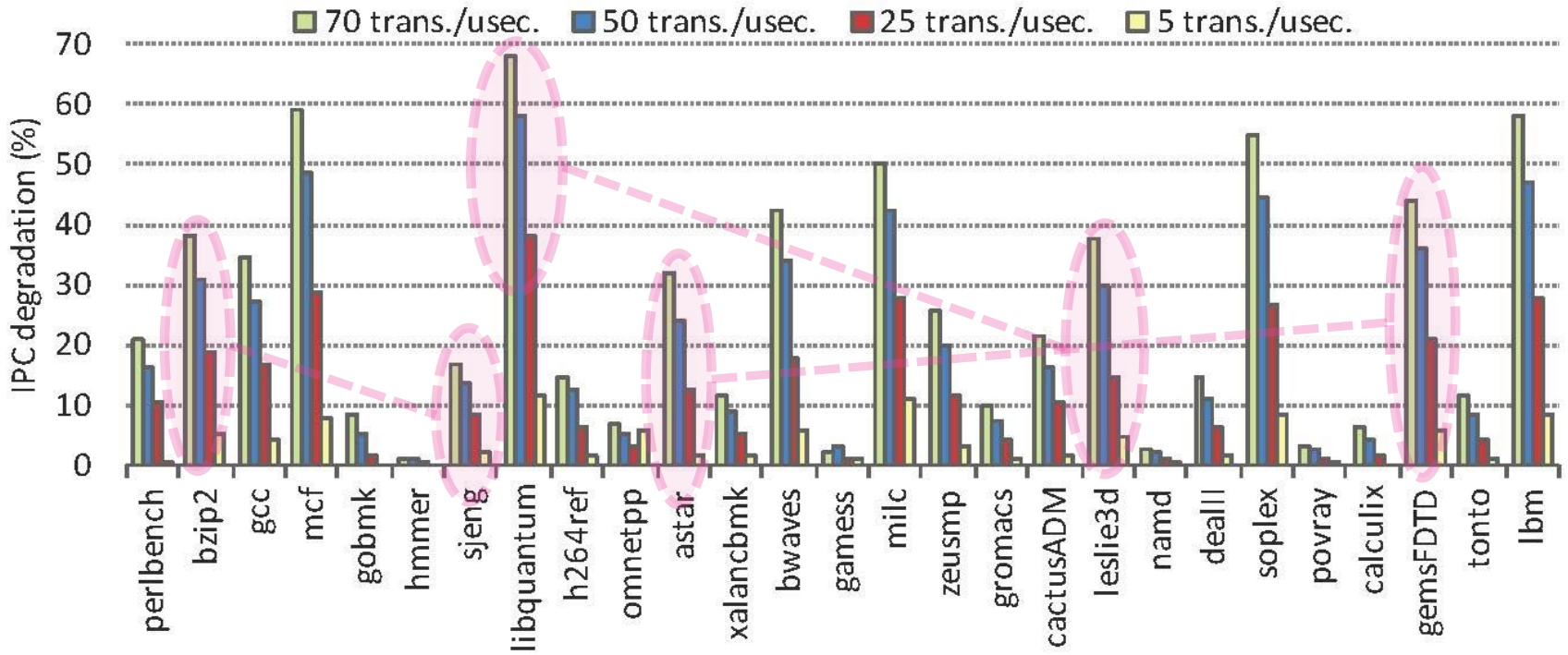
IPC-degradation memory-hierarchy bandwidth-aware scheduler





Applications with similar bandwidth suffer different performance degradation

IPC-degradation memory-hierarchy bandwidth-aware scheduler



Idea:

- Favor the more sensitive processes by running them in scenarios with lower bandwidth utilization
- Penalty coefficient ≈ extra *reserved* bandwidth (but not used)

IPC-degradation memory-hierarchy bandwidth-aware scheduler

Algorithm 3 IPC-degradation memory-hierarchy bandwidth-aware scheduler

Require: Benchmarks submitted with execution time and TR_{MM} in stand-alone execution, and penalty coefficients.

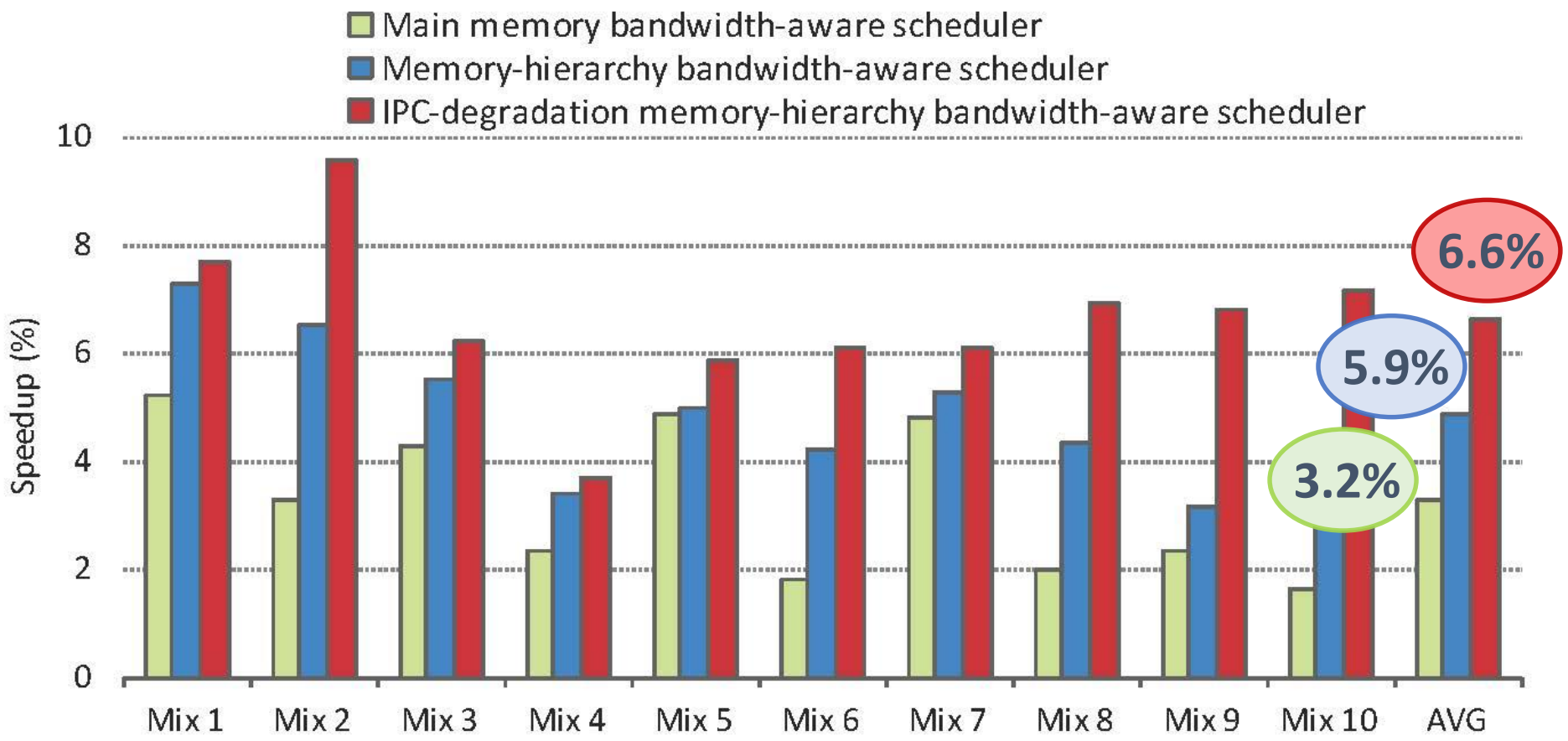
1: $SIABW = \frac{\sum_{p=0}^{P} (TR_{MM}^{p} + PenaltyCoef^{p}) * T^{p}}{\frac{\sum_{p=0}^{P} T^{p}}{\#cores}}$

2: **while** there are unfinished jobs **do**
3: Block the executing processes and place them at the queue tail
4: **for** each process P executed in the last quantum **do**
5: **for** each cache level L **do**
6: Update TR for process P in cache level L
7: **end for**
8: **end for**
9: BW_{Remain} = SIABW
10: Select the process P_head at the queue head
11: $BW_{Remain} -= (TR_{MM}^{P\_head} + PenaltyCoef^{P\_head})$, CPU_{Remain} = #cores−1
12: **while** $CPU_{Remain} > 0$ **do**
13: select the process P that maximizes
14: $FITNESS(p) = \frac{1}{\left| \frac{BW_{Remain}}{CPU_{Remain}} - (TR_{MM}^{P} + Penalty\ Coef^{p}) \right|}$
15: $BW_{Remain} -= (TR_{MM}^{P} + PenaltyCoef^{P})$, CPU_{Remain} − −
16: **end while**
17: **for** each level i in the cache-hierarchy with shared caches beginning from the LLC **do**
18: $AVG\_TR(L_i) = \frac{\sum TR_{L(i)}}{\#Caches\ at\ Li}$
19: **for** each cache in level L_i **do**
20: BW_{Remain} = AVG_TR(L_i), CPU_{Remain} = # cores sharing the cache
21: **while** $CPU_{Remain} > 0$ **do**
22: From the remaining processes selected to share the immediately lower memory level, select the process P that maximizes
23: $FITNESS(p) = \frac{1}{\left| \frac{BW_{Remain}}{CPU_{Remain}} - TR_{Li}^{P} \right|}$
24: $BW_{Remain} -= TR_{Li}^{P}$, CPU_{Remain} − −
25: **end while**
26: **end for**
27: **end for**
28: Unblock the processes, and allocate them in the chosen cores
29: Sleep during the quantum
30: **end while**

Experimental evaluation

Turnaround time speedup (II)

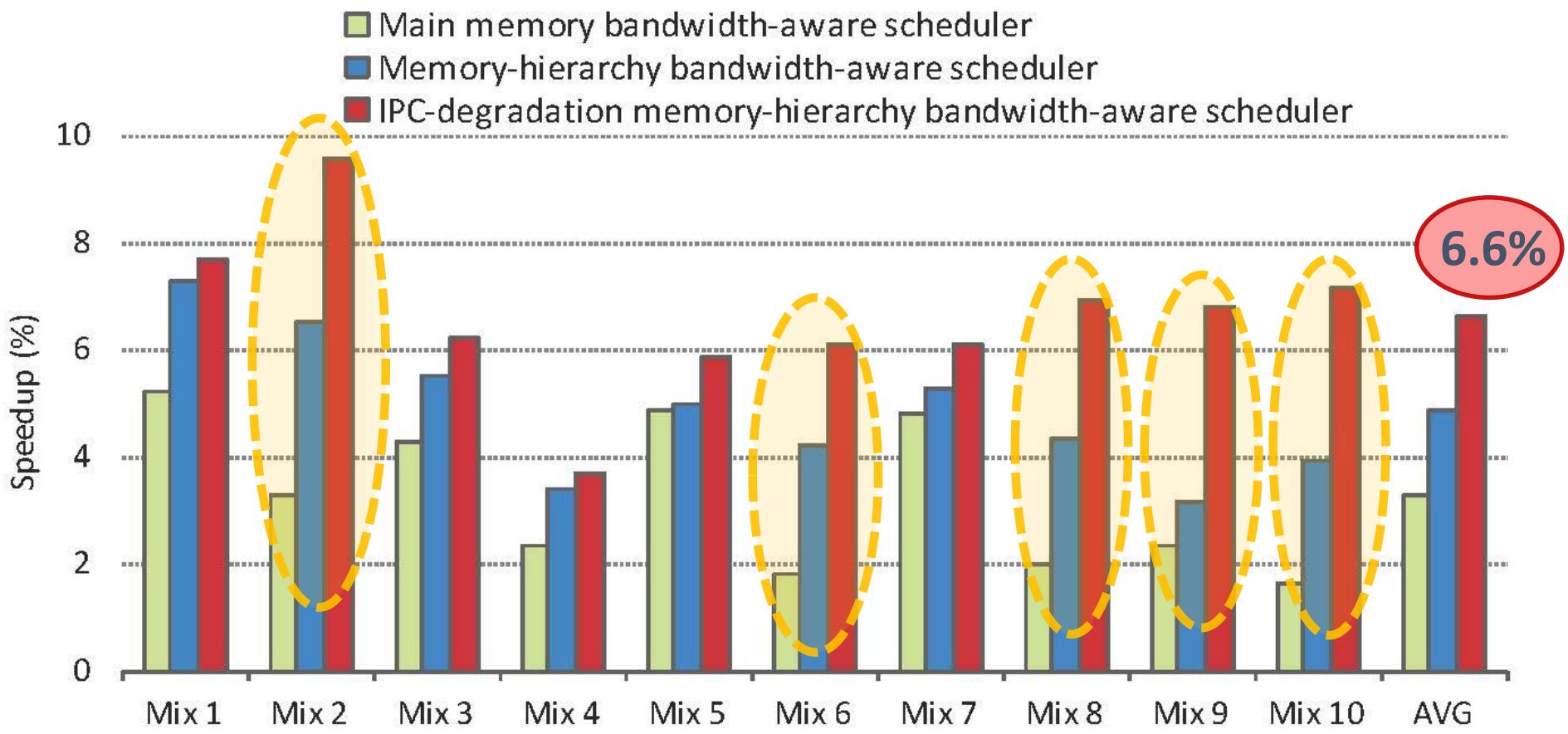
The proposal doubles the average speedup of the state-of-the-art



Experimental evaluation

Turnaround time speedup (II)

The proposal triples the speedup of the state-of-the-art in some workloads



Outline

I. Bandwidth-Aware Scheduling on Multicores

II. Bandwidth-Aware Scheduling on SMT Multicores

   I. Effects of L1 bandwidth on performance

   II. SMT Bandwidth-aware scheduling

   III. Experimental evaluation

III. Progress-Aware Scheduling on SMT Multicores

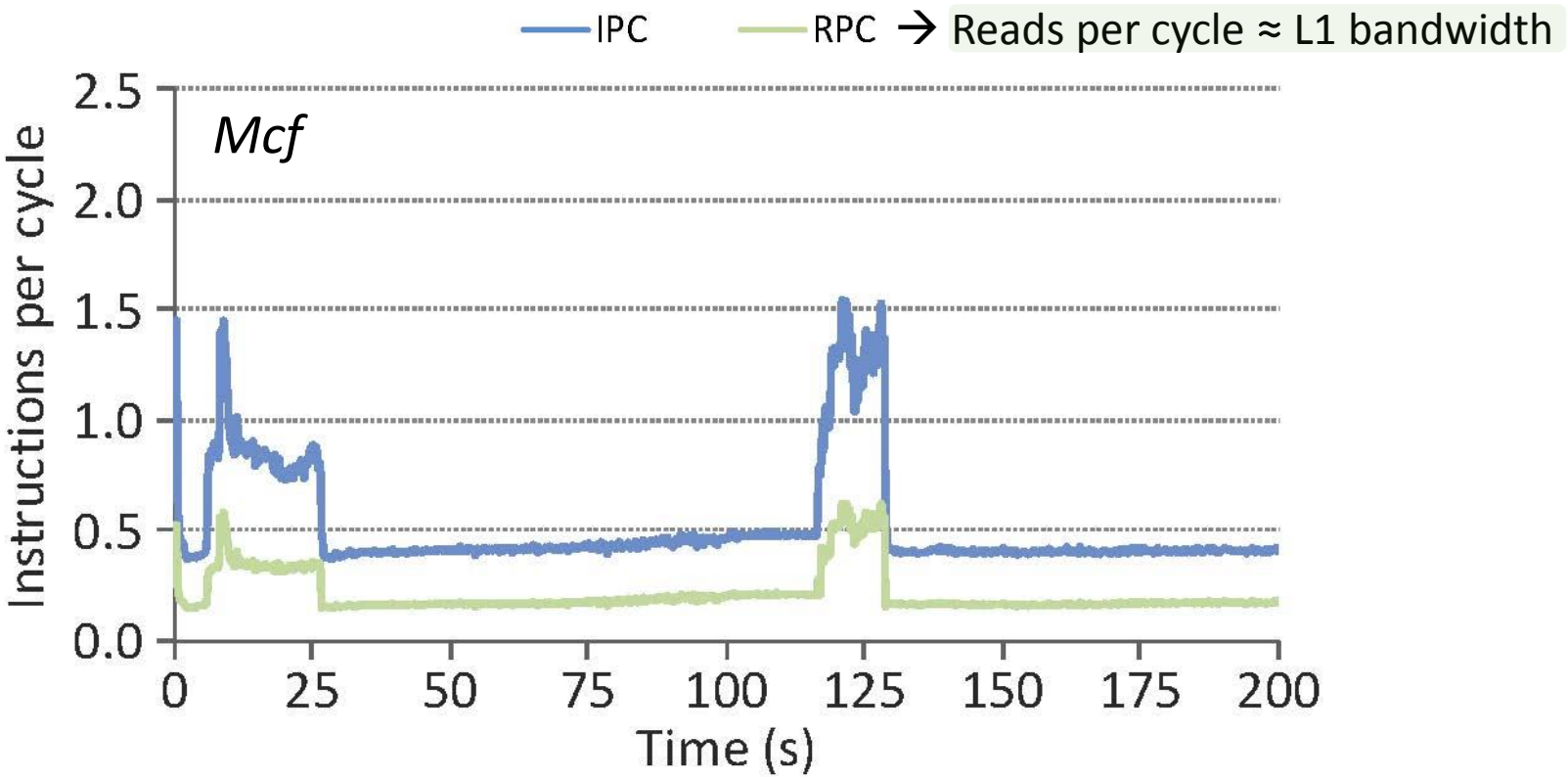
IV. Symbiotic Job Scheduling on the IBM POWER8

Effects of L1 bandwidth on performance

- Stand-alone execution
  - Dynamic L1 bandwidth and IPC
    - Phase behavior
- Concurrent execution
  - Two threads running on an SMT core share the L1 cache
  - L1 bandwidth contention can limit their performance

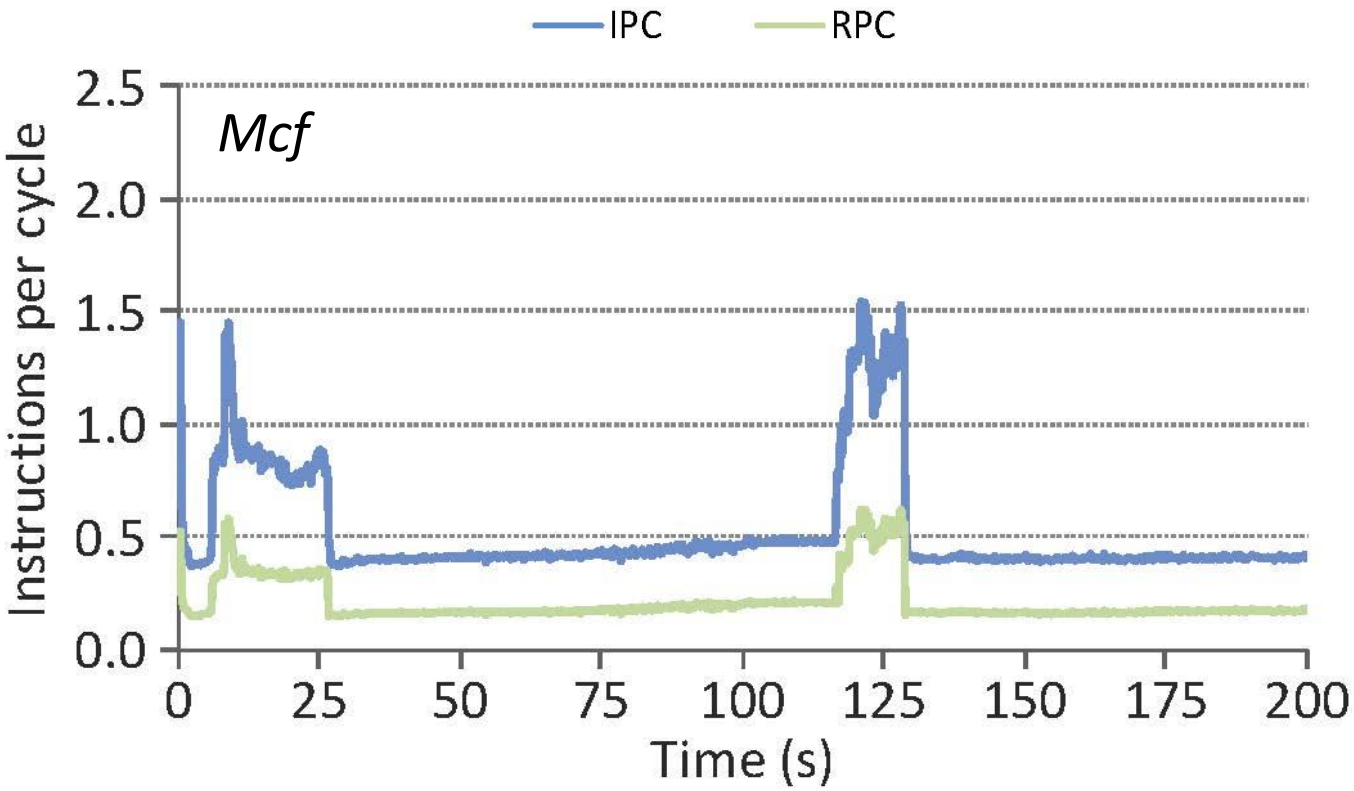
Connection between L1 bandwidth and performance

Standalone execution



Connection between L1 bandwidth and performance

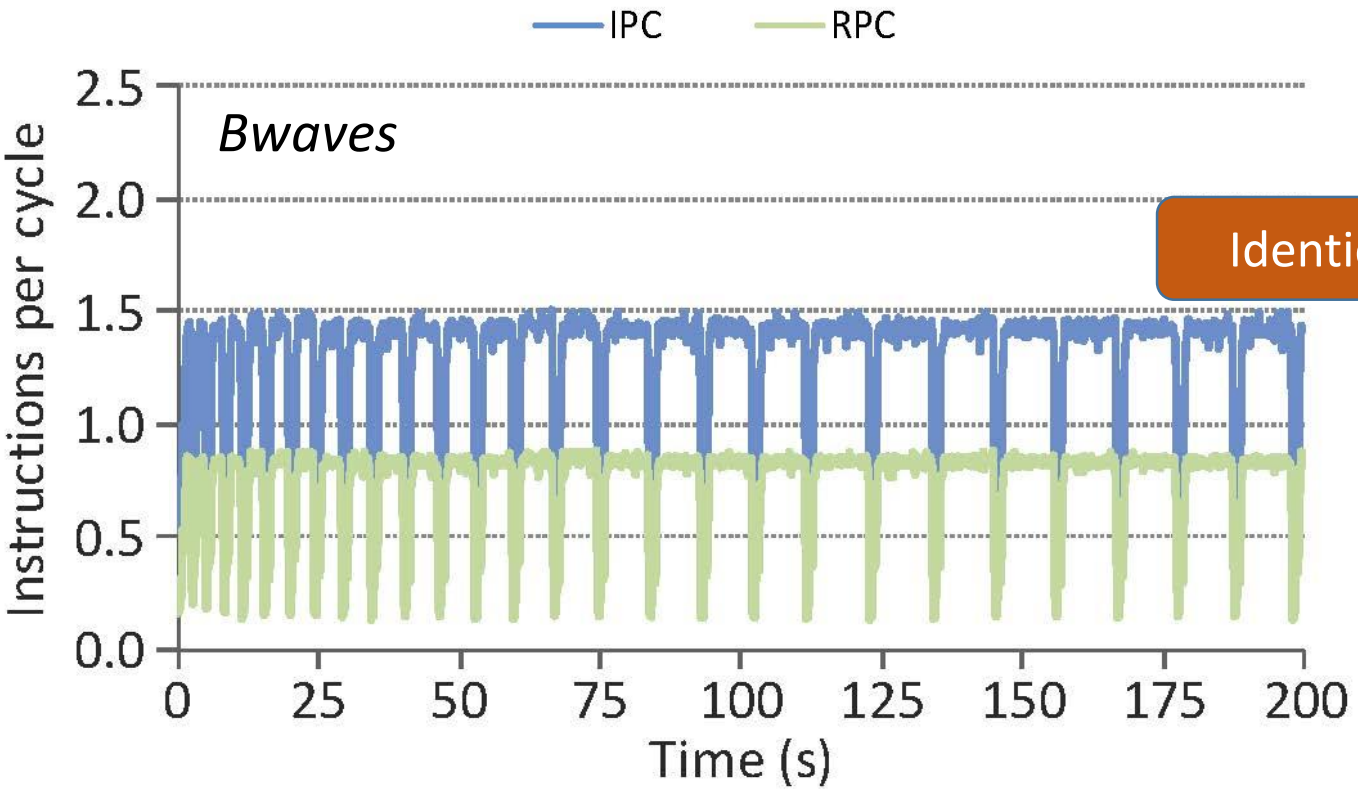
Standalone execution



Strong connection between IPC and RPC

Connection between L1 bandwidth and performance

Standalone execution

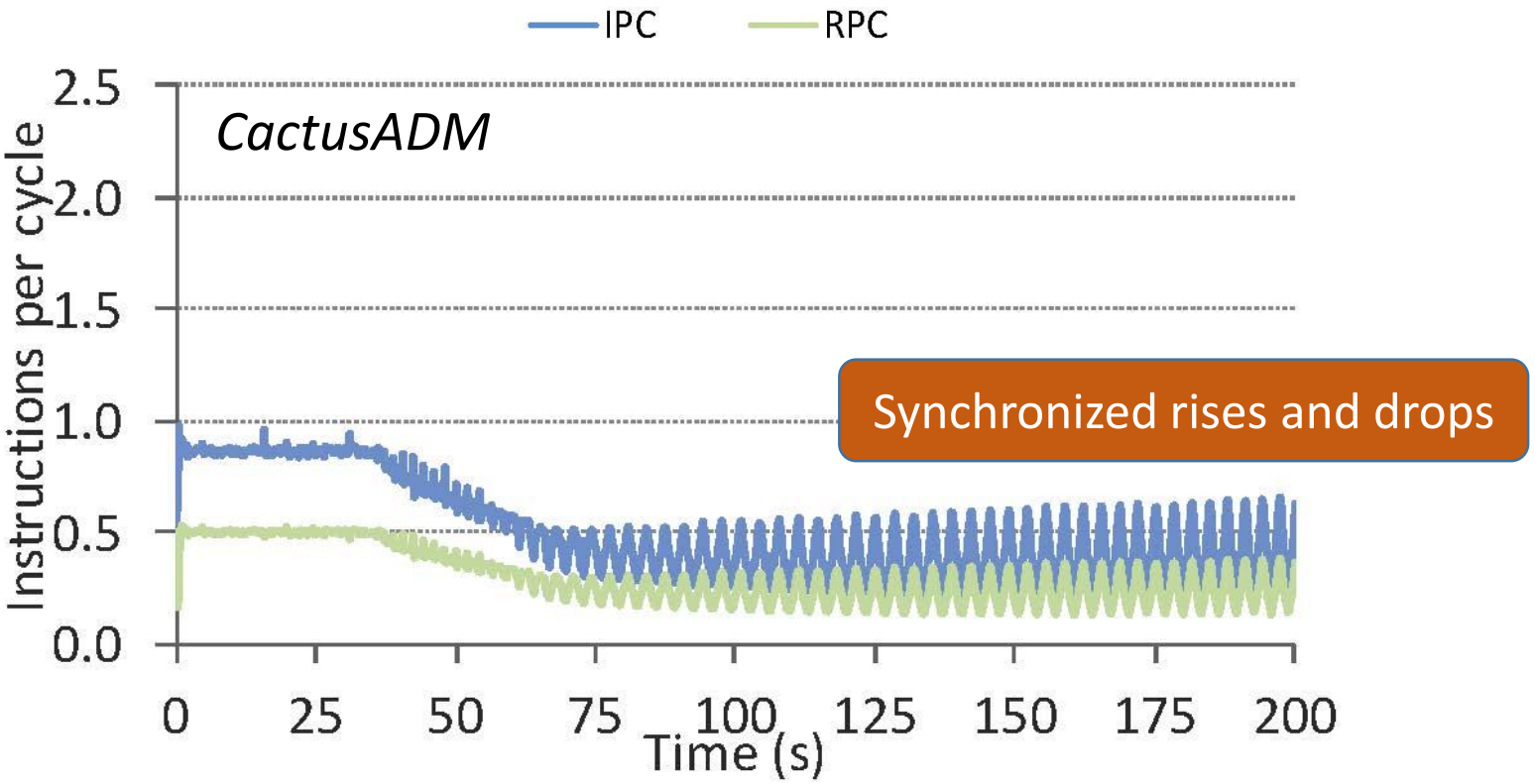


Identical shape

Strong connection between IPC and RPC

Connection between L1 bandwidth and performance

Standalone execution

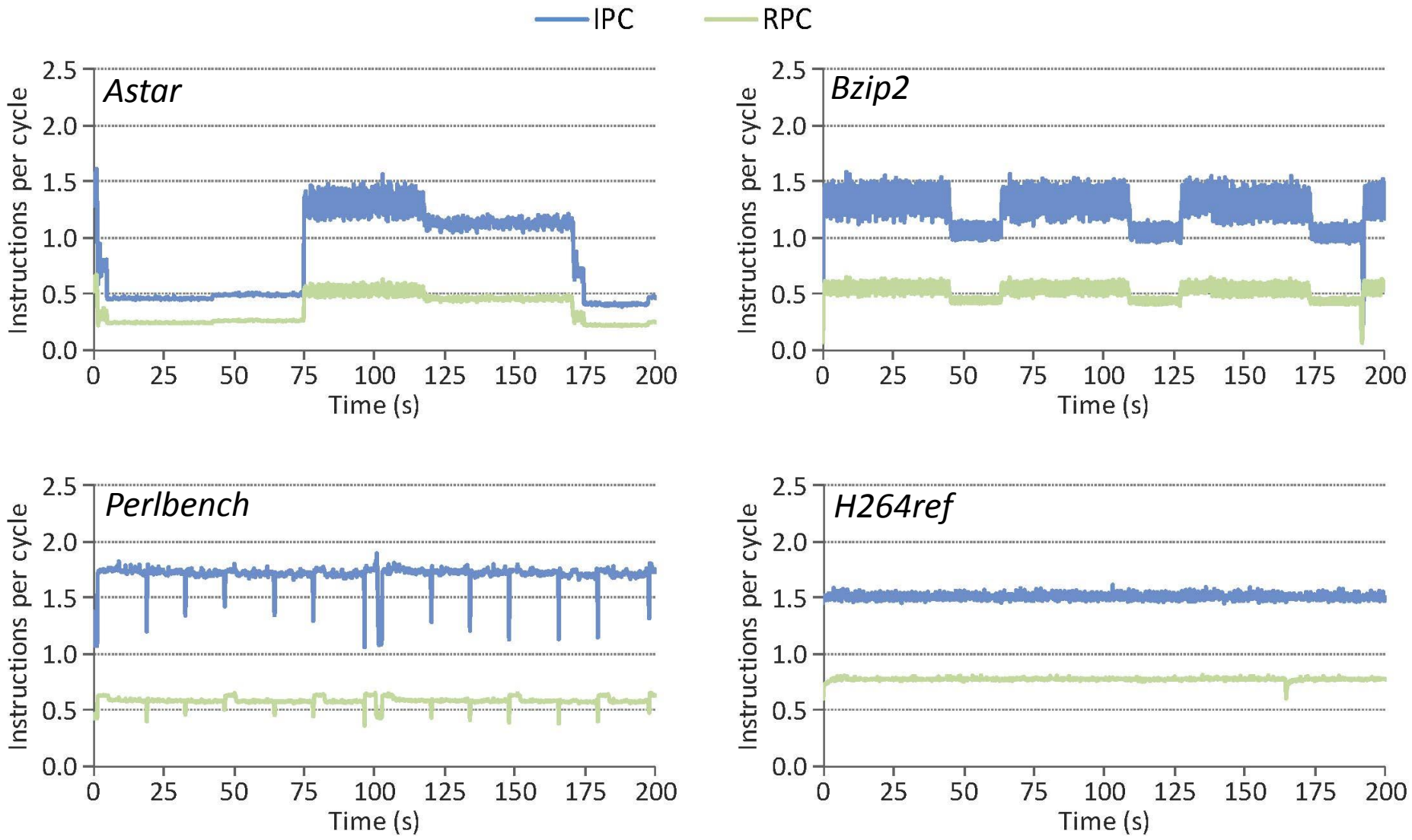


Synchronized rises and drops

Strong connection between IPC and RPC

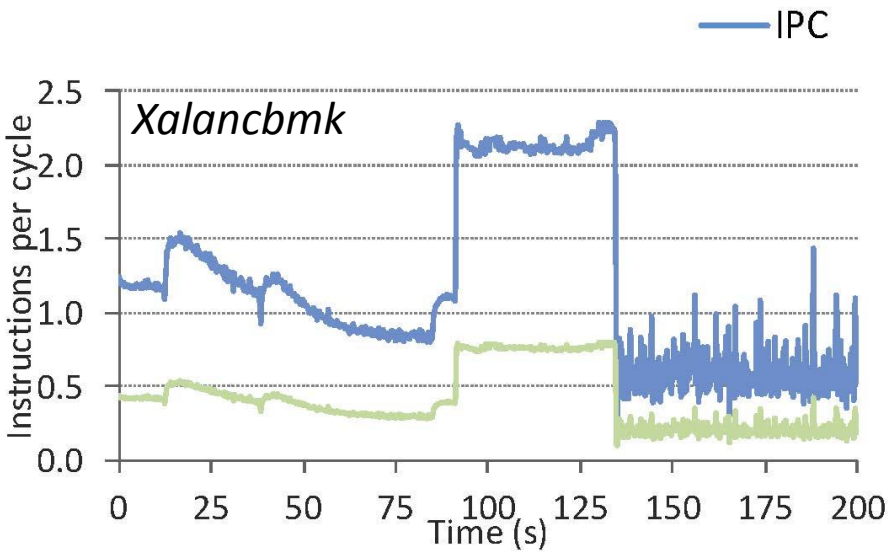
Connection between L1 bandwidth and performance

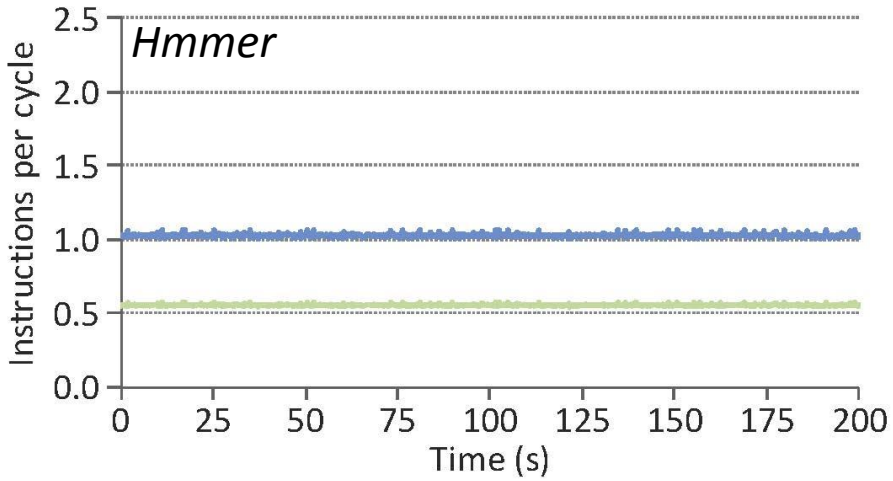
Standalone execution

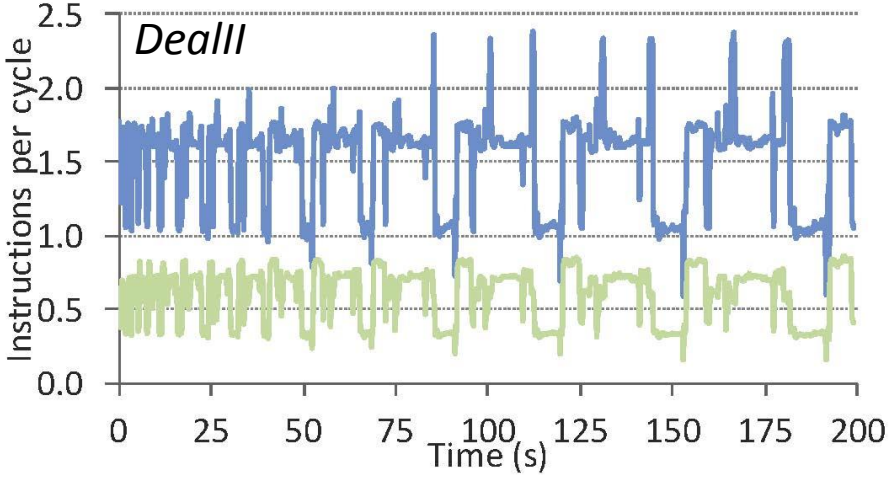


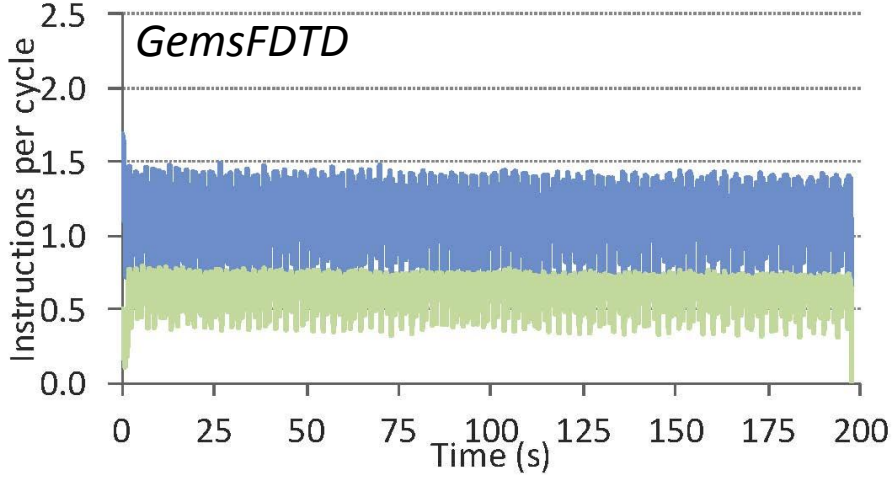
Connection between L1 bandwidth and performance

Standalone execution









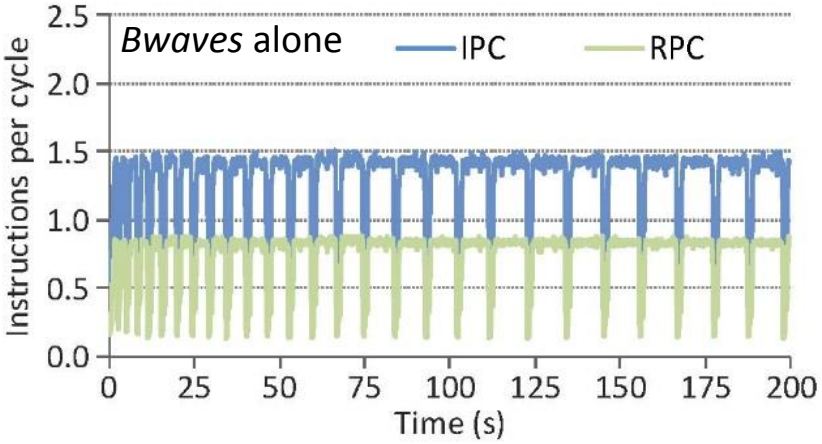
Effects of L1 bandwidth on performance

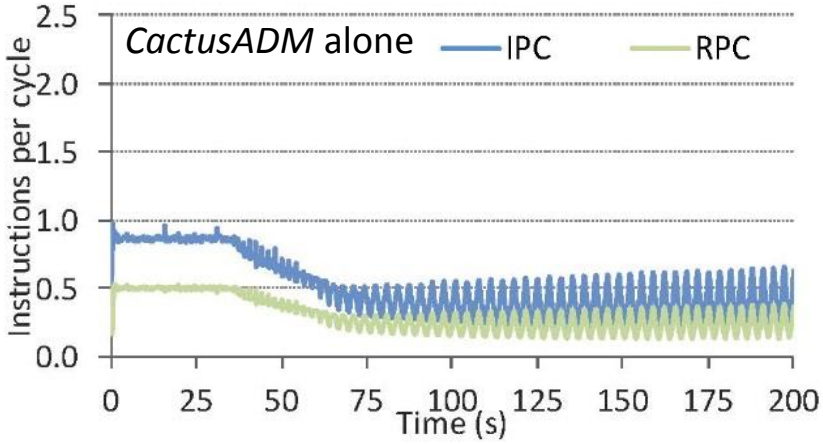
- Stand-alone execution
  - Dynamic L1 bandwidth and IPC
    - Phase behavior

- Concurrent execution
  - Two threads running on an SMT core share the L1 cache
  - L1 bandwidth contention can limit their performance

Connection between L1 bandwidth and performance

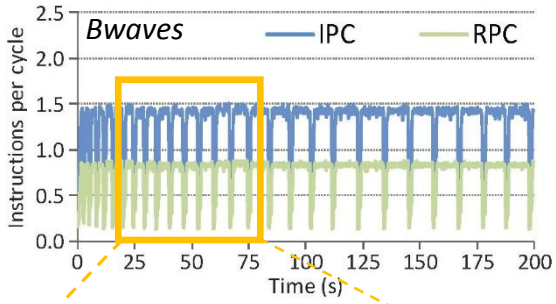
Interferences between co-runners

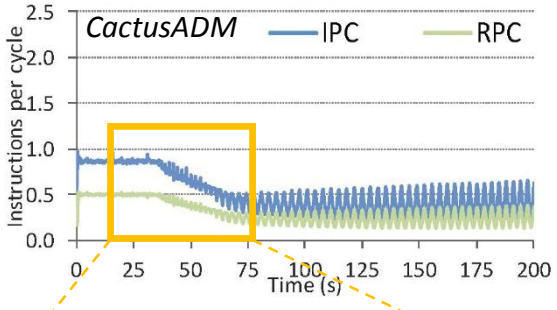


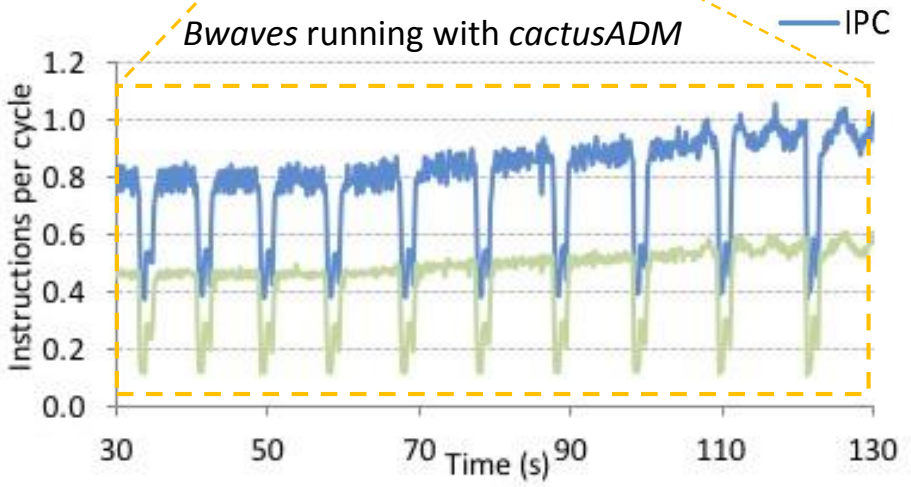


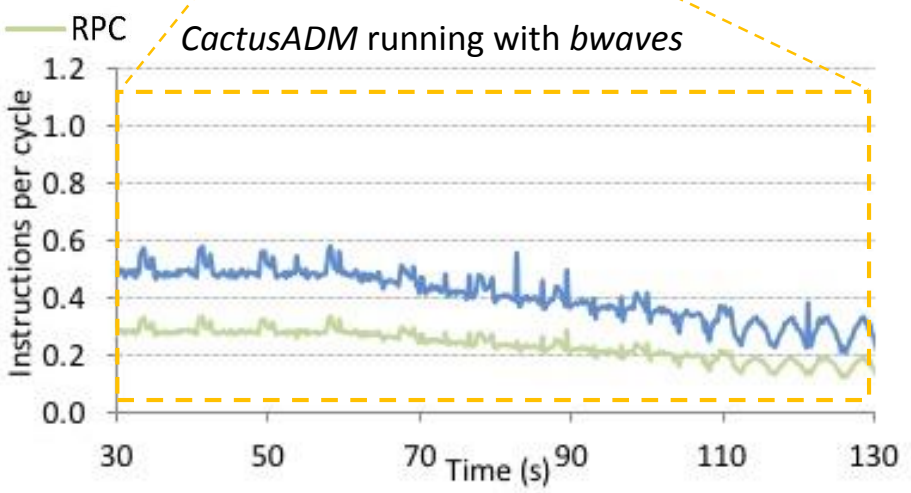
Connection between L1 bandwidth and performance

Interferences between co-runners



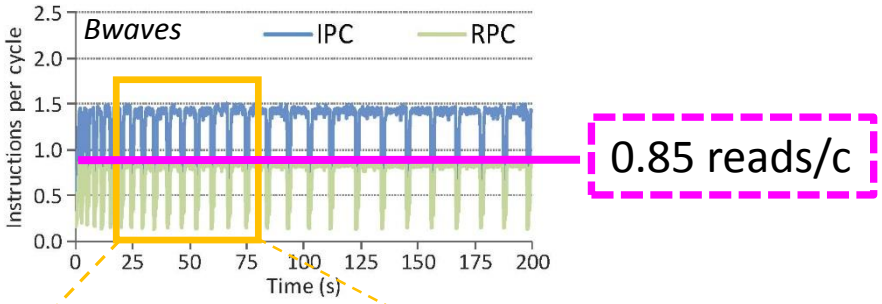


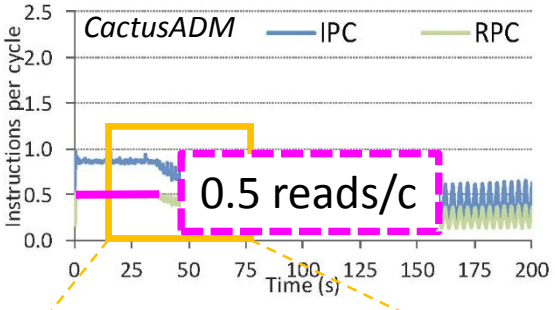


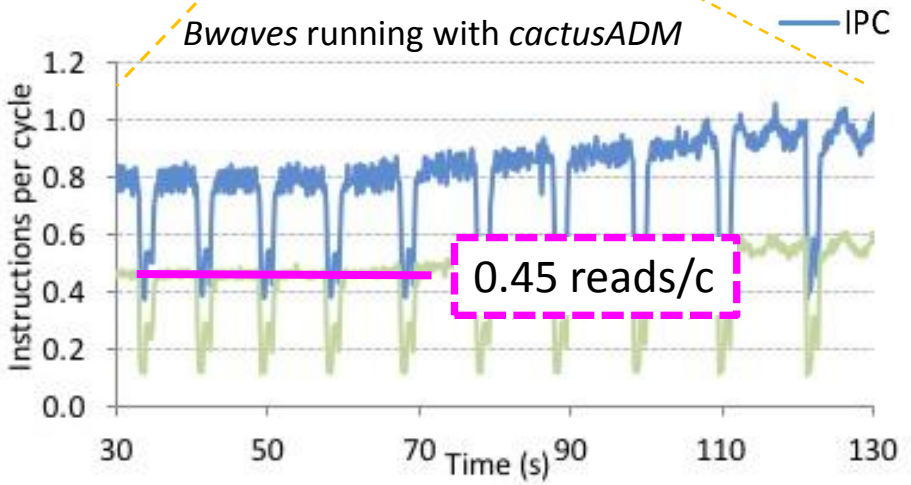


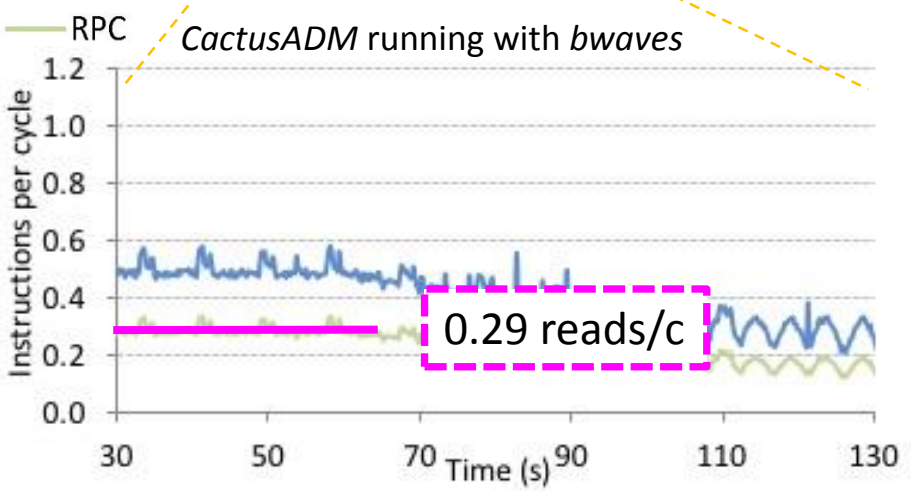
Connection between L1 bandwidth and performance

Interferences between co-runners





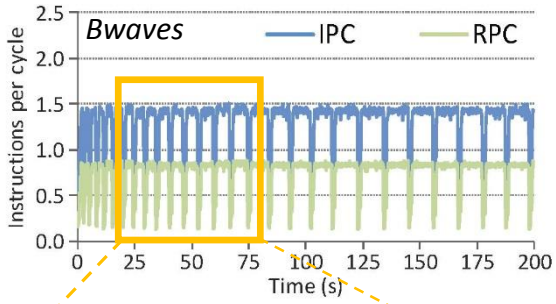


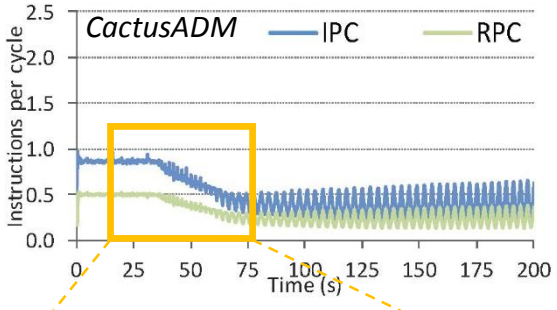


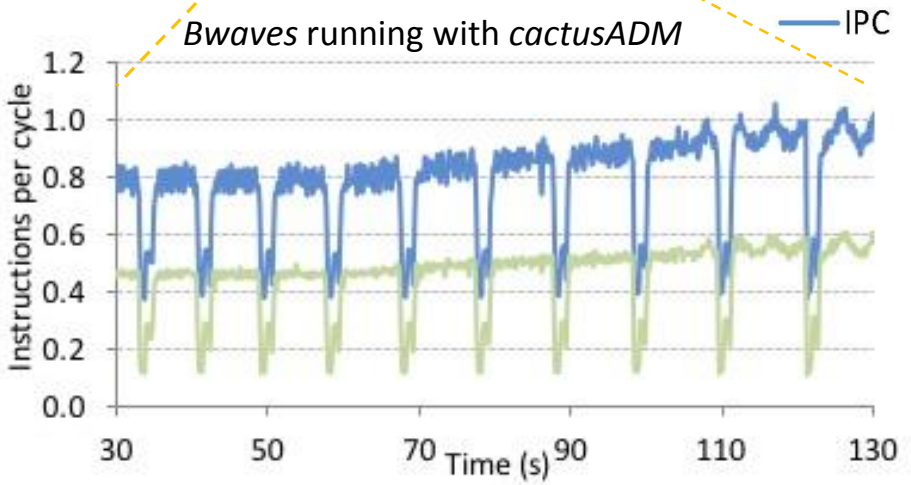
Lower L1 bandwidth

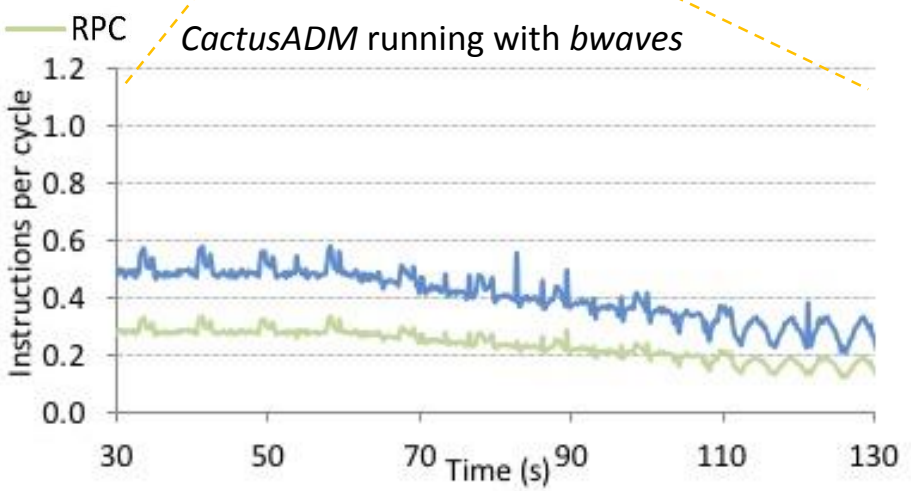
Connection between L1 bandwidth and performance

Interferences between co-runners





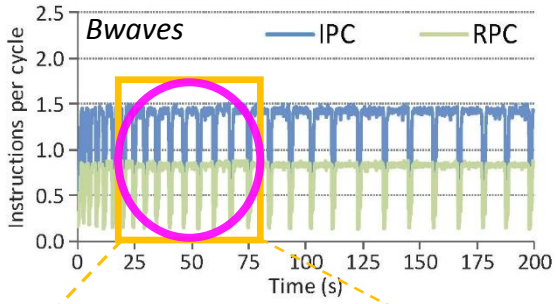


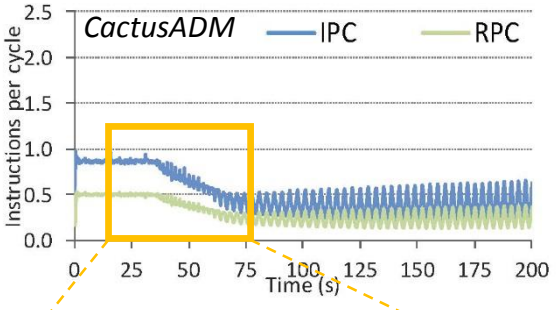


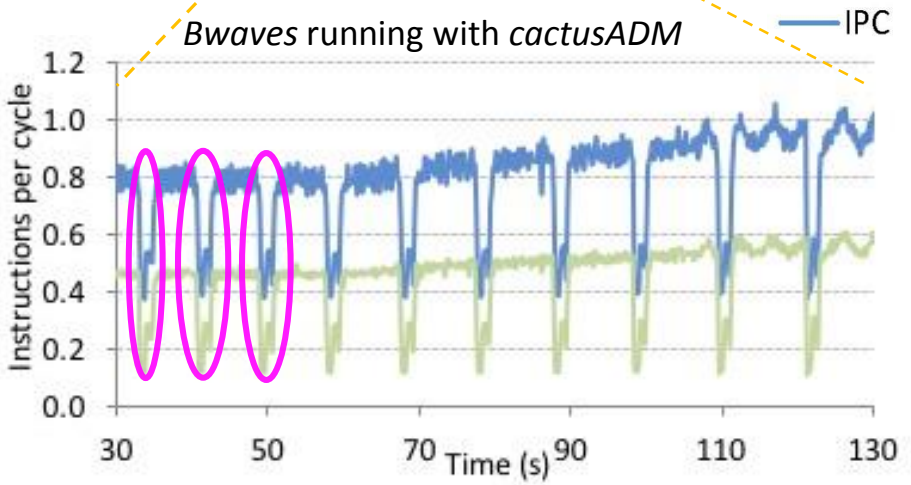
Connection between IPC and RPC of co-runners

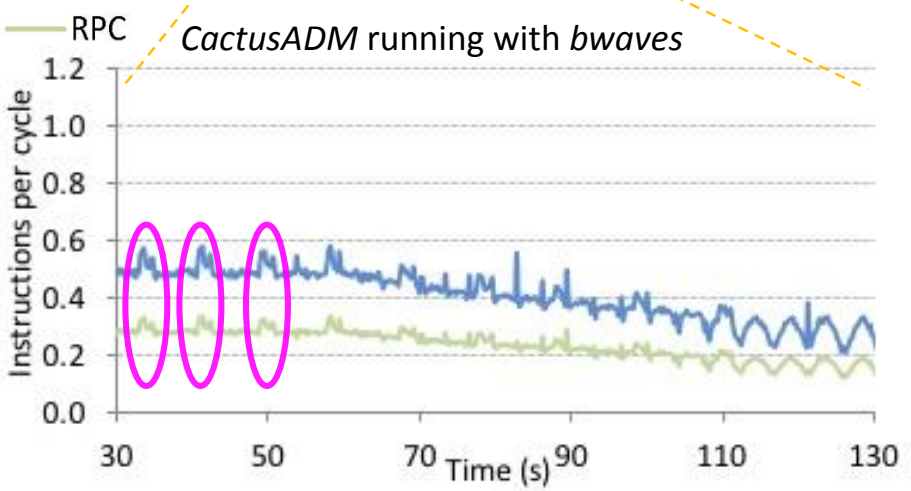
Connection between L1 bandwidth and performance

Interferences between co-runners





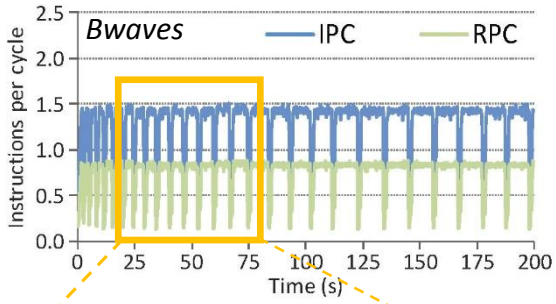


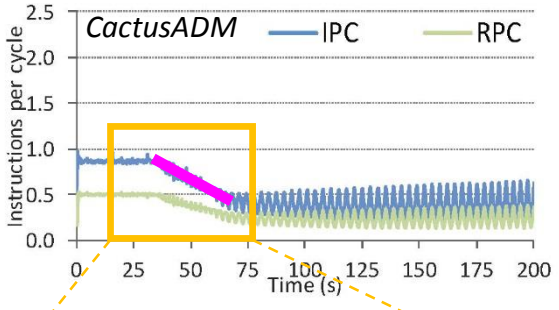


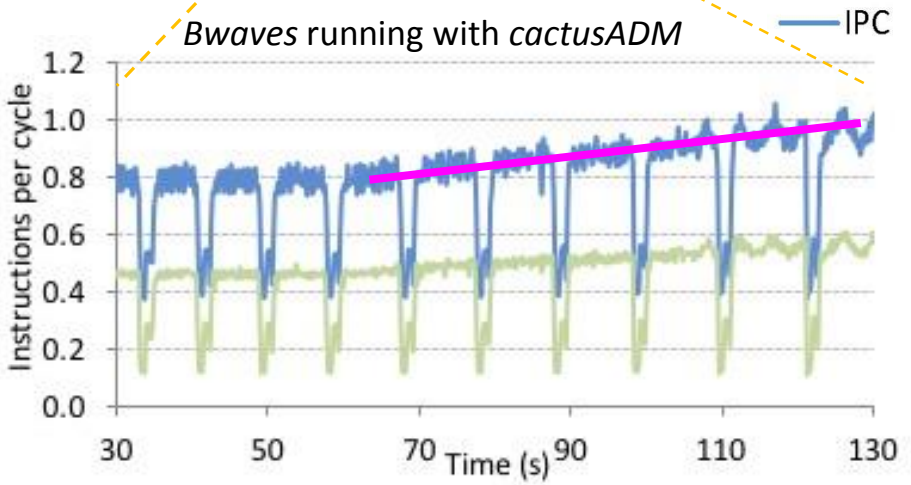
Rises and drops of a process affect the co-runner

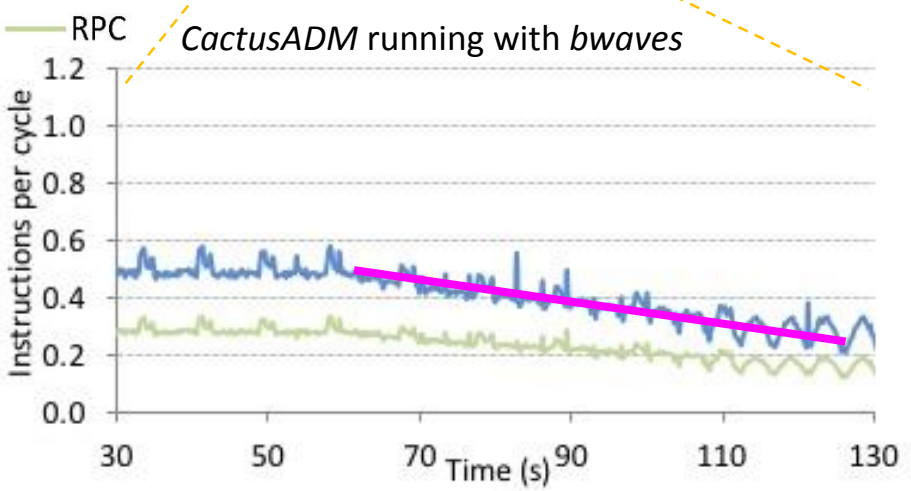
Connection between L1 bandwidth and performance

Interferences between co-runners









Rises and drops of a process affect the co-runner

Dynamic L1 bandwidth-aware process allocation

Goal: mitigate L1 bandwidth contention

- Allocation guided by L1 bandwidth
  - L1 bandwidth updated at runtime with performance counters
    - Adapts to phase behavior
    - No preliminary information required

Dynamic L1 bandwidth-aware process allocation

Goal: mitigate L1 bandwidth contention

- Allocation guided by L1 bandwidth
  - L1 bandwidth updated at runtime with performance counters
    - Adapts to phase behavior
    - No preliminary information required
- L1 requests balanced among the L1 caches to minimize contention

Dynamic L1 bandwidth-aware process allocation

Goal: mitigate L1 bandwidth contention

- Allocation guided by L1 bandwidth
  - L1 bandwidth updated at runtime with performance counters
    - Adapts to phase behavior
    - No preliminary information required
- L1 requests balanced among the L1 caches to minimize contention

Algorithm 6 Dynamic L1 bandwidth-aware process allocation policy

1: Sort the selected processes in ascending TR_{L1}
2: **while** there are unallocated processes **do**
3: Select the processes P_{head} and P_{tail} with maximum and minimum TR_{L1}
4: Allocate P_{head} and P_{tail} to the same core
5: **end while**

Self-reliant main memory bandwidth-aware process selection

Proposed process selection:

- Evenly distributes the memory requests along the workload execution time
- Does not require preliminary information
- Updated every quantum

- Replaces the IABW with the OATR

$$OATR = \frac{\sum_{p=0}^{N} BW_{MM}^{p}}{N} * \#CPUs$$

Average main memory bandwidth of p

Number of hardware contexts

Online Average Transaction Rate

Bandwidth-aware scheduler for SMT multicores

Minimizes bandwidth contention on SMT multicores

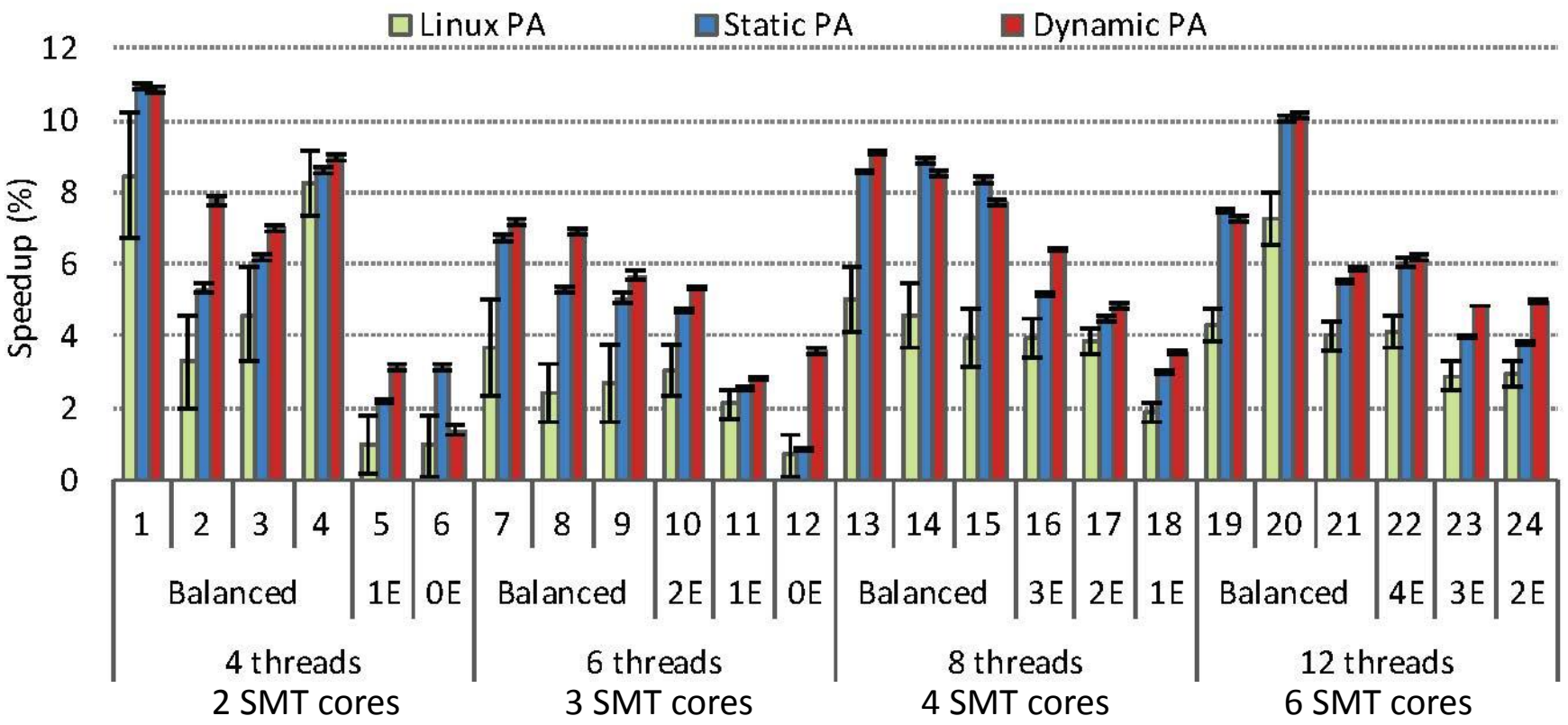
- At main memory and the L1 caches

Algorithm 4 SMT bandwidth-aware scheduler (BaS)

1: Update the bandwidth requirements for the next quantum of each process p executed in the previous quantum:
   - Gather consumed L1 bandwidth (TR^{p}_{L1})
   - Gather consumed main memory bandwidth (TR^{p}_{MM})
2: Process selection - Aware of main memory bandwidth requirements
3: Process allocation - Aware of L1 bandwidth requirements

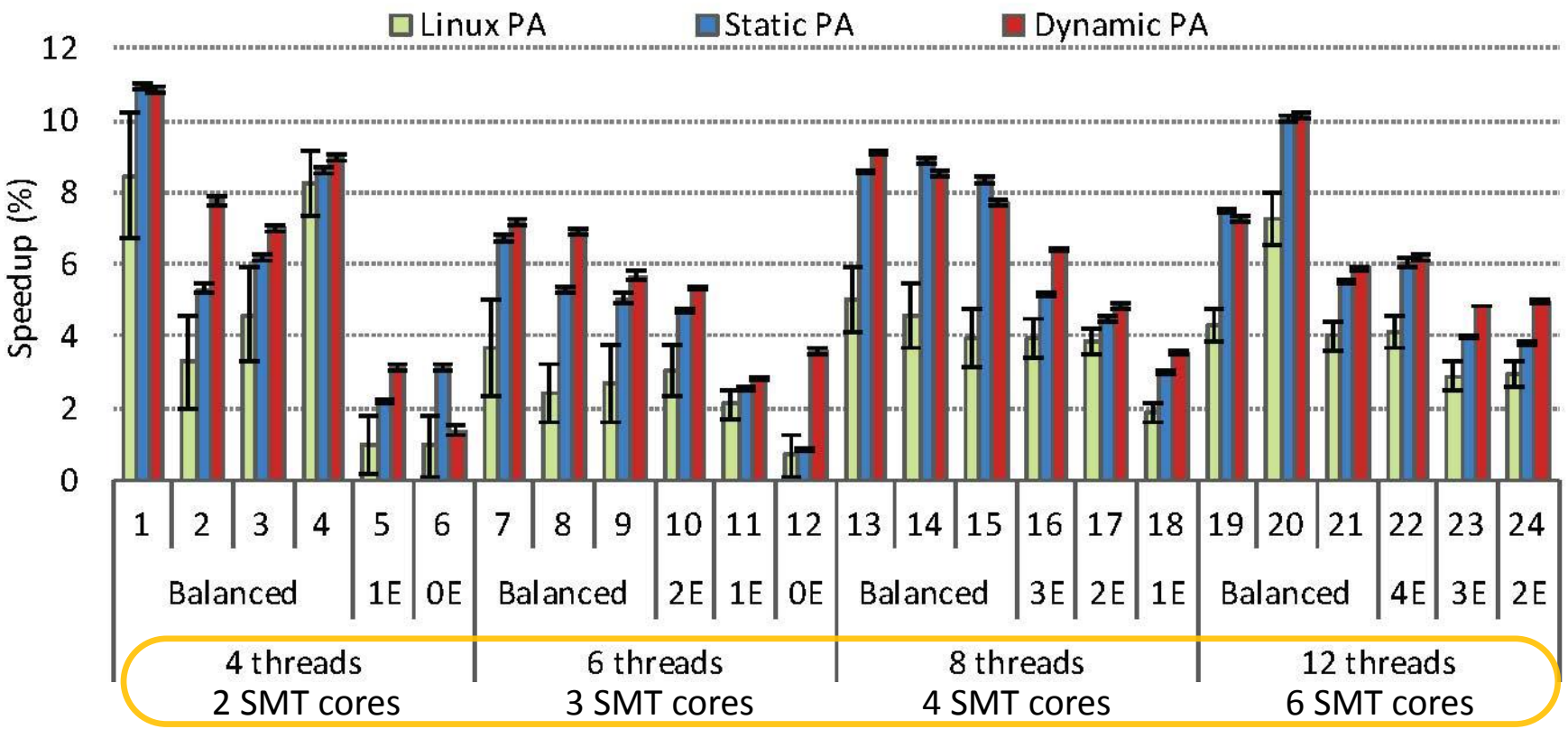
Experimental evaluation

Dynamic process allocation policy – Average IPC



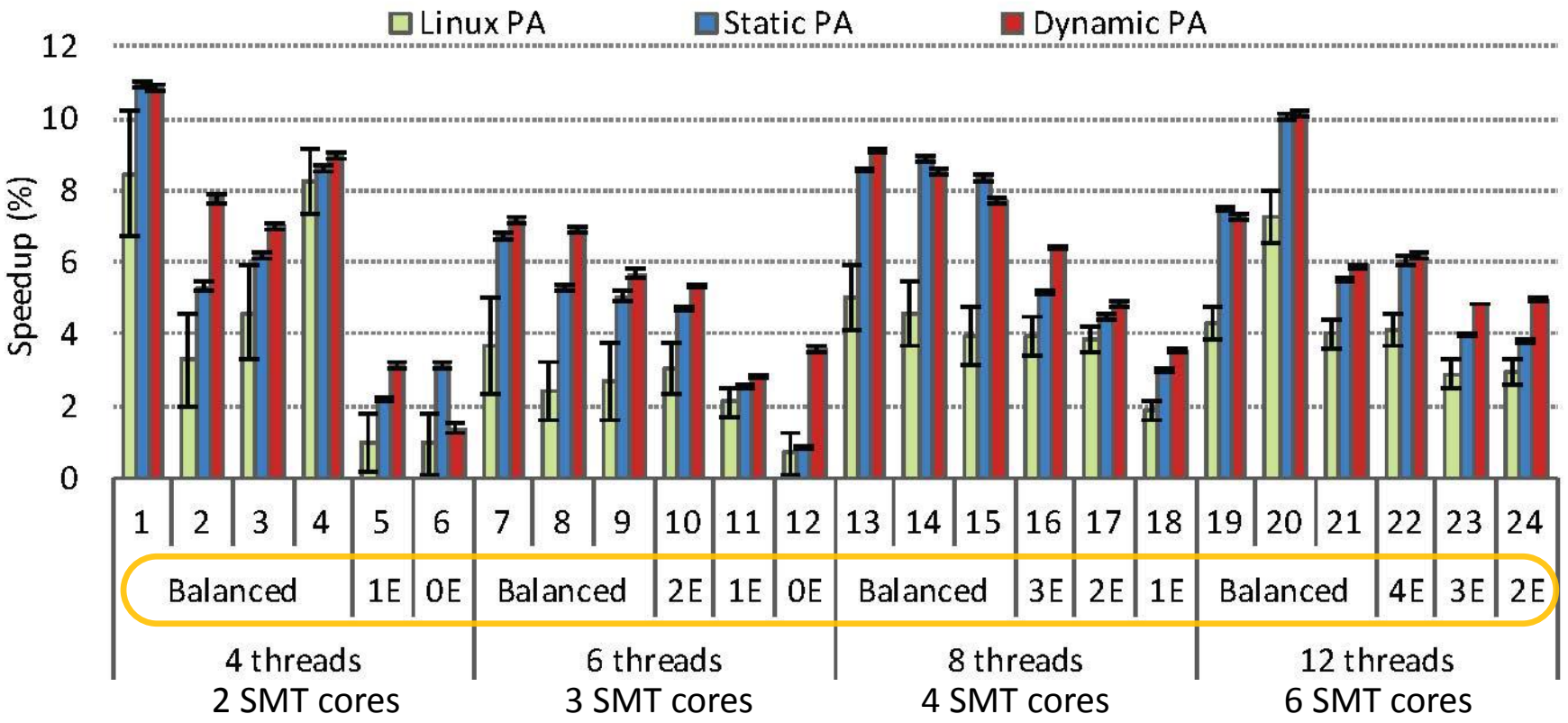
Experimental evaluation

Dynamic process allocation policy – Average IPC



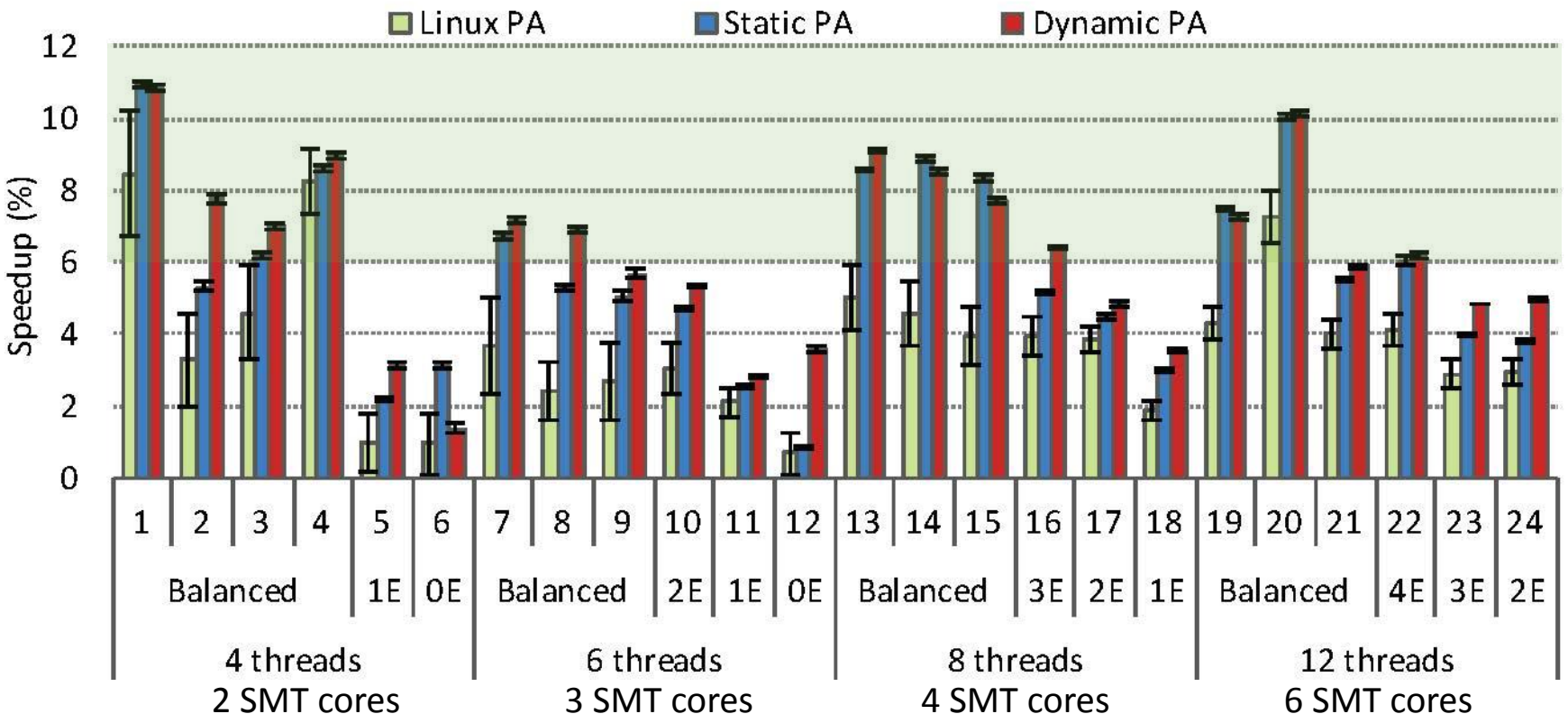
Experimental evaluation

Dynamic process allocation policy – Average IPC



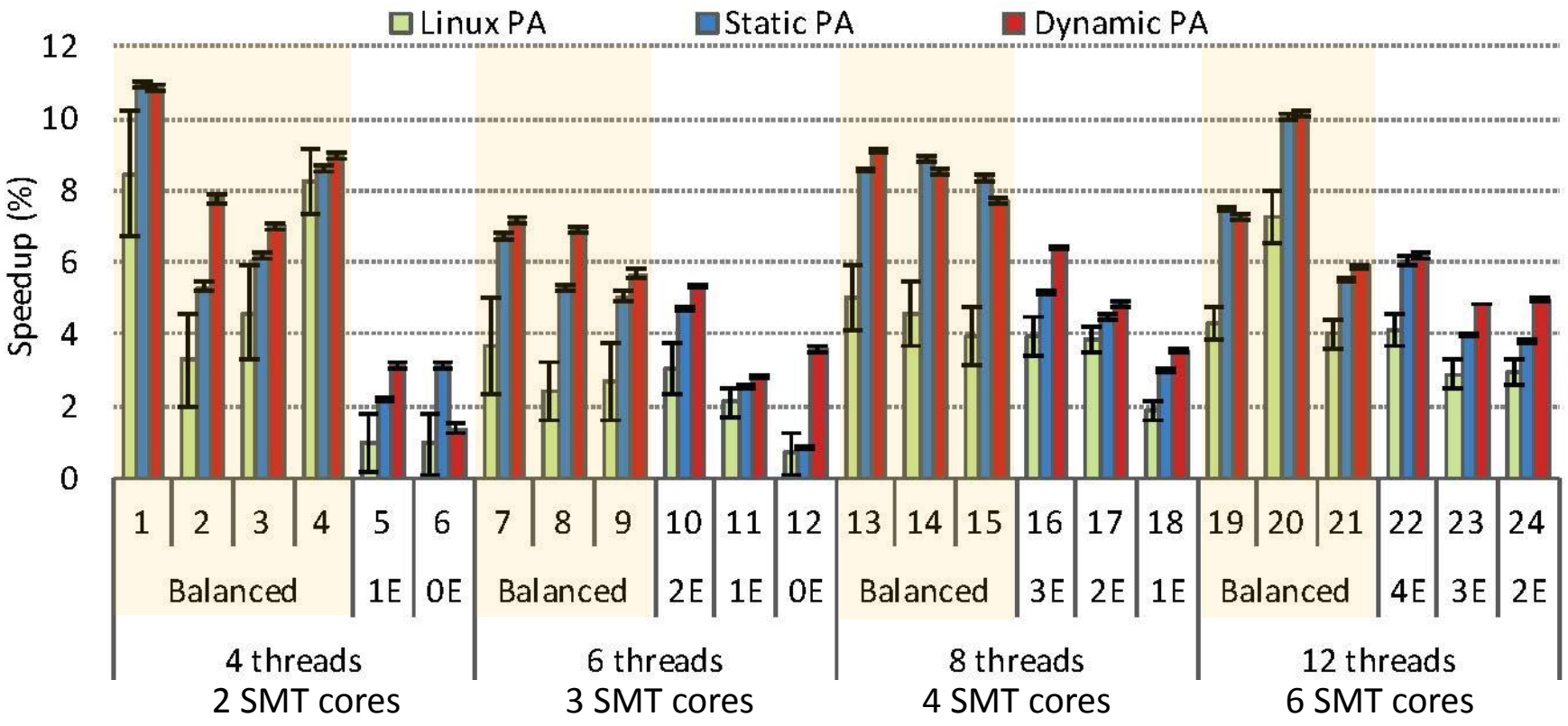
Experimental evaluation

Dynamic process allocation policy – Average IPC



Experimental evaluation

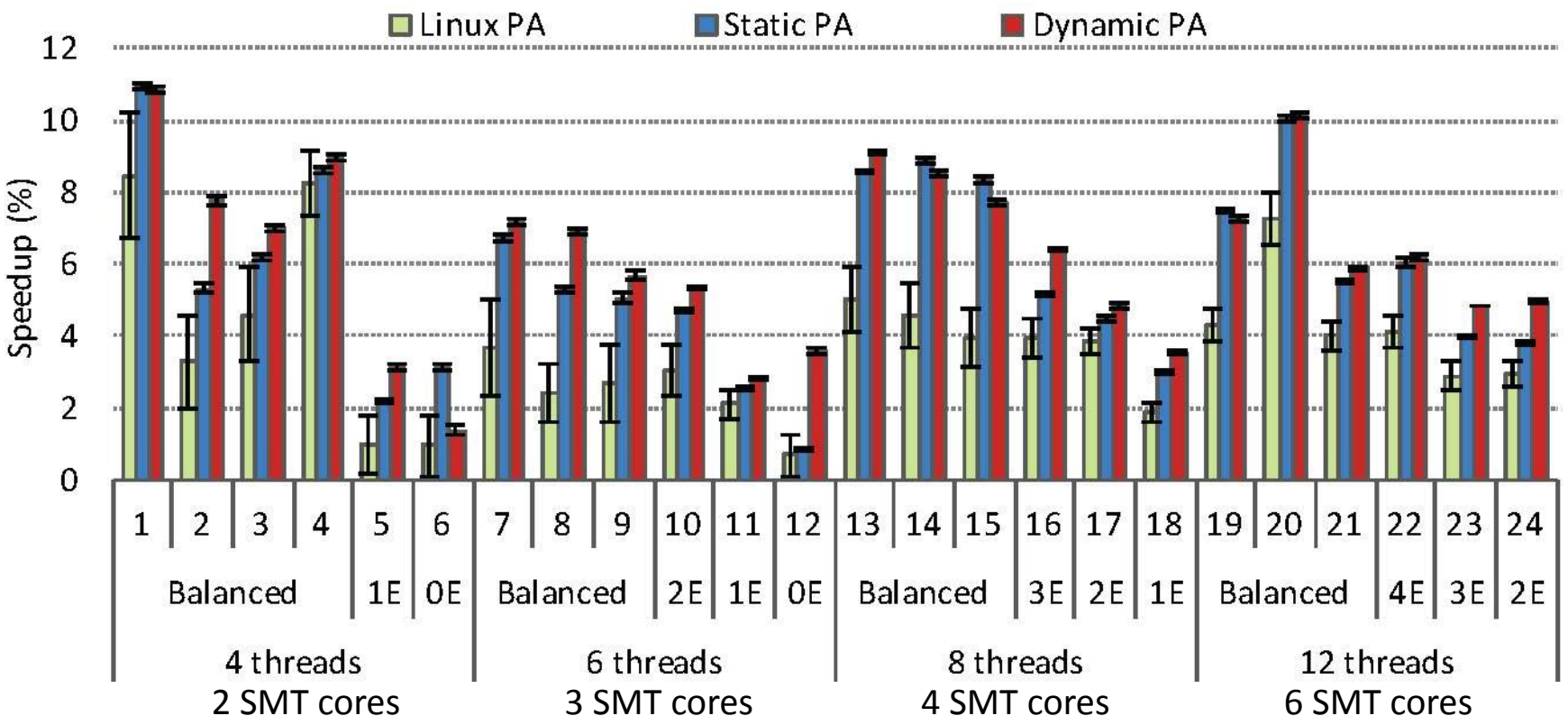
Dynamic process allocation policy – Average IPC



Higher speedups in balanced mixes

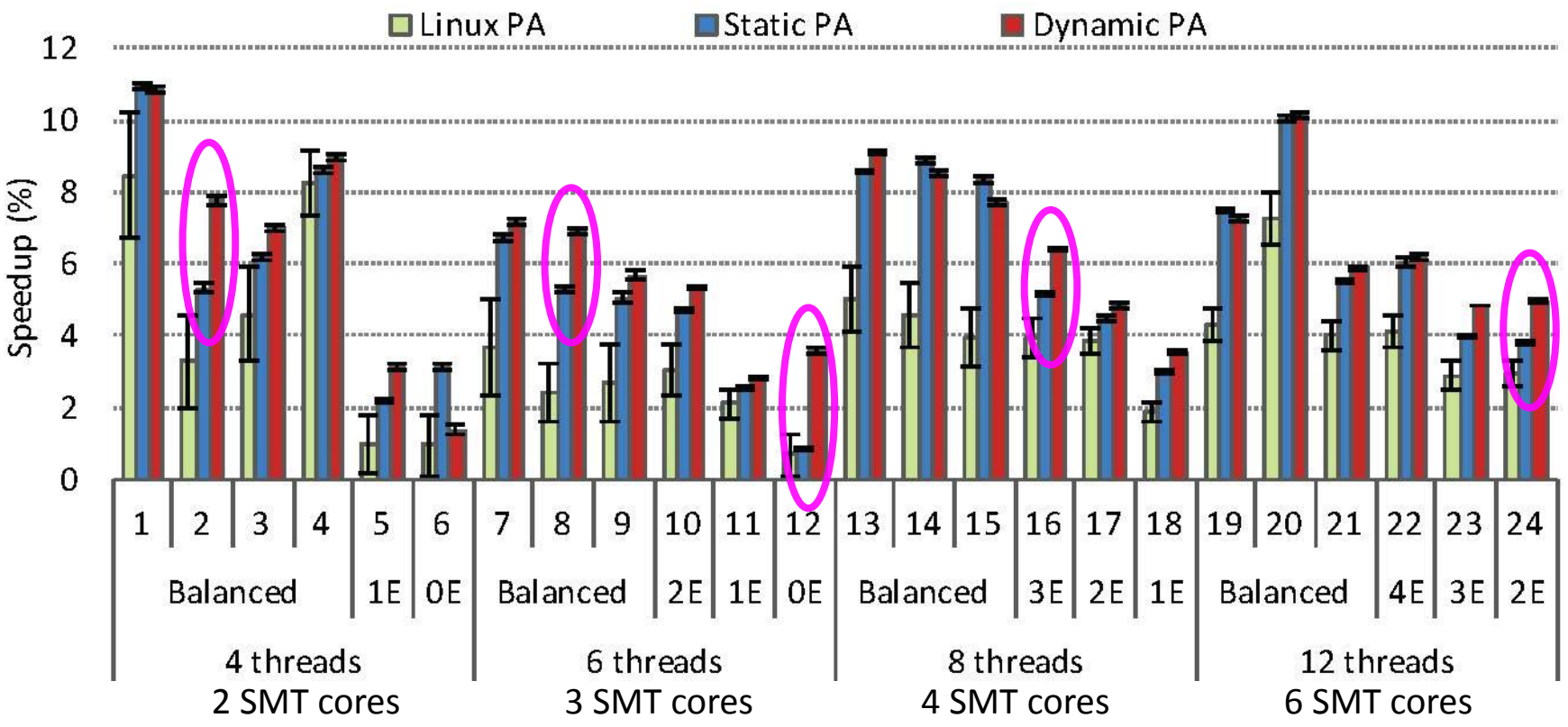
Experimental evaluation

Dynamic process allocation policy – Average IPC



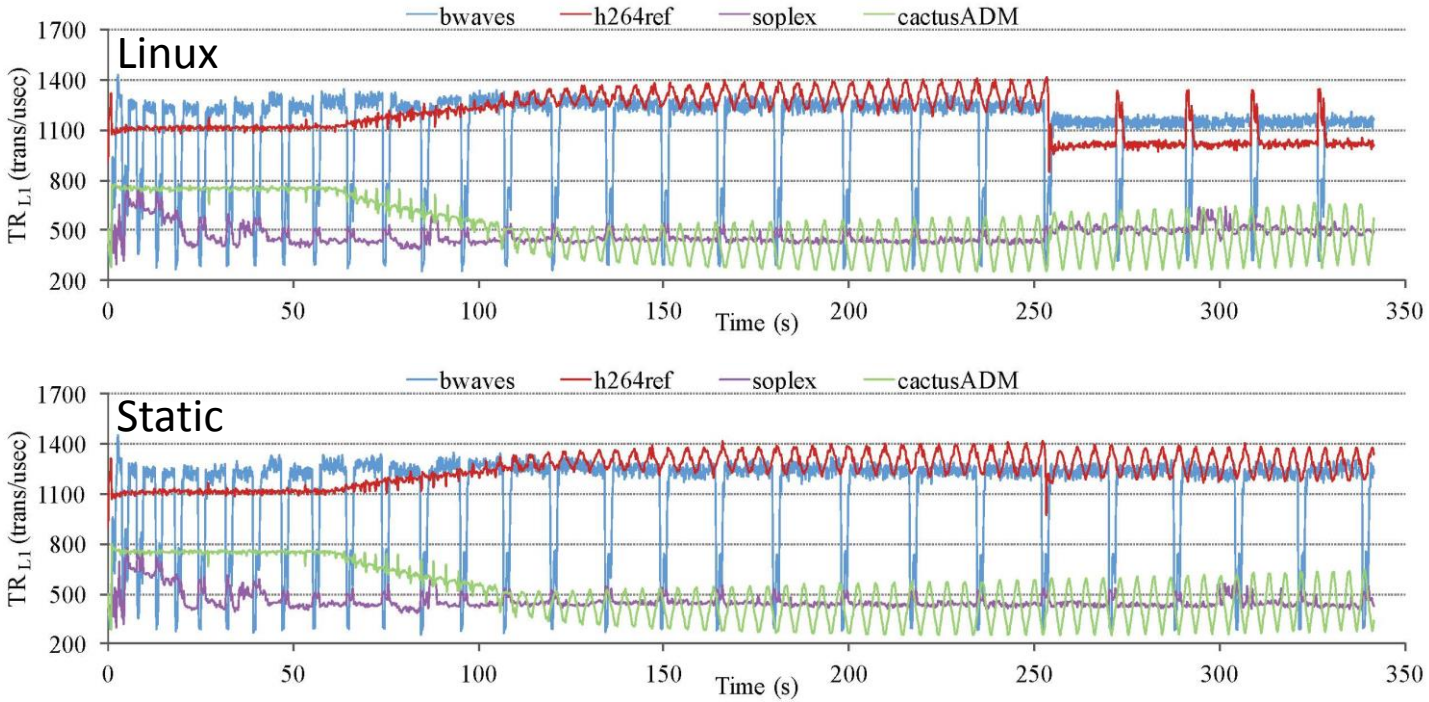
Experimental evaluation

Dynamic process allocation policy – Average IPC



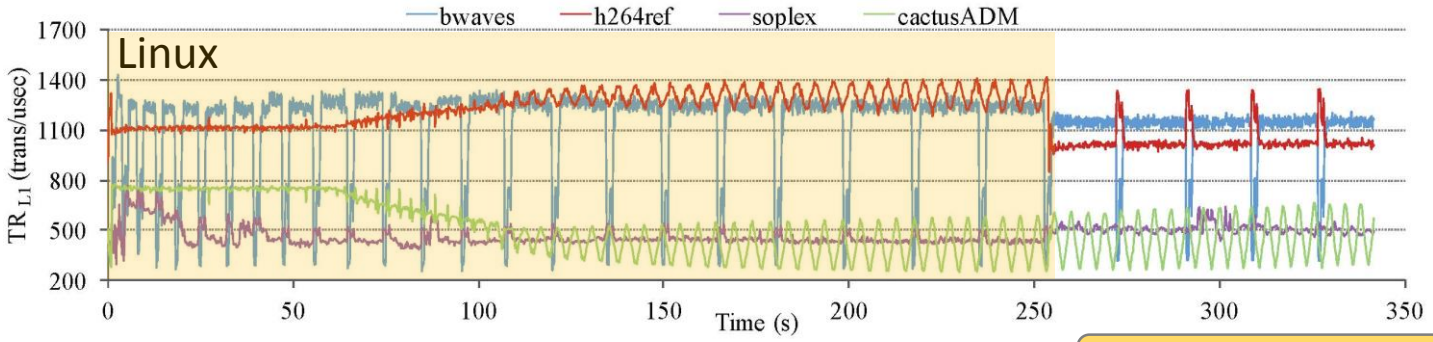
Experimental evaluation

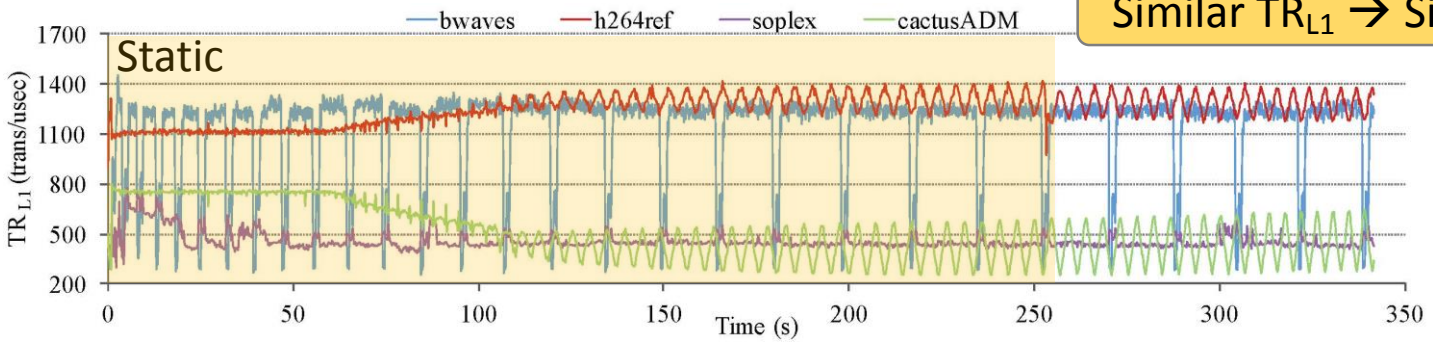
Process allocation policies – Evolution of L1 bandwidth



Experimental evaluation

Process allocation policies – Evolution of L1 bandwidth

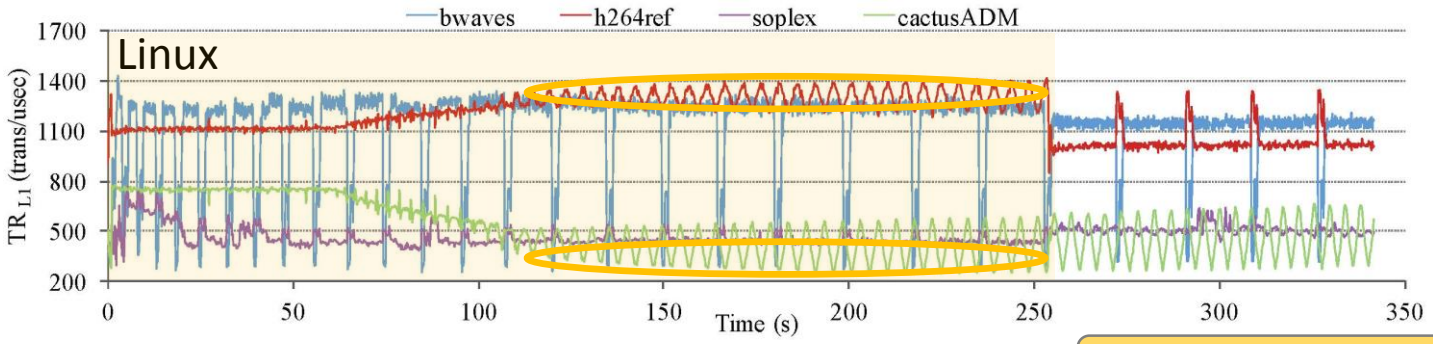


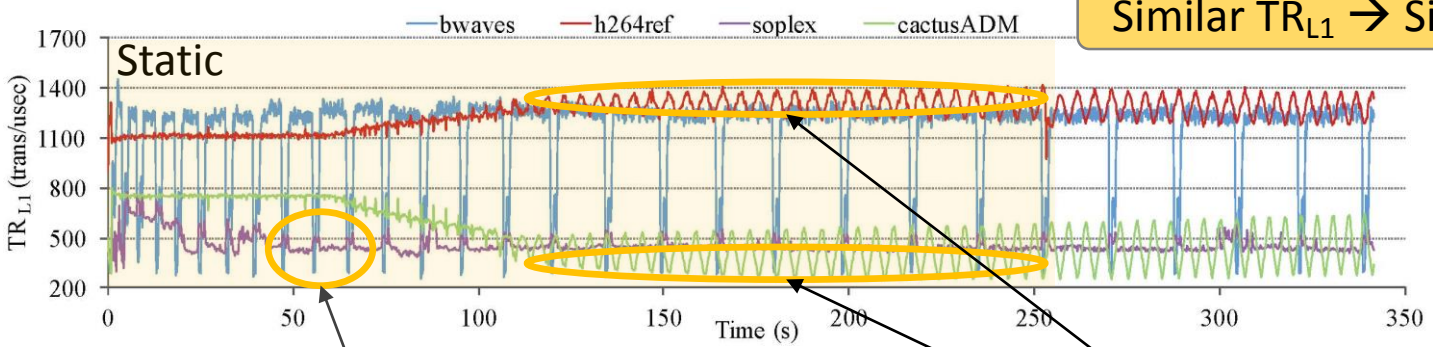


Similar TR_{L1} → Similar allocation

Experimental evaluation

Process allocation policies – Evolution of L1 bandwidth





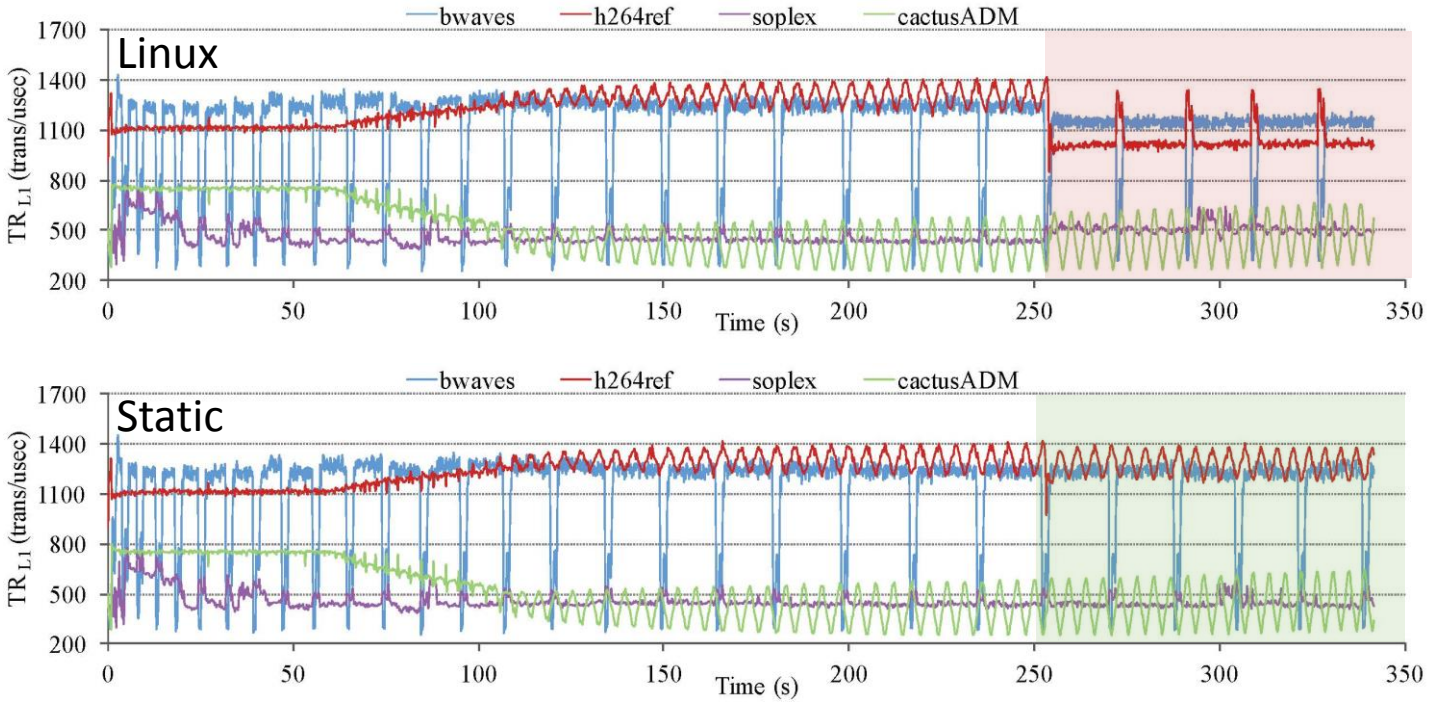
Similar TR_{L1} → Similar allocation

bwaves with *soplex*

h264ref with *cactusADM*

Experimental evaluation

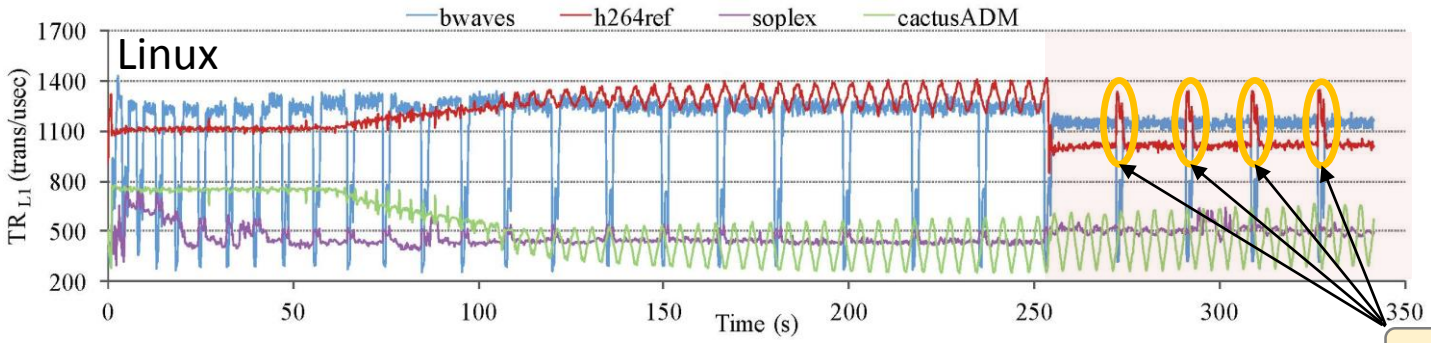
Process allocation policies – Evolution of L1 bandwidth

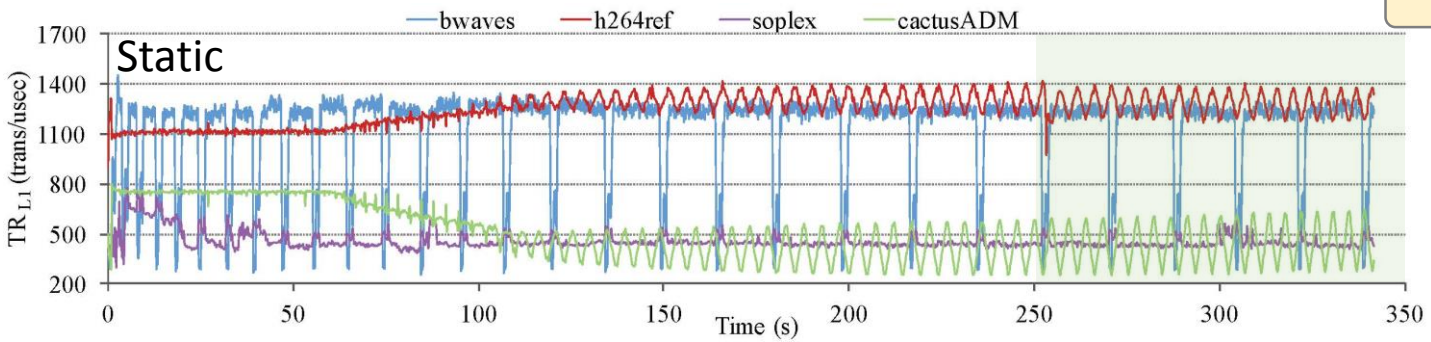


Linux changes the process allocation and achieves lower TR_{L1}

Experimental evaluation

Process allocation policies – Evolution of L1 bandwidth



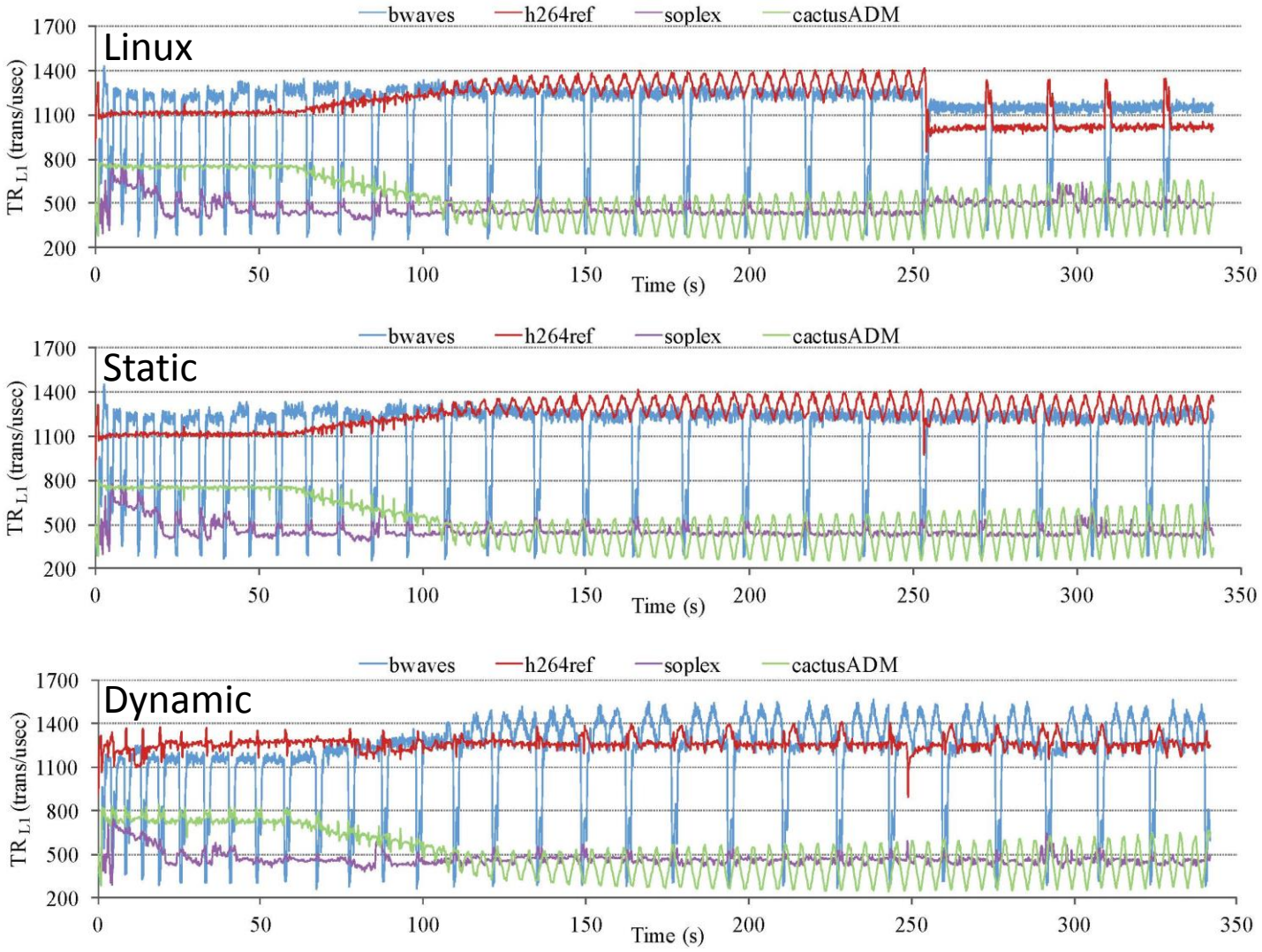


h264ref with *bwaves*

Linux changes the process allocation and achieves lower TR_{L1}

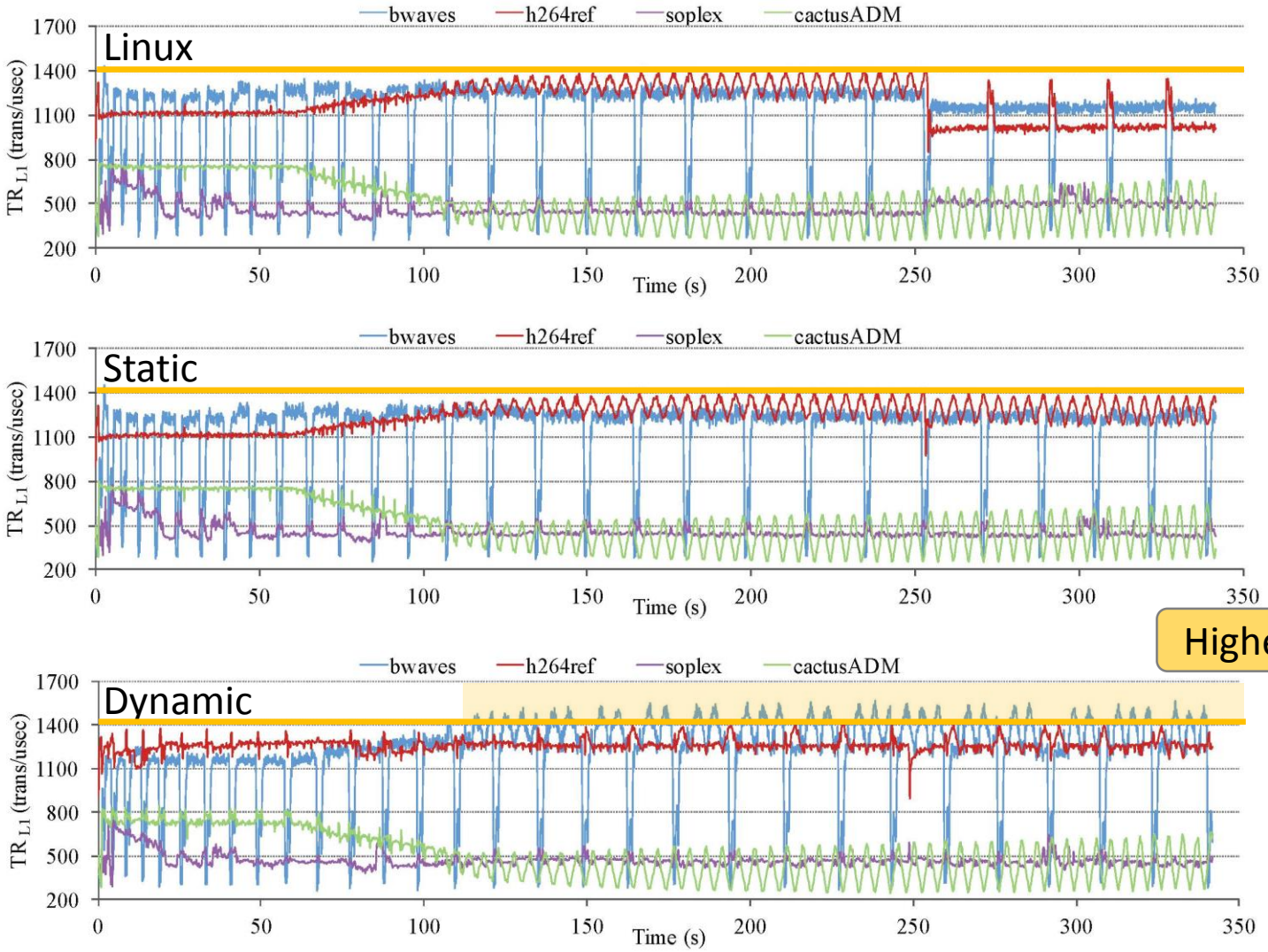
Experimental evaluation

Process allocation policies – Evolution of L1 bandwidth



Experimental evaluation

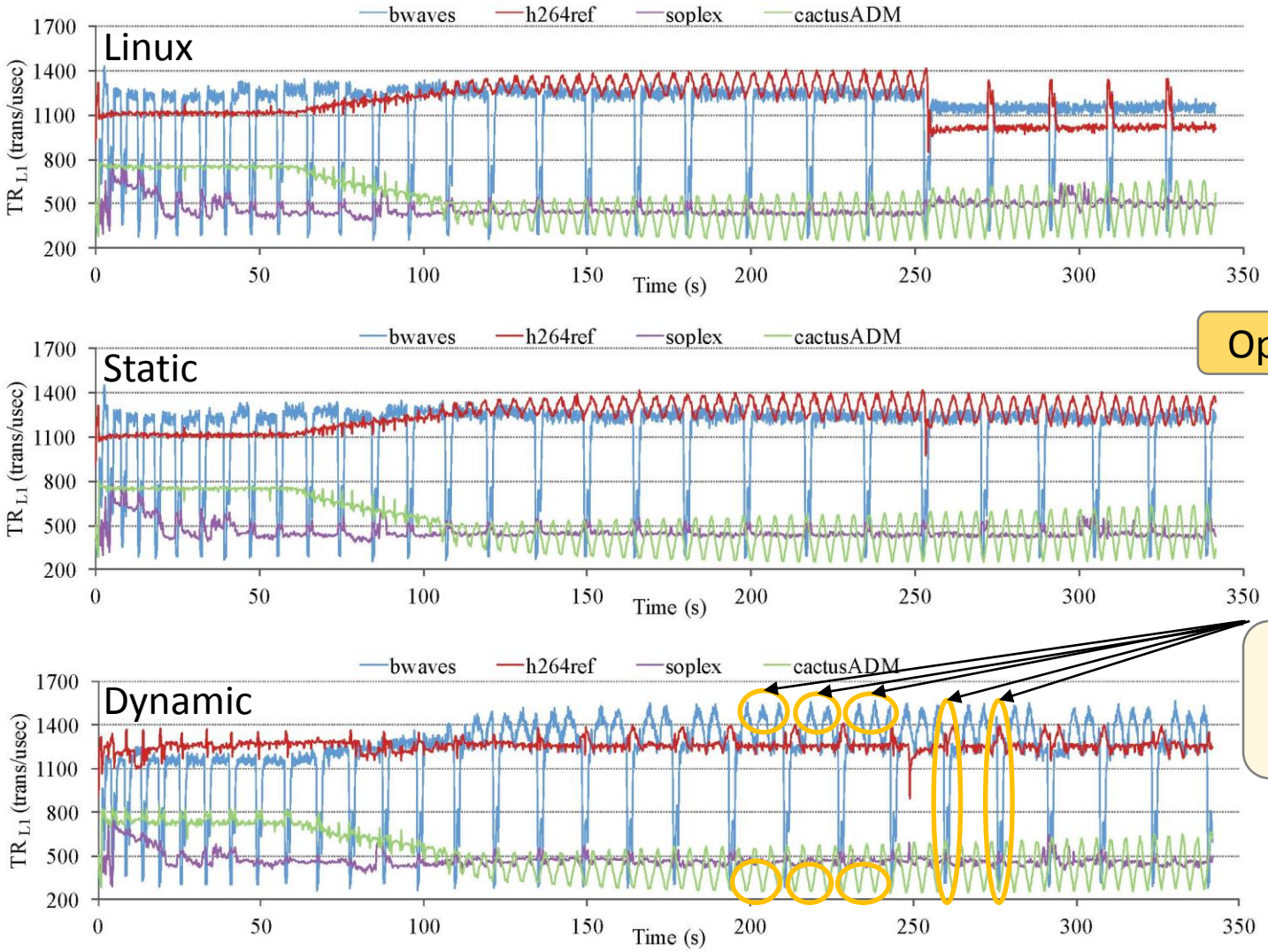
Process allocation policies – Evolution of L1 bandwidth



Higher TR_{L1} → Higher IPC

Experimental evaluation

Process allocation policies – Evolution of L1 bandwidth



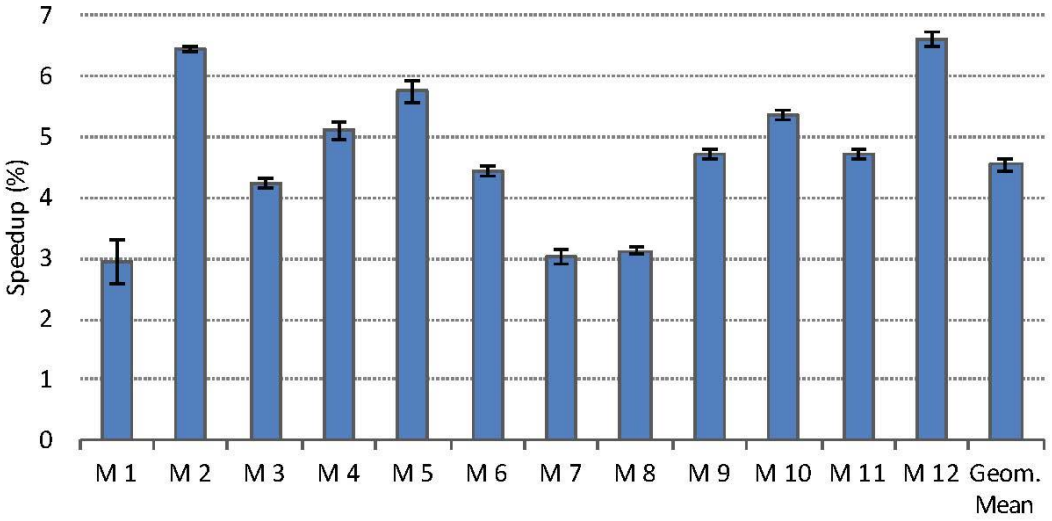
Optimal TR_{L1} balancing

Processes with highest and lowest TR_{L1} always on the same core

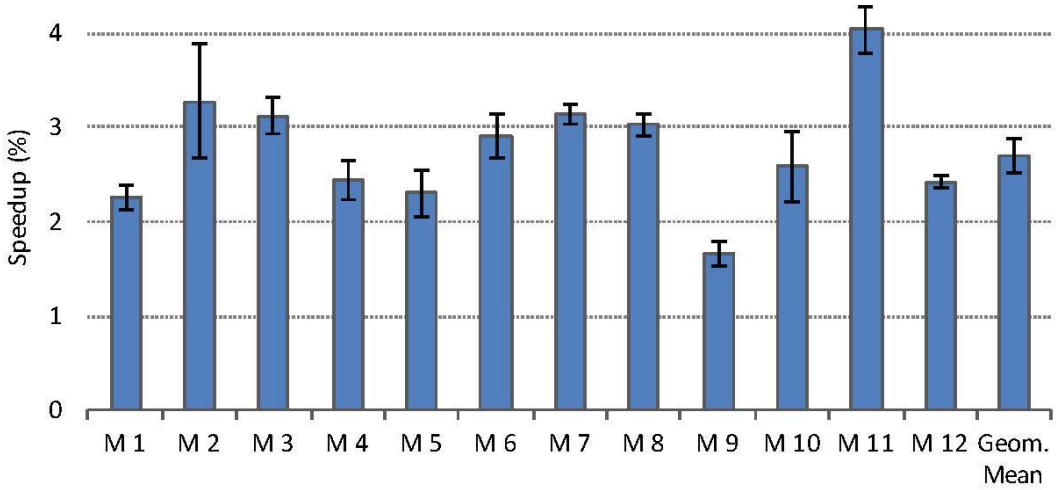
Experimental evaluation

SMT bandwidth-aware scheduler – Speedups

Speedup of the average IPC over Linux



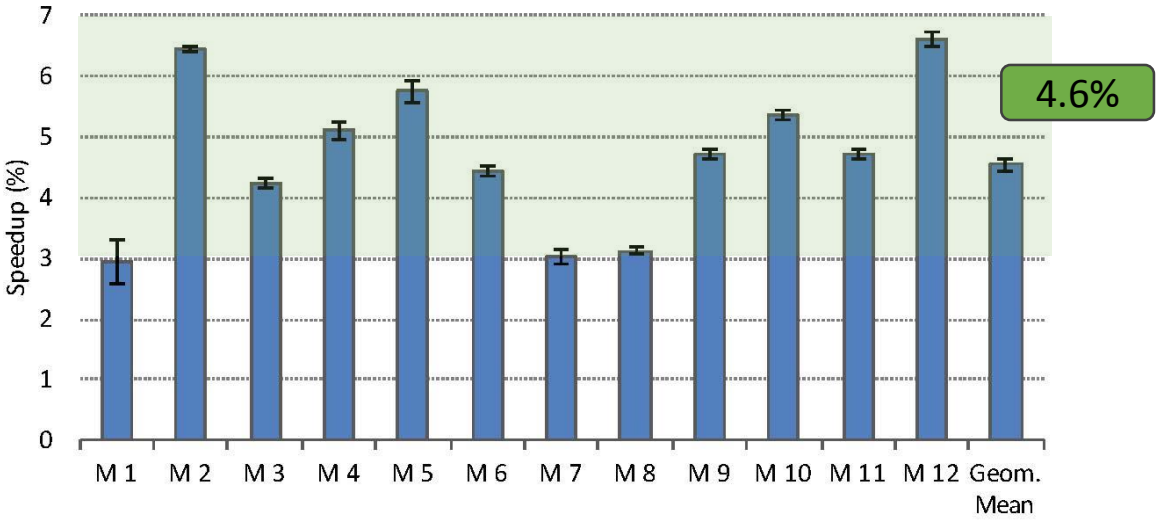
Speedup of the turnaround time over Linux



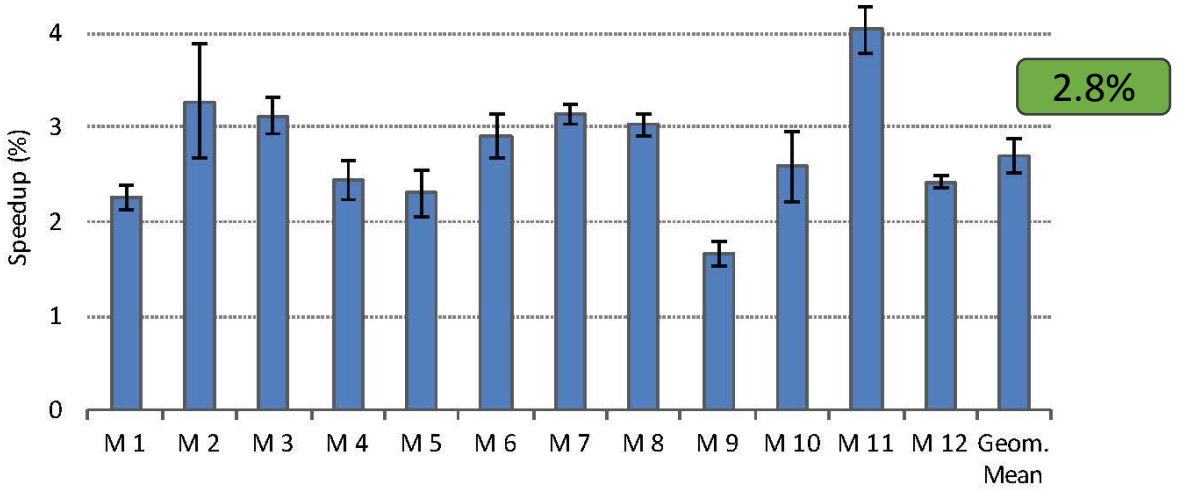
Experimental evaluation

SMT bandwidth-aware scheduler – Speedups

Speedup of the average IPC over Linux



Speedup of the turnaround time over Linux



Outline

I. Bandwidth-Aware Scheduling on Multicores

II. Bandwidth-Aware Scheduling on SMT Multicores

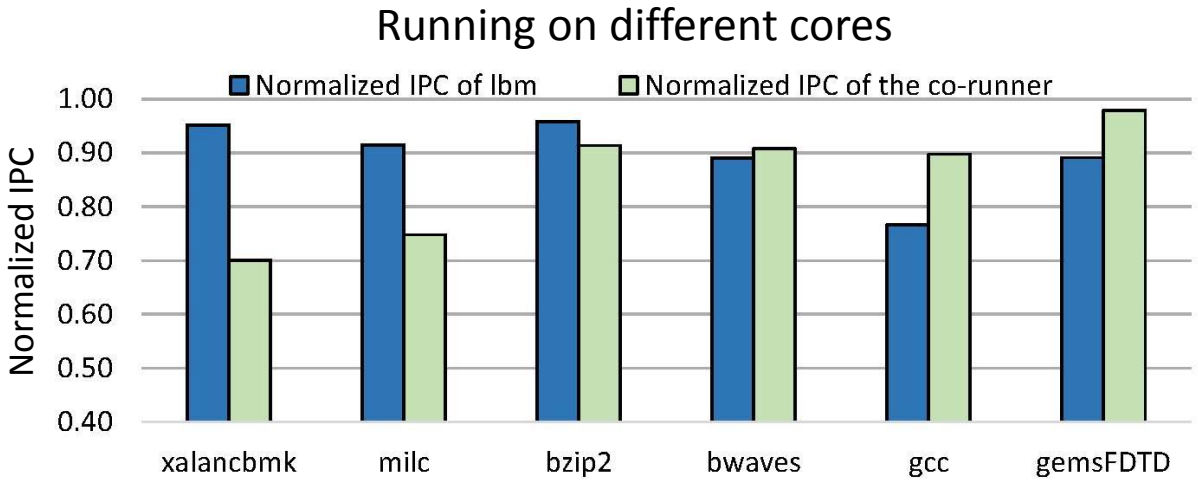
III. Progress-Aware Scheduling on SMT Multicores

1. Motivation
2. Estimating progress
3. Progress-aware Fair scheduler
4. Experimental evaluation
5. Progress-aware Perf&Fair scheduler
6. Experimental evaluation

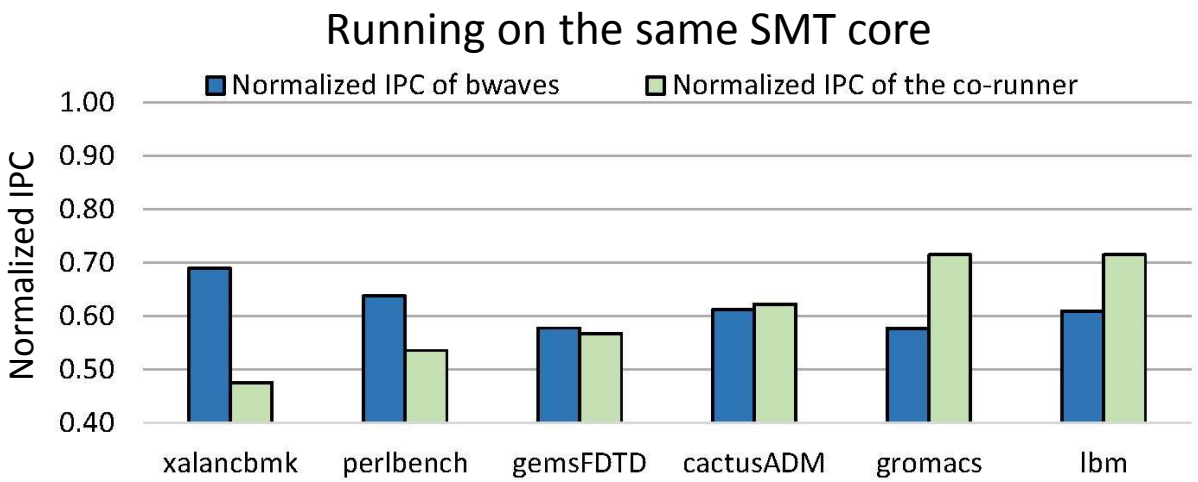
IV. Symbiotic Job Scheduling on the IBM POWER8

Motivation

Are current SMT multicores unfair?



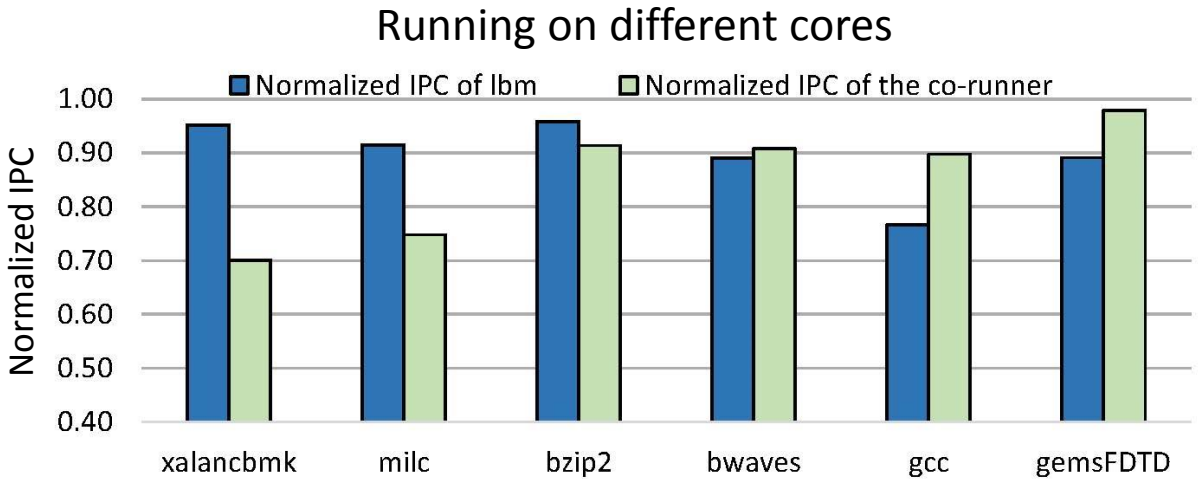
Shared resources:
main memory and LLC

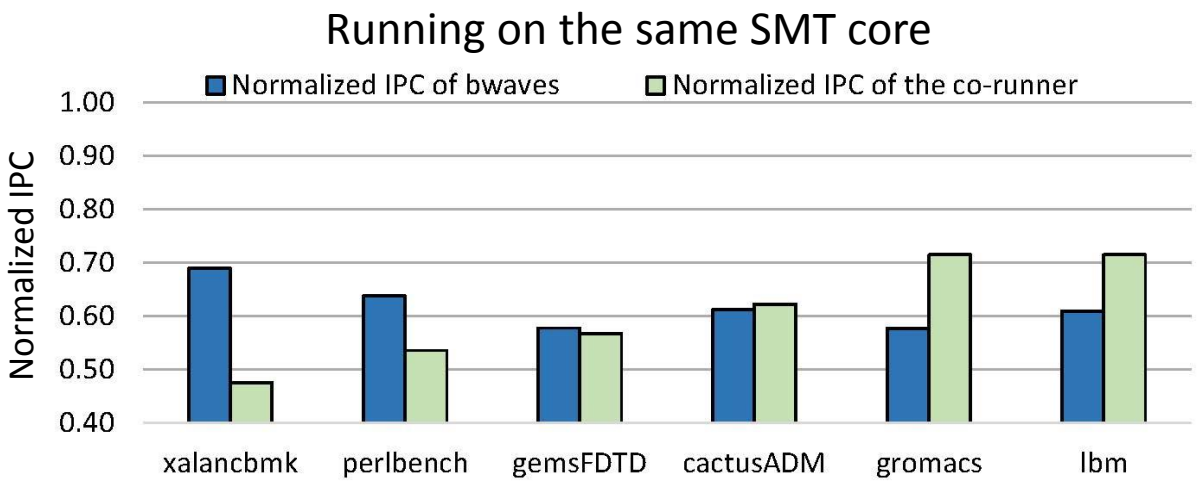


Shared resources:
main memory, LLC and
intra-core shared resources

Motivation

Are current SMT multicores unfair?

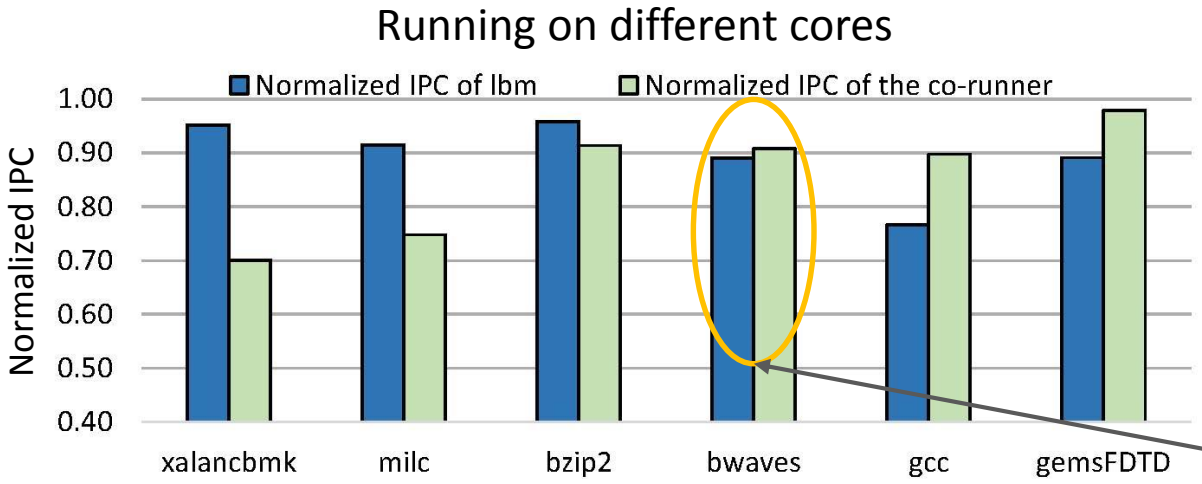




Different progress depending on the co-runner

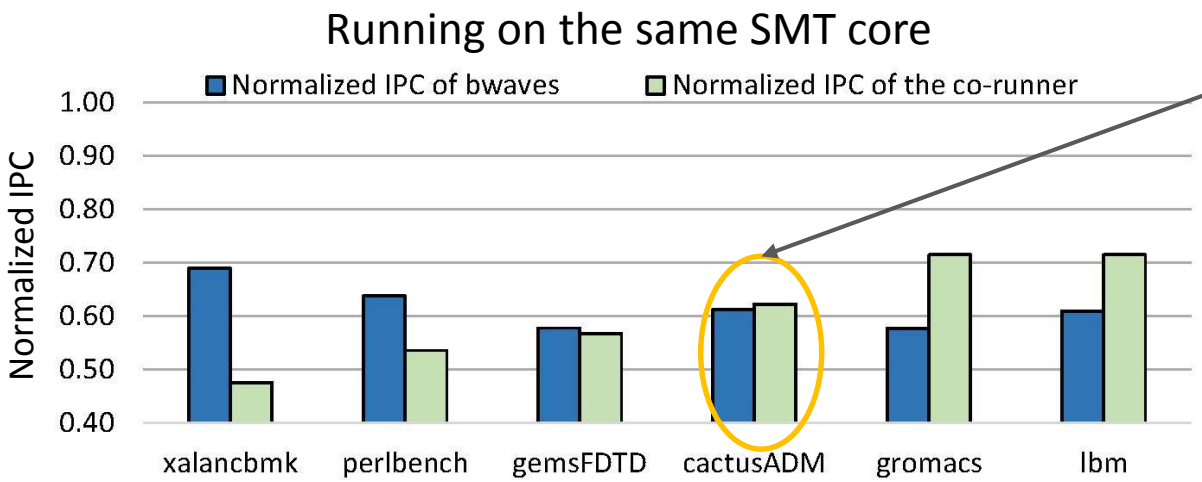
Motivation

Are current SMT multicores unfair?



$$Unf = \frac{Max\ slowdown}{Min\ slowdown}$$

Couples that run fairly

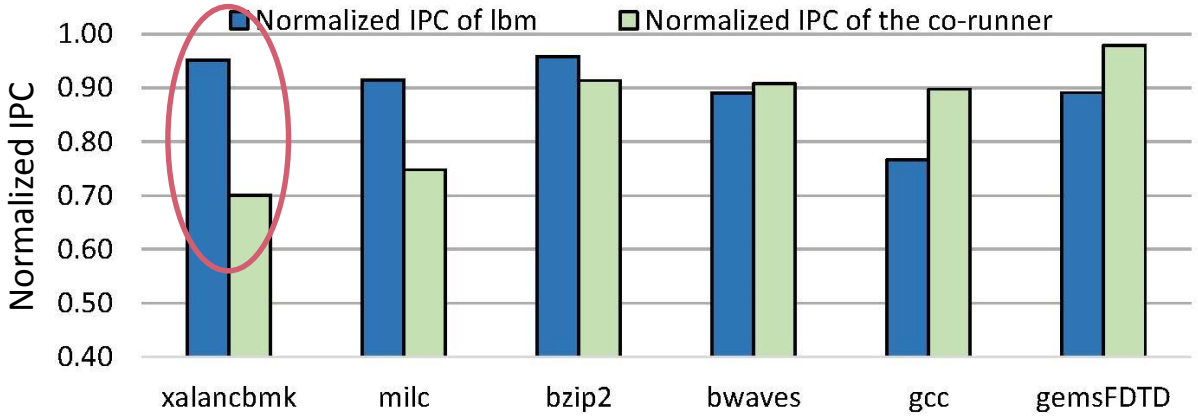


Motivation

Are current SMT multicores unfair?

Unf. = 1.36

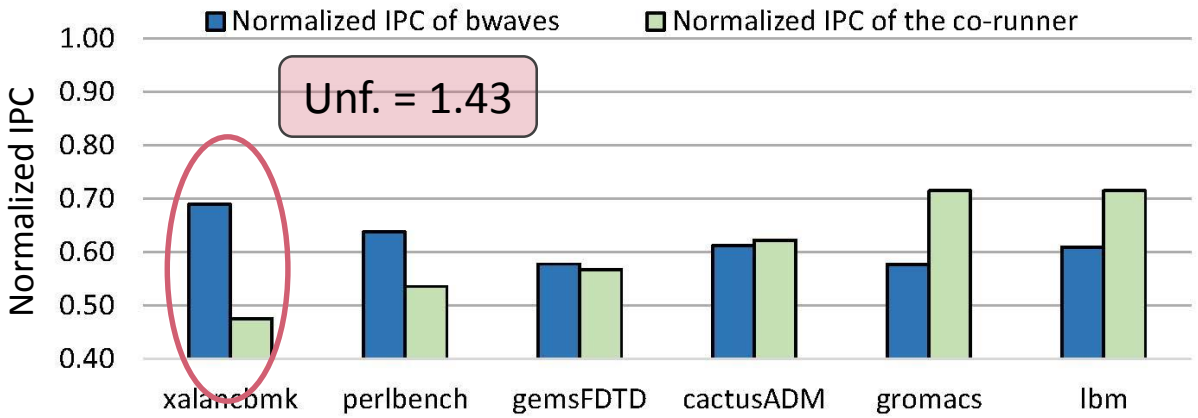
Running on different cores



$$Unf = \frac{Max\ slowdown}{Min\ slowdown}$$

Couples that run unfairly

Running on the same SMT core



Estimating progress

Progress can be estimated as:

$$Progress = \sum_{i=0}^{Q} \frac{IPC^{i}_{co-runners}}{IPC^{i}_{alone}}$$

Estimating progress

Progress can be estimated as:

$$Progress = \sum_{i=0}^{Q} \frac{IPC^{i}_{co-runners}}{IPC^{i}_{alone}}$$

IPC_{alone}

- Estimated in a low-contention schedule
  - Avoids intra-core interference: processes allocated alone on an SMT core
  - Minimize inter-core interference: appropriate co-runners

Estimating progress

Progress can be estimated as:

$$Progress = \sum_{i=0}^{Q} \frac{IPC^{i}_{co-runners}}{IPC^{i}_{alone}}$$

IPC_{alone}

- Estimated in a low-contention schedule
- IPC in the low-contention schedule
  - Assumed equal to IPC alone
  - Assumed valid for the *n* following quanta

Estimating progress

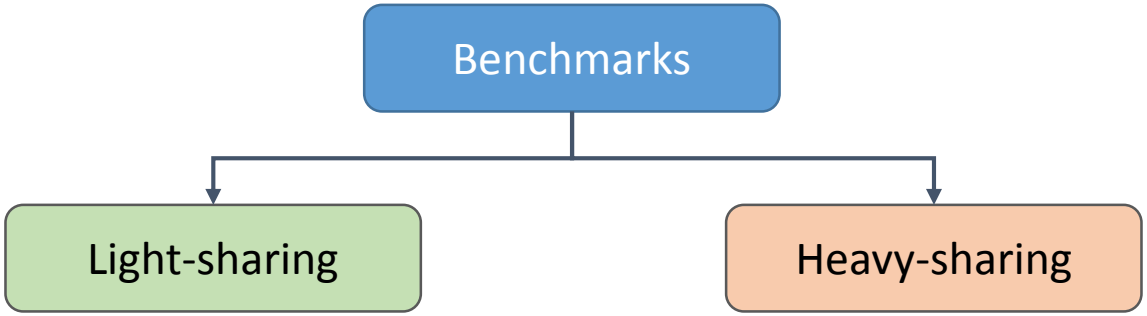
Progress can be estimated as:

$$Progress = \sum_{i=0}^{Q} \frac{IPC^{i}_{co-runners}}{IPC^{i}_{alone}}$$

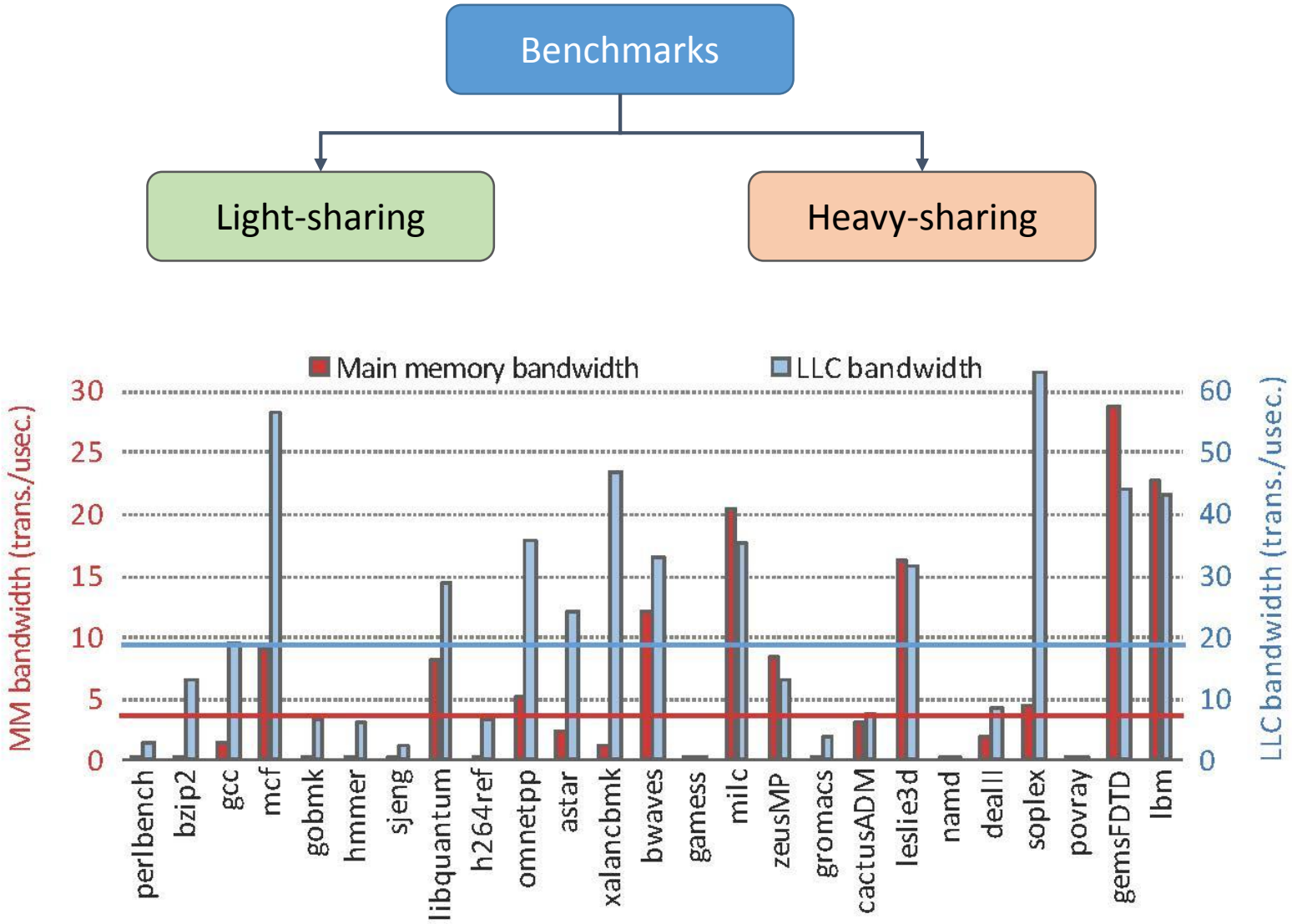
IPC_{alone}

- Estimated in a low-contention schedule
- IPC in the low-contention schedule
- Inaccuracy estimating IPC_{alone}
  - Standalone IPC assumed valid for a too long interval
  - Process interference in the low-contention schedule is high

Estimating progress

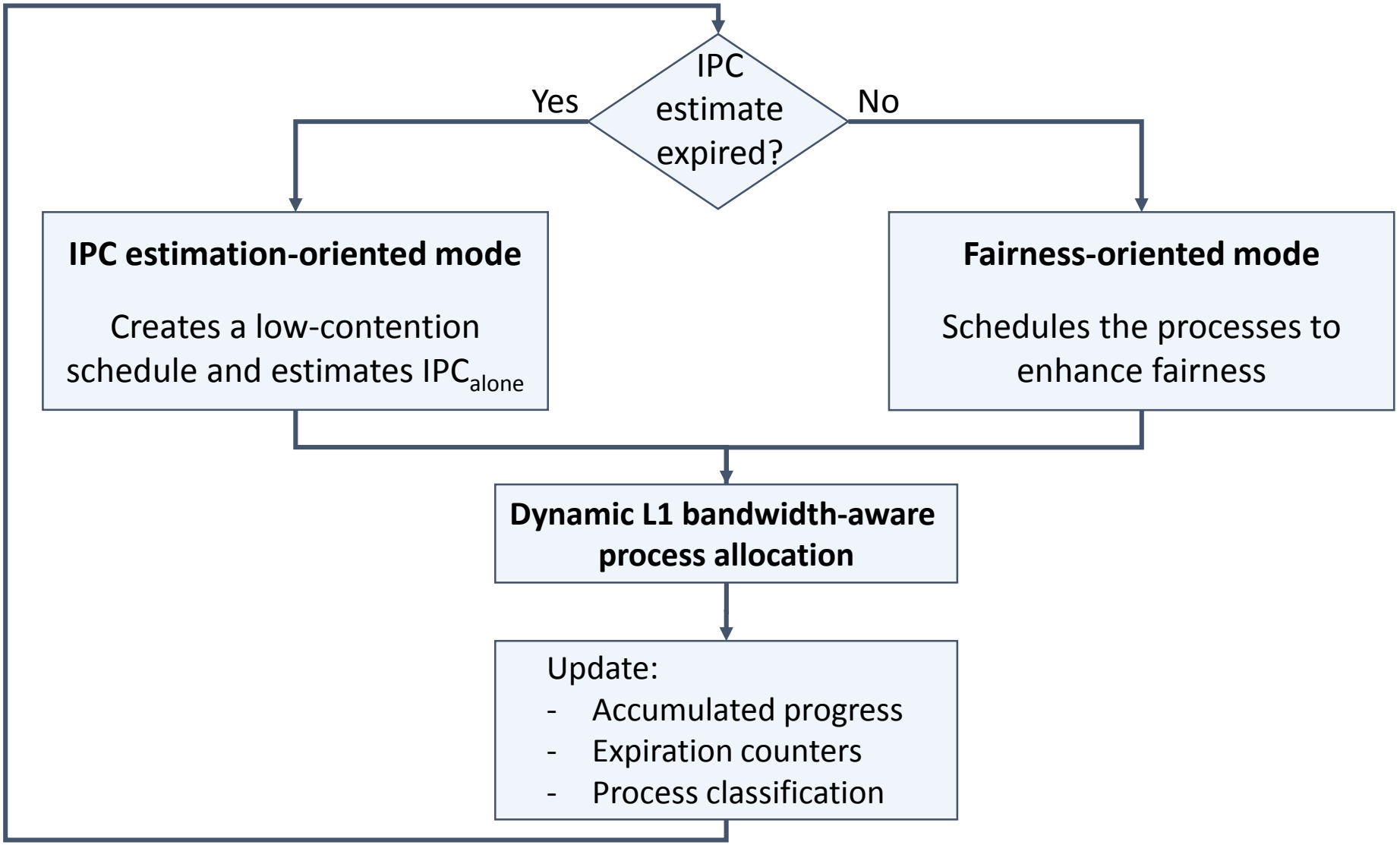


Estimating progress



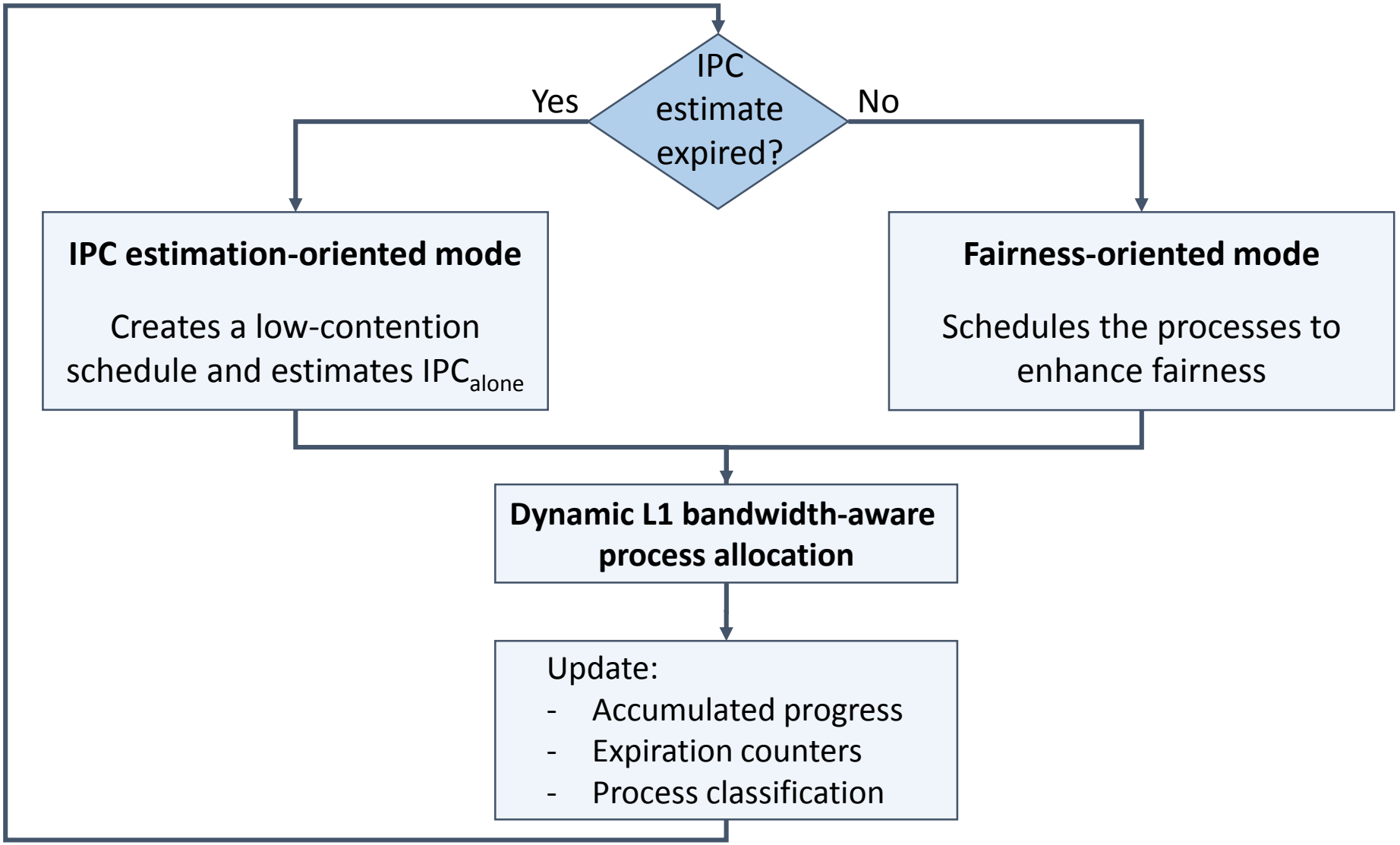
Progress-aware Fair scheduler

Main steps



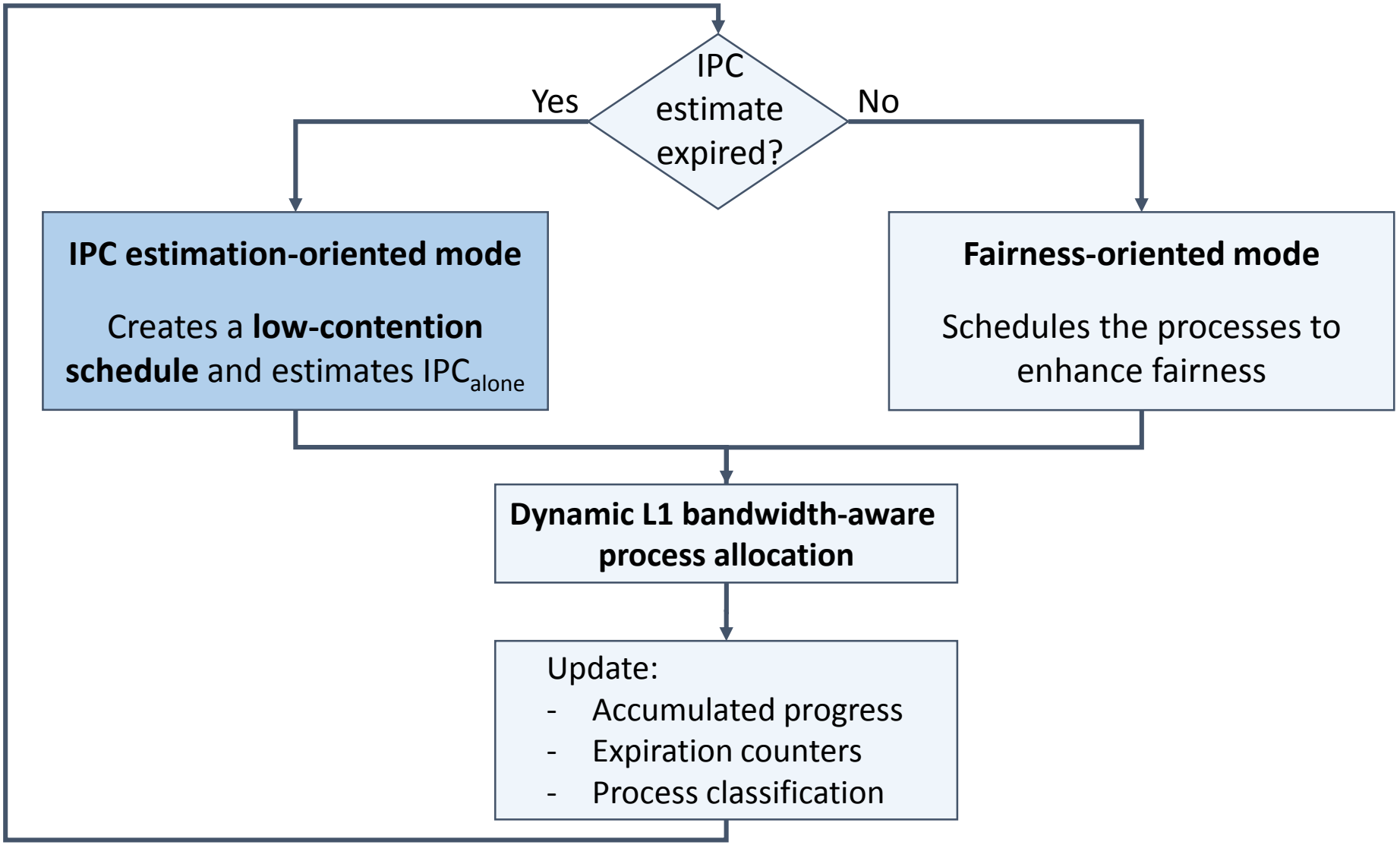
Progress-aware Fair scheduler

Main steps



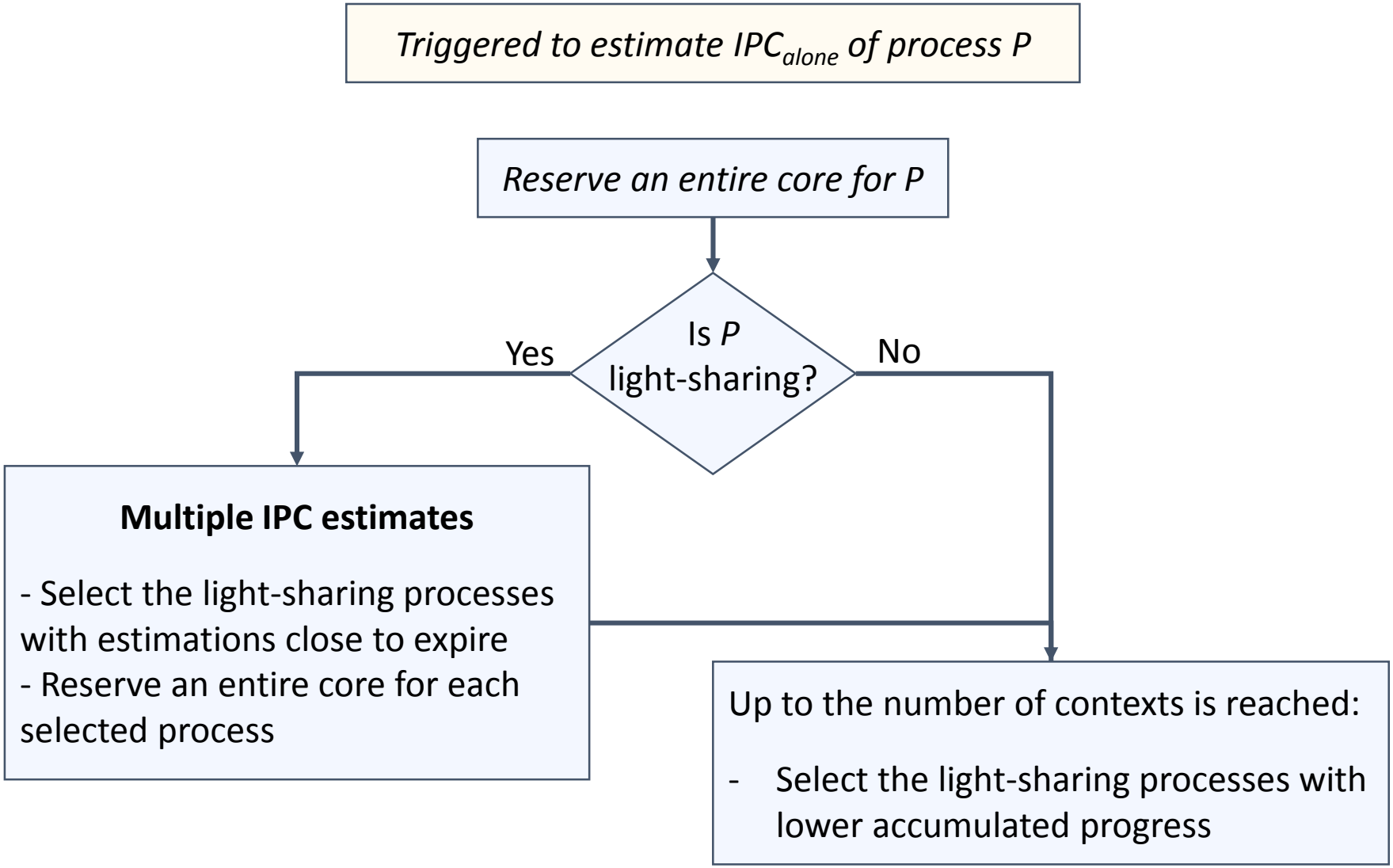
Progress-aware Fair scheduler

Main steps



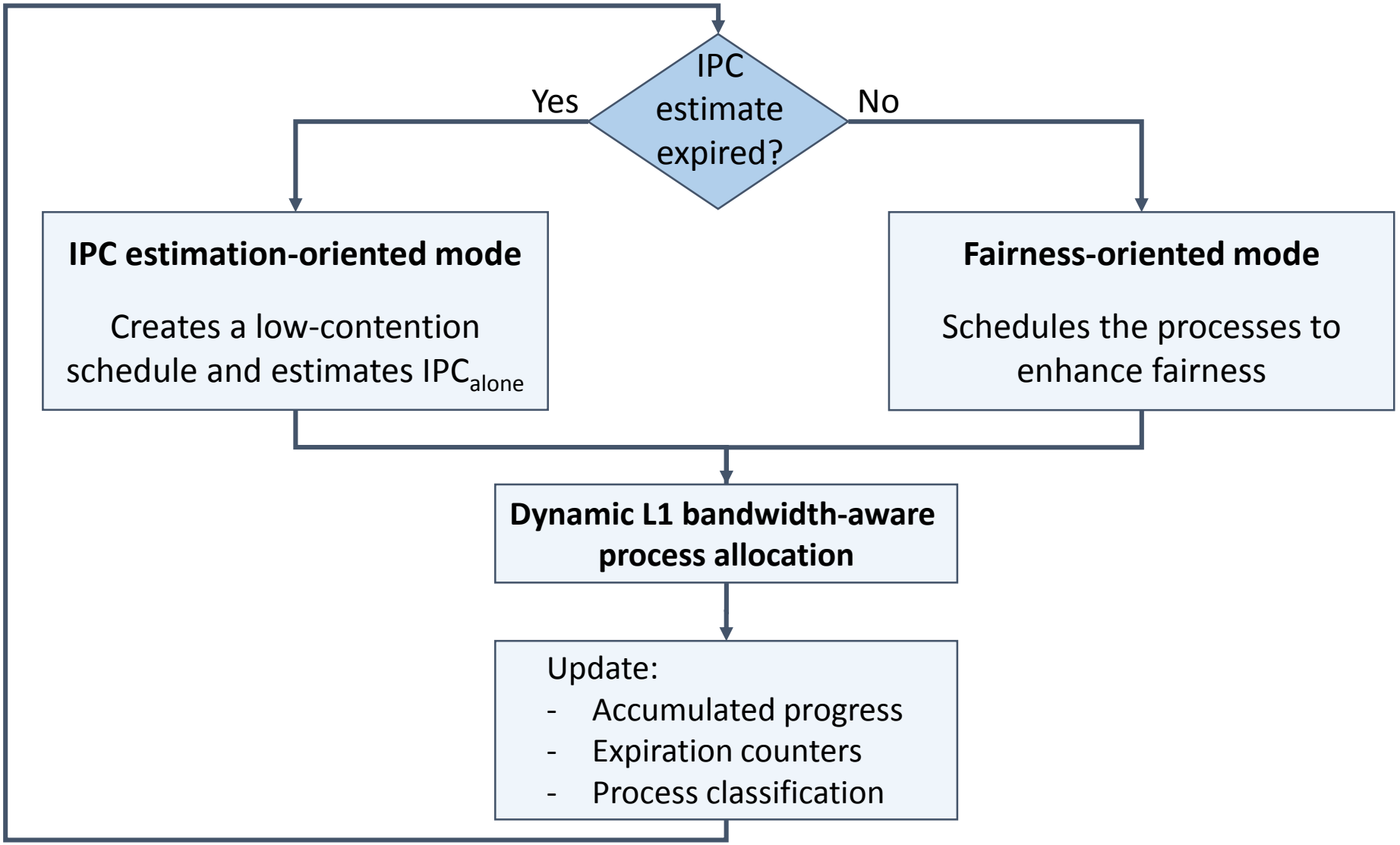
Progress-aware Fair scheduler

IPC estimation-oriented mode



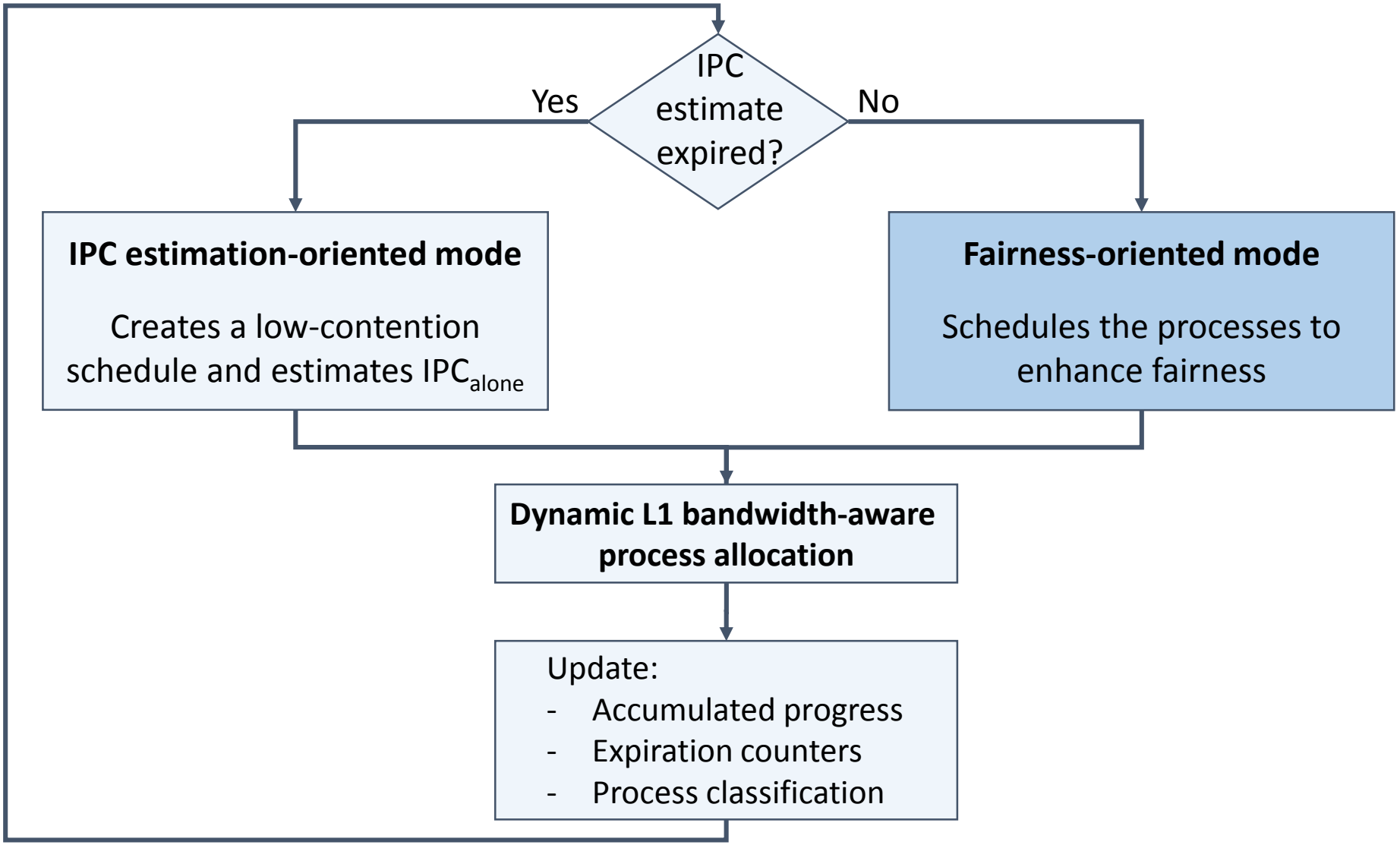
Progress-aware Fair scheduler

Main steps



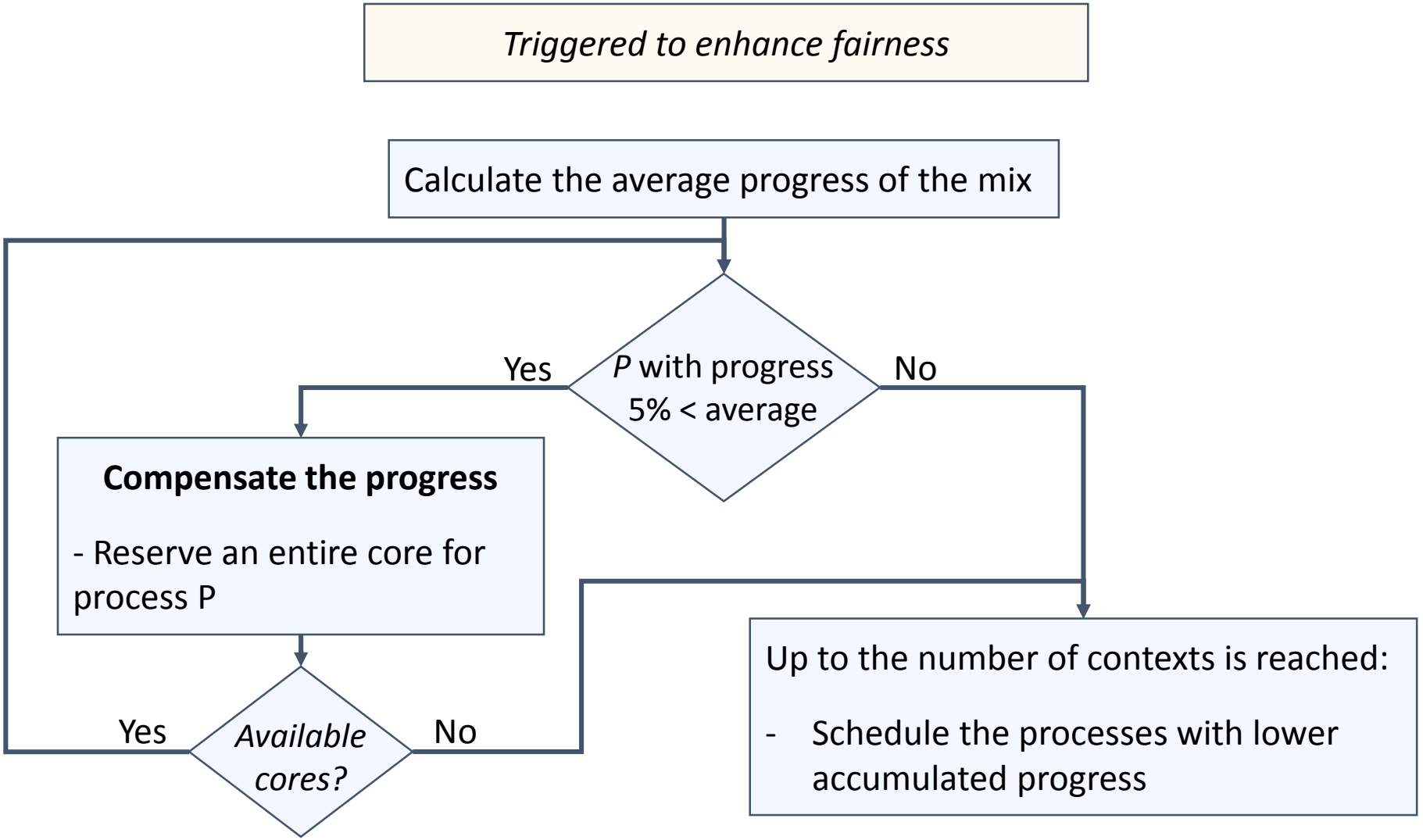
Progress-aware Fair scheduler

Main steps



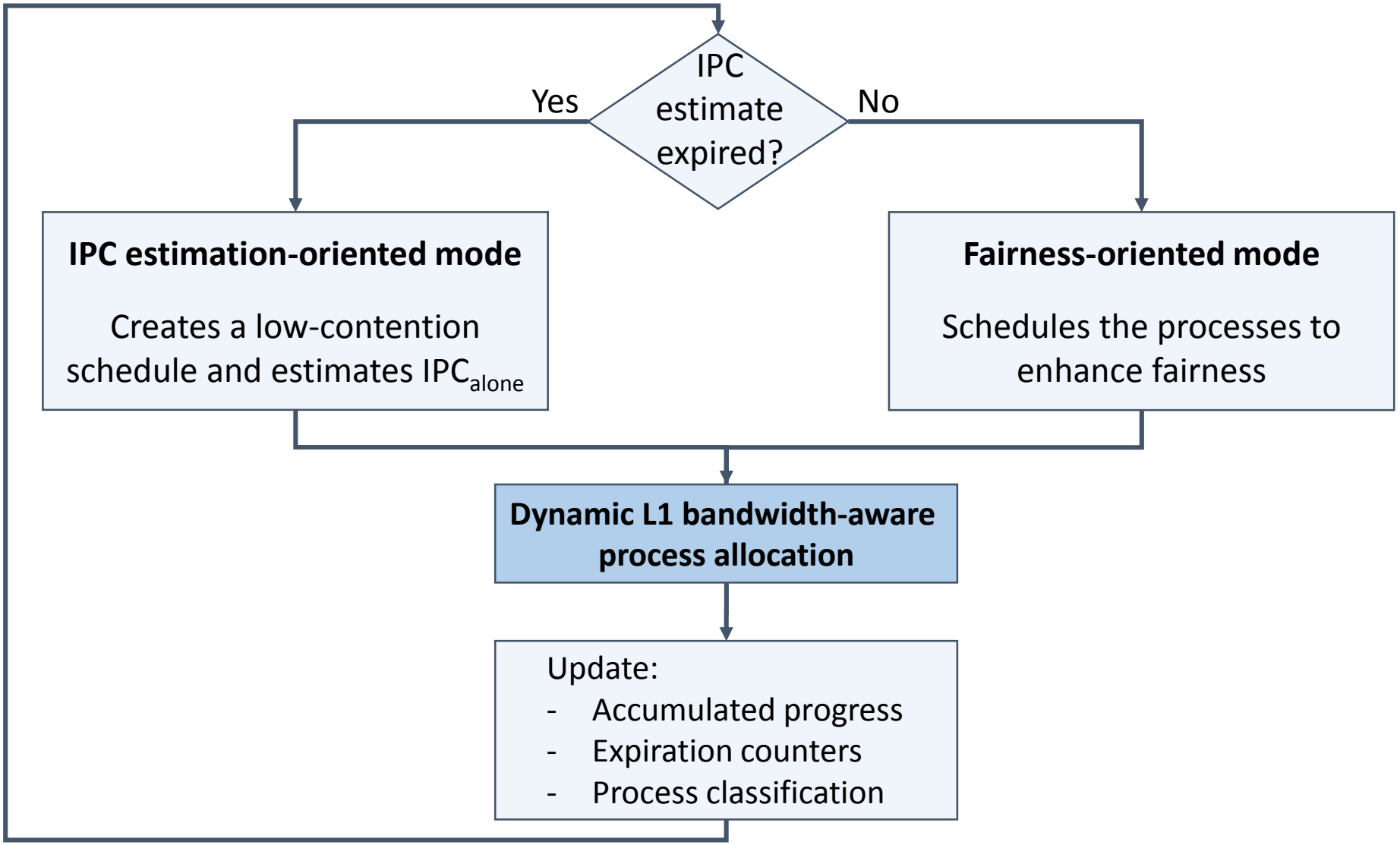
Progress-aware Fair scheduler

Fairness-oriented mode



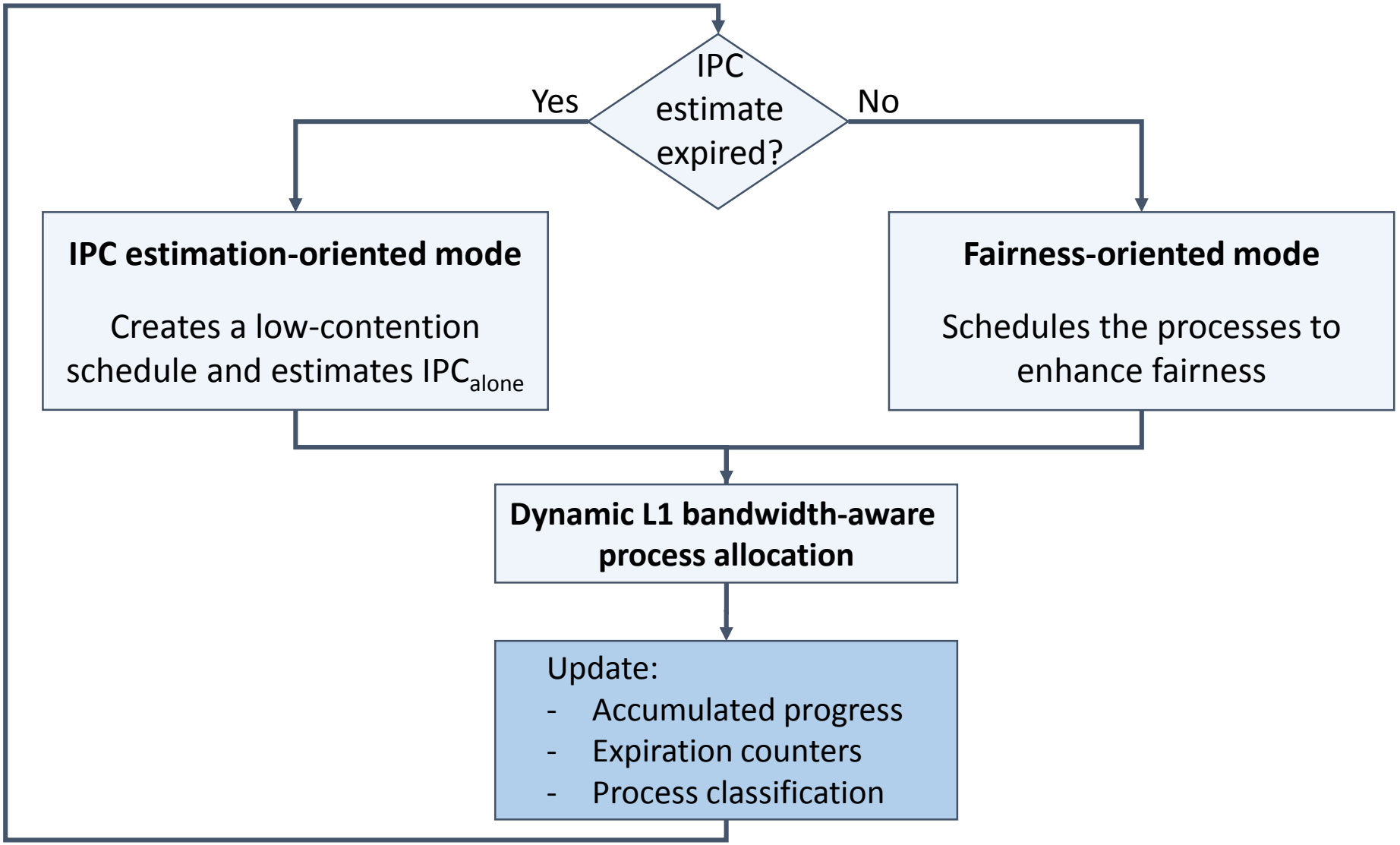
Progress-aware Fair scheduler

Main steps



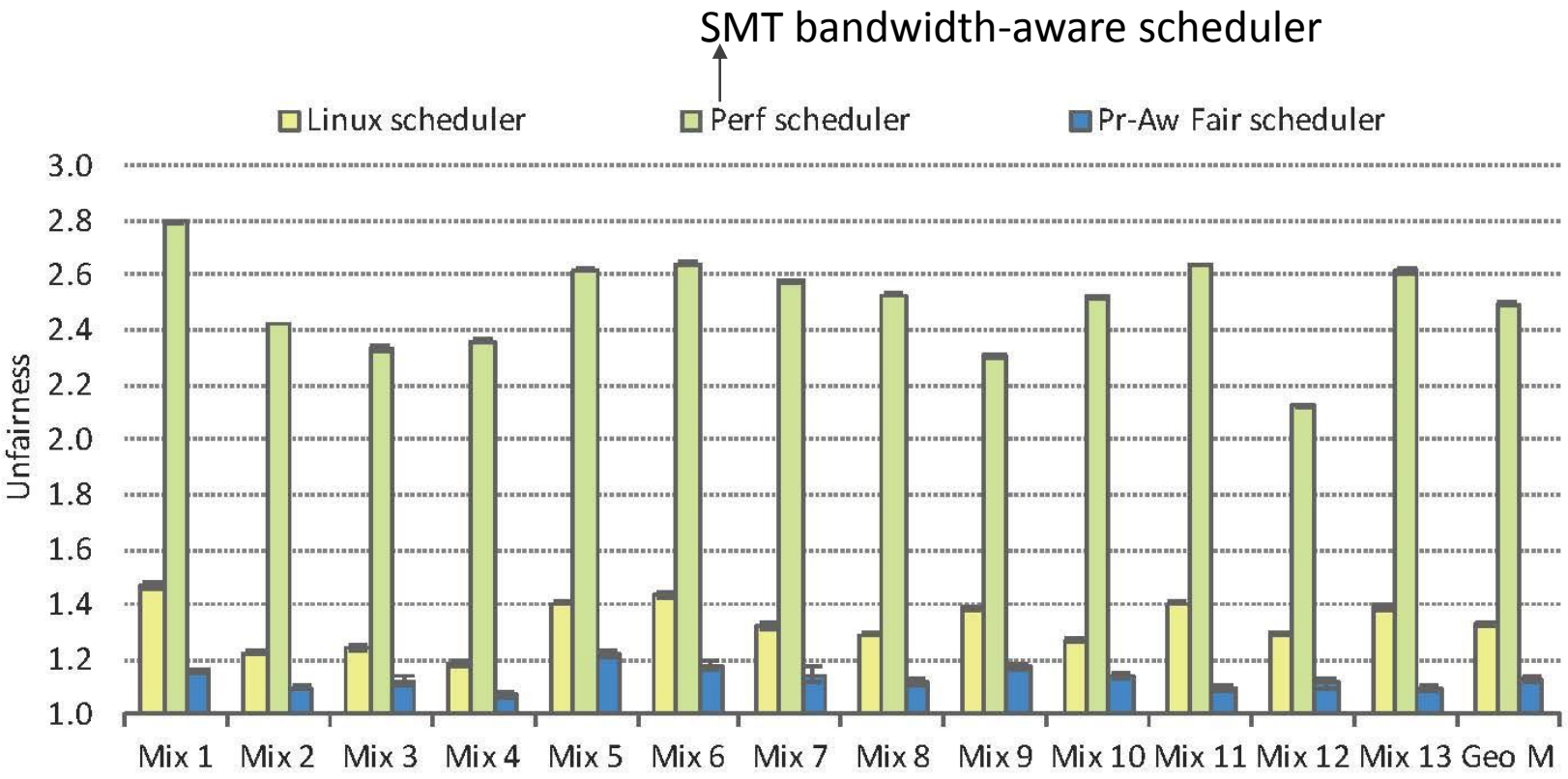
Progress-aware Fair scheduler

Main steps



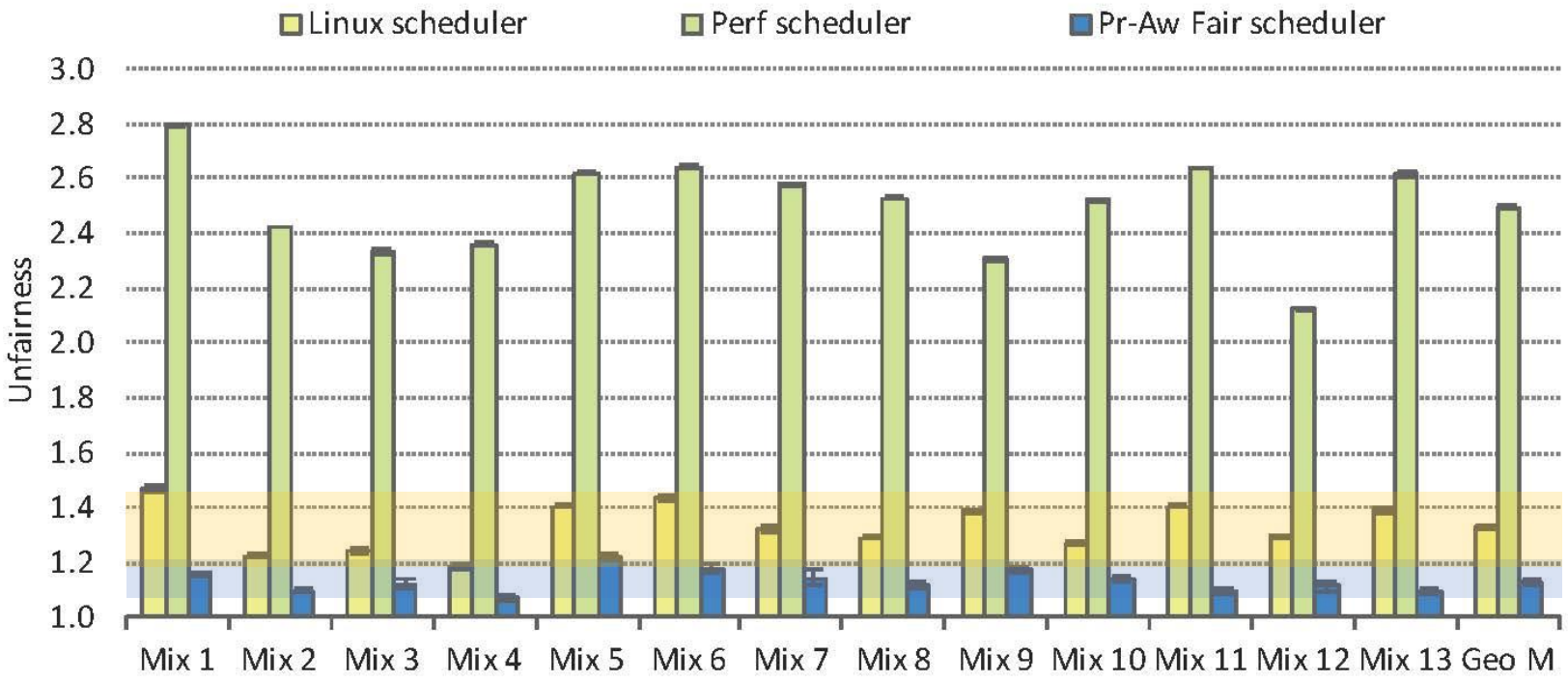
Experimental evaluation

Fairness (I)



Experimental evaluation

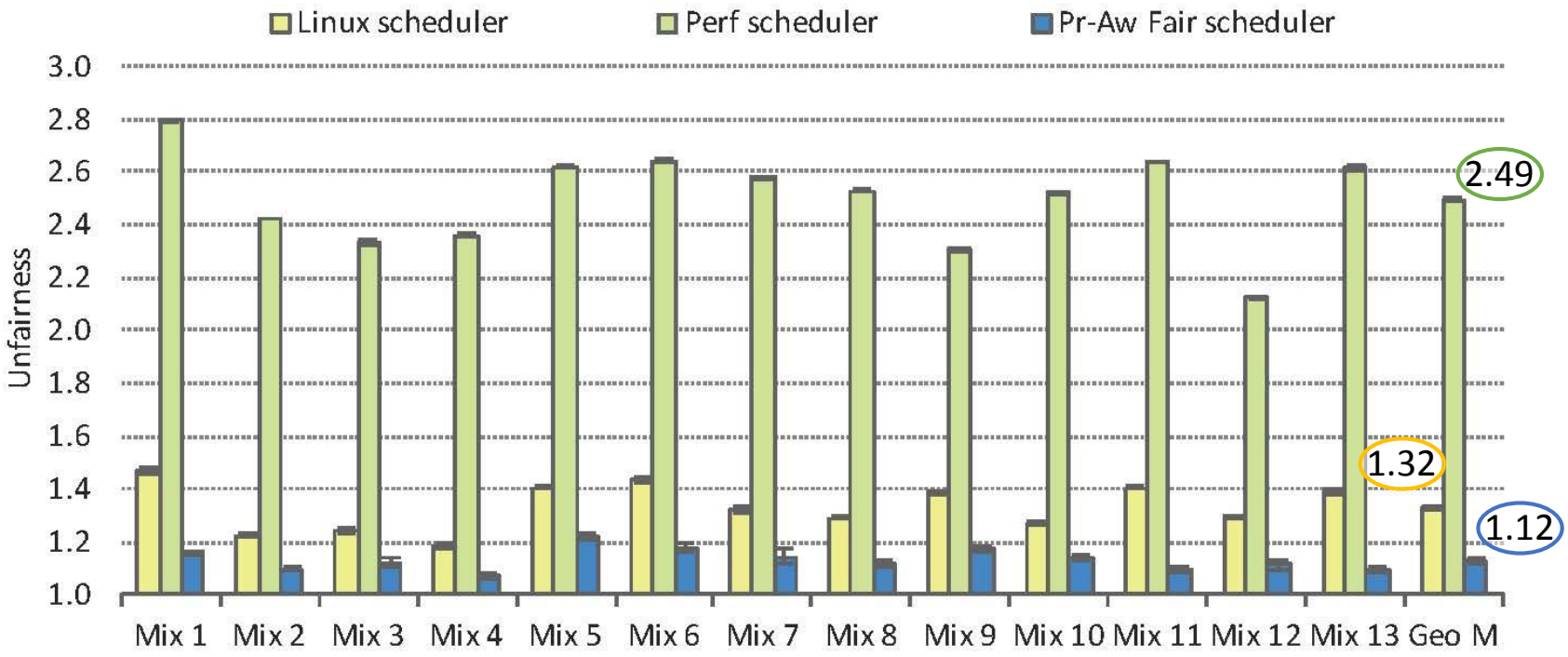
Fairness (I)



Fair scheduler greatly reduces unfairness

Experimental evaluation

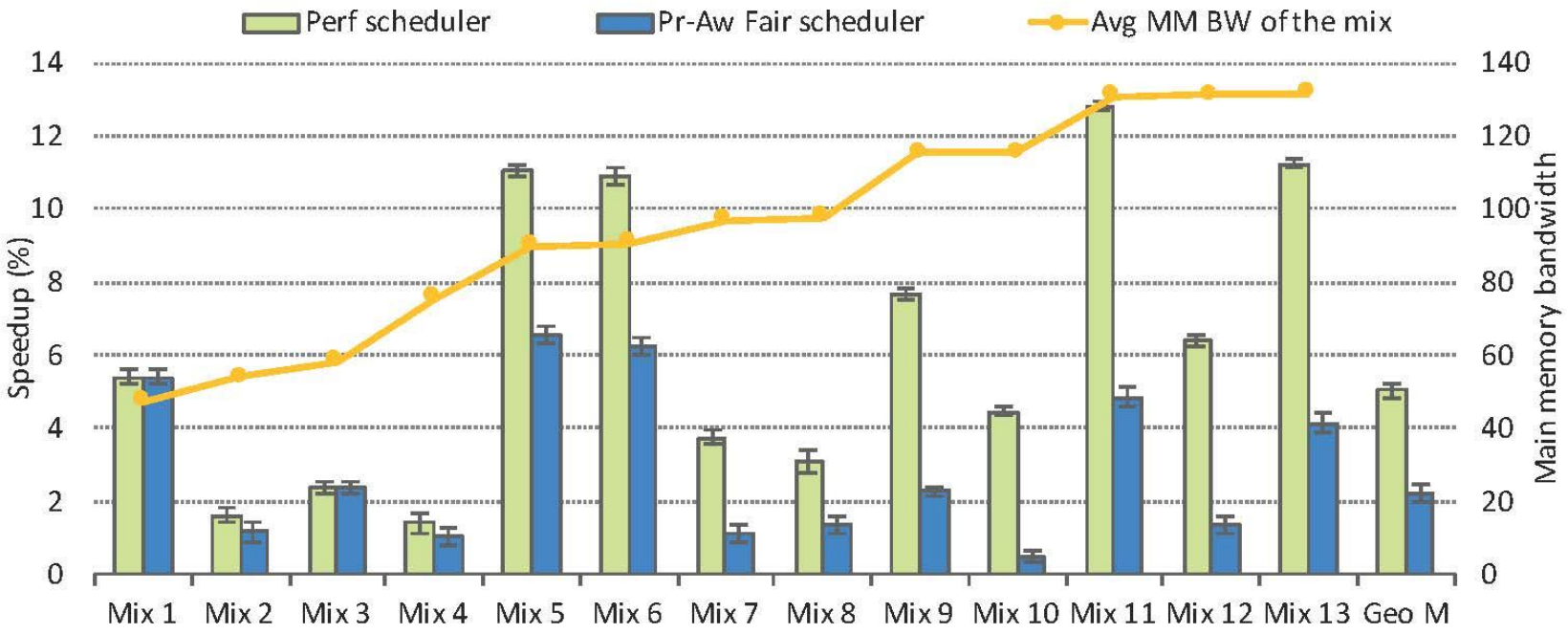
Fairness (I)



Fair reduces Linux unfairness to a third

Experimental evaluation

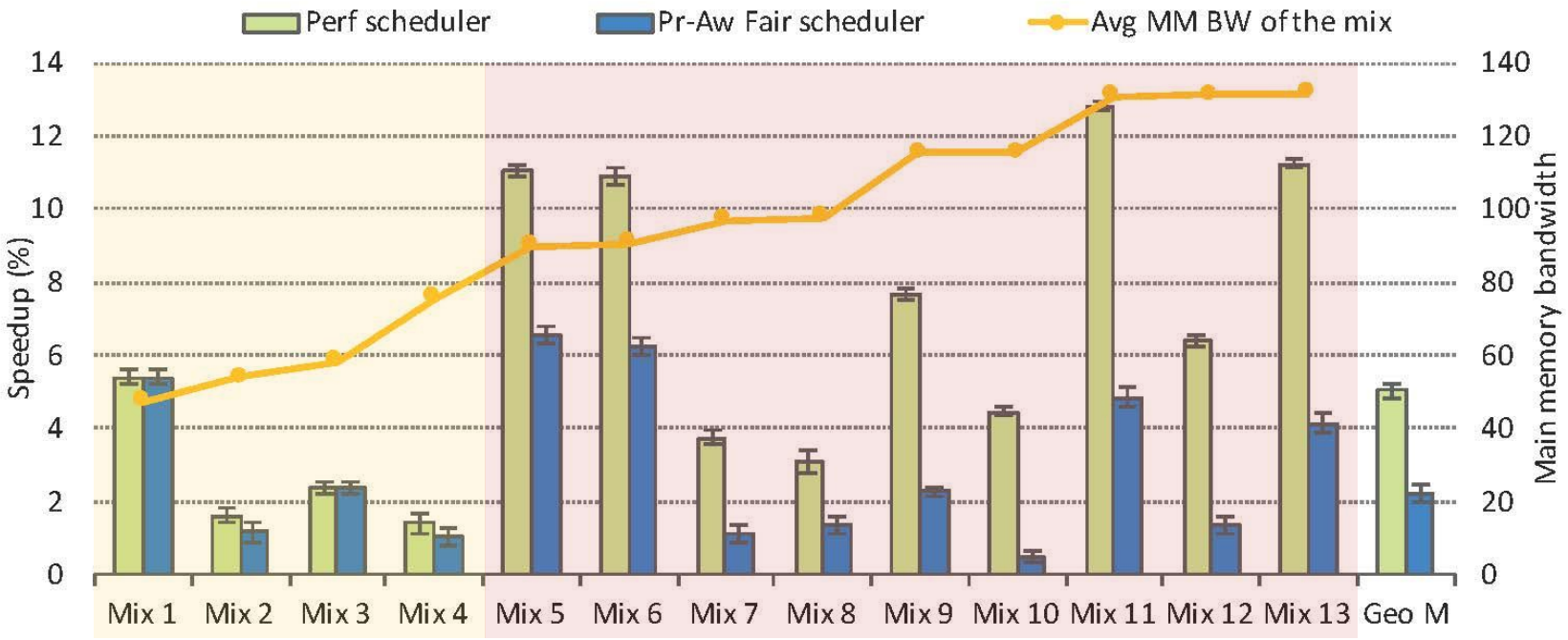
Speedup of the turnaround time over Linux (I)



- The yellow line shows the average main memory bandwidth of the mixes.

Experimental evaluation

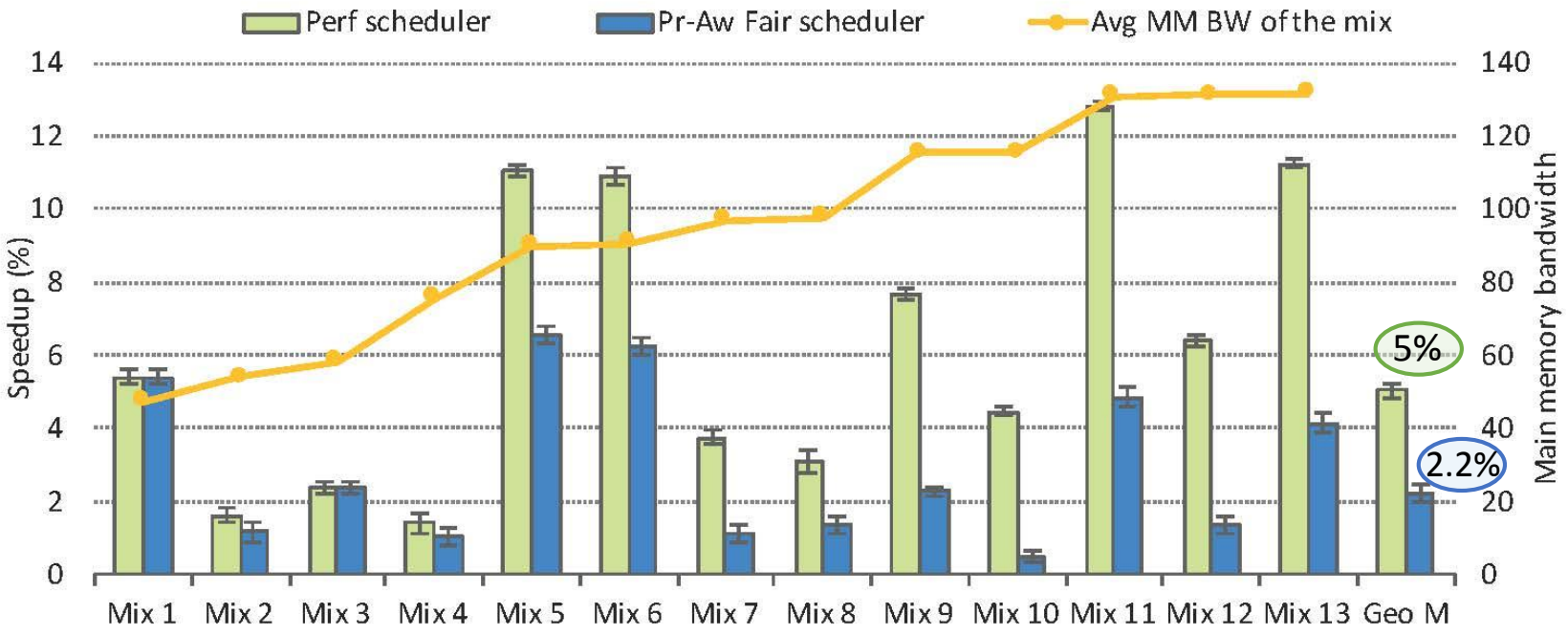
Speedup of the turnaround time over Linux (I)



- Fair scheduler performance:
  - Similar to Perf when bandwidth contention is not too high
  - Lower than Perf for higher bandwidth contention

Experimental evaluation

Speedup of the turnaround time over Linux (I)

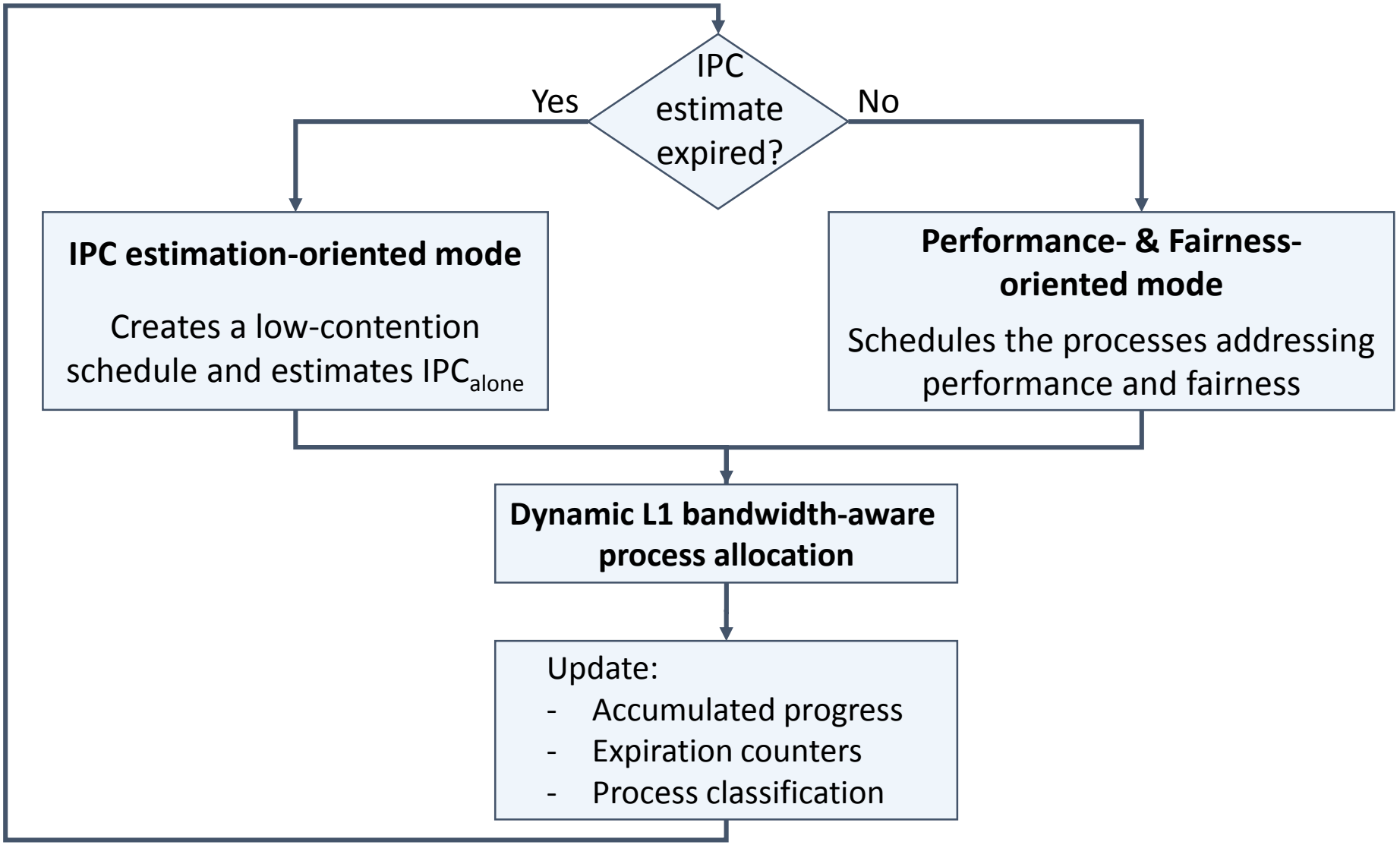


Despite focusing on fairness *Fair* improves Linux performance

- Fair scheduler performance:
  - Similar to Perf when bandwidth contention is not too high
  - Lower than Perf for higher bandwidth contention

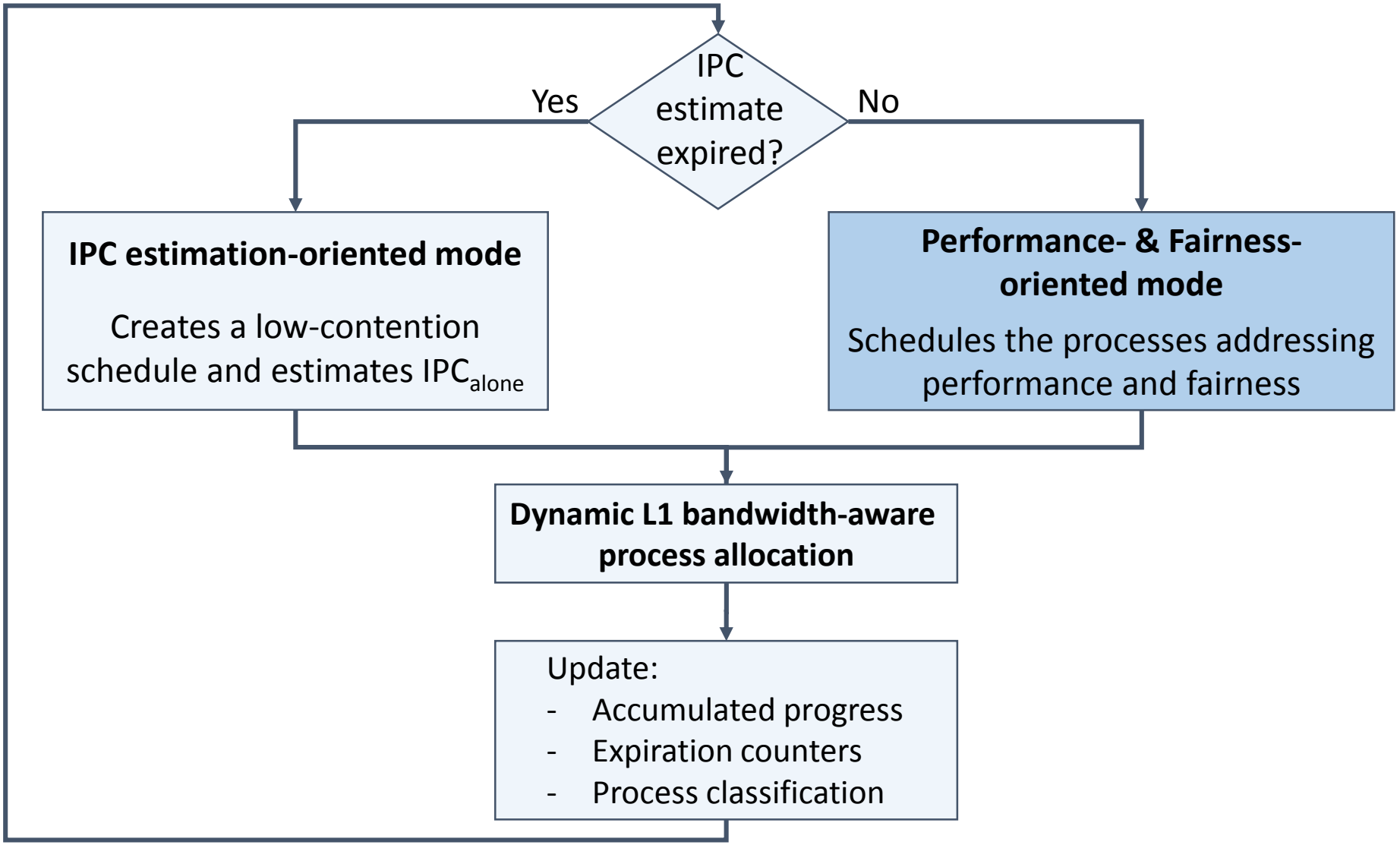
Progress-aware Perf&Fair scheduler

Main steps



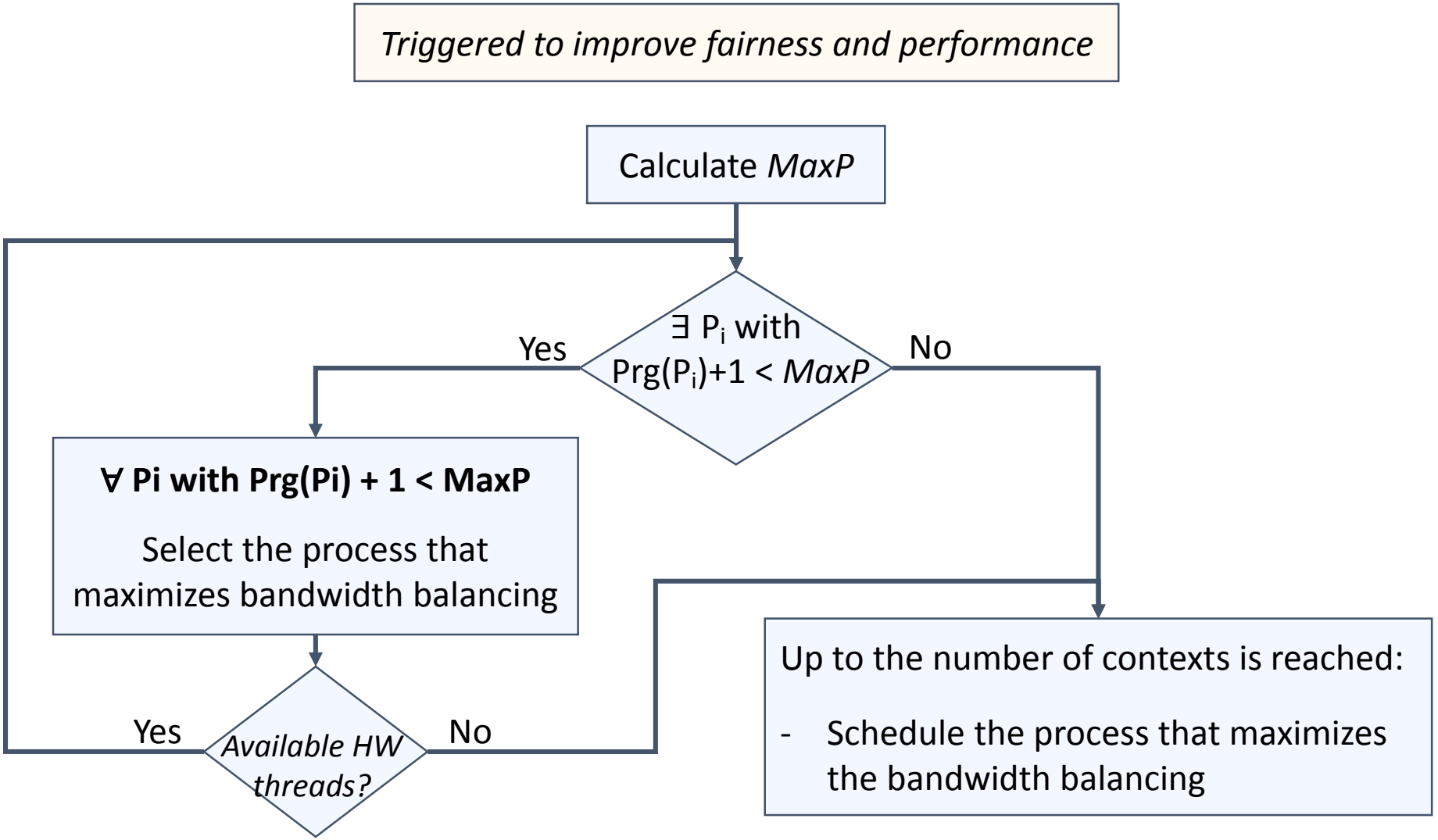
Progress-aware Perf&Fair scheduler

Main steps



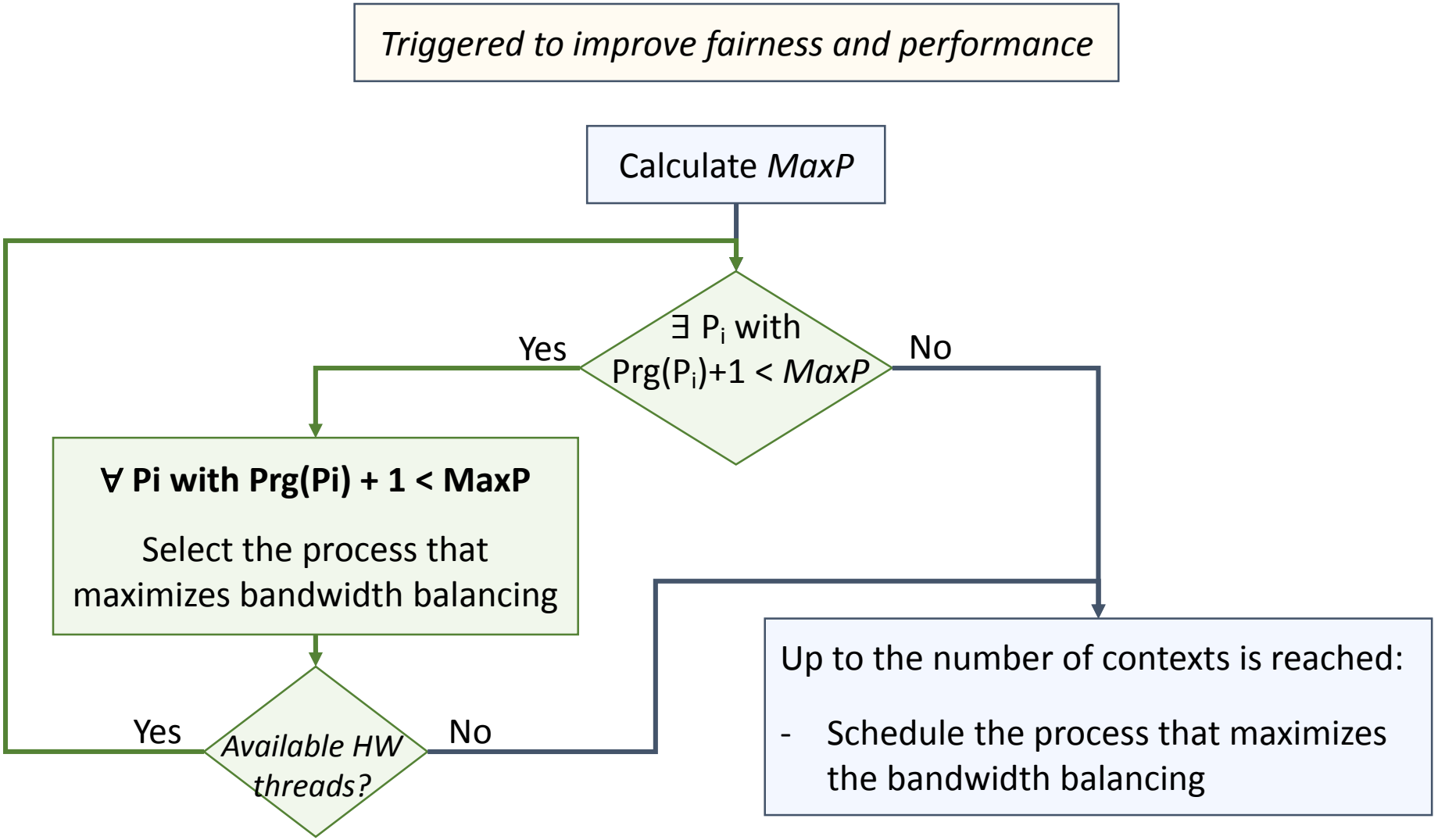
Progress-aware Perf&Fair scheduler

Performance- & Fairness- oriented mode



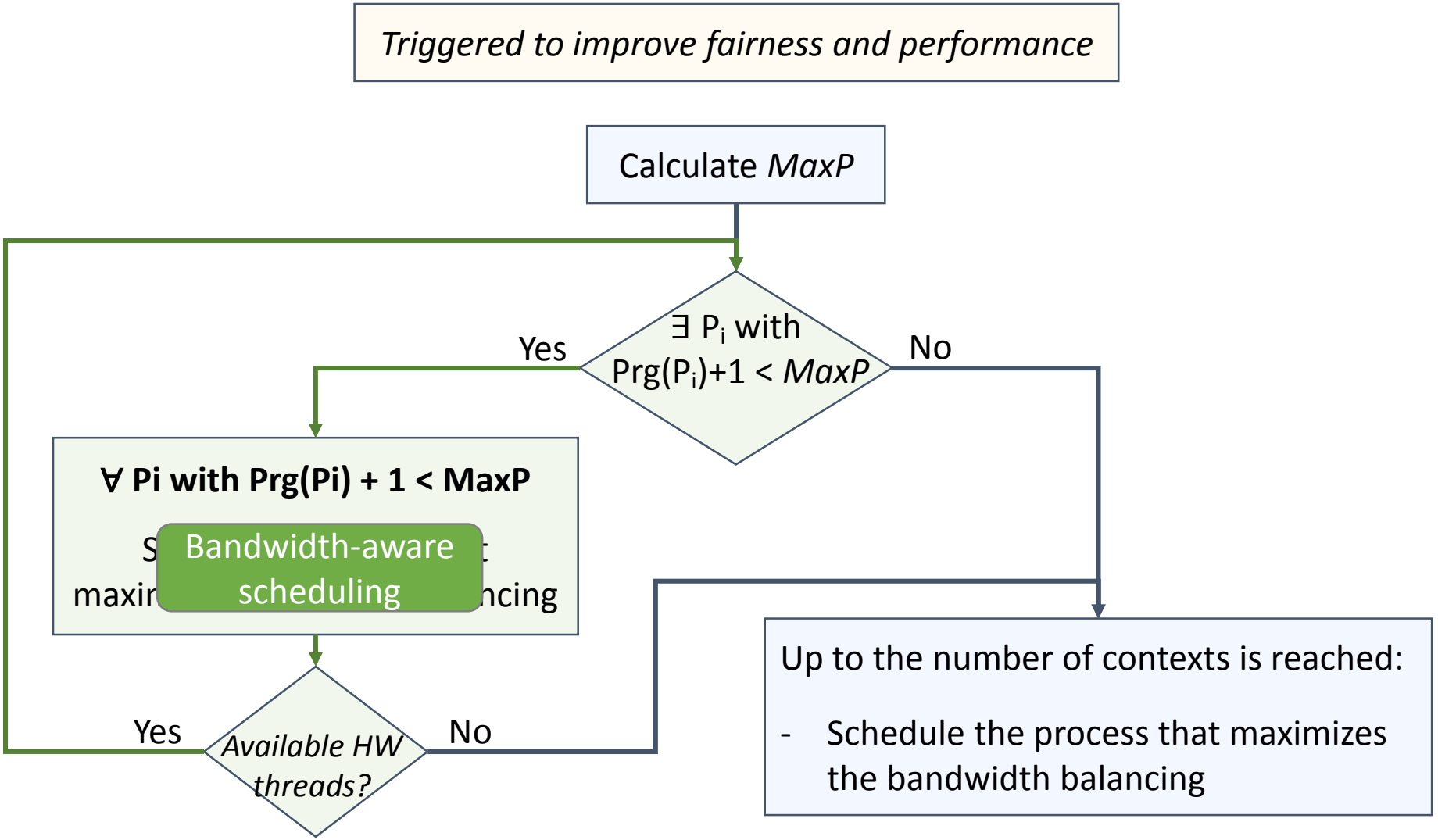
Progress-aware Perf&Fair scheduler

Performance- & Fairness- oriented mode



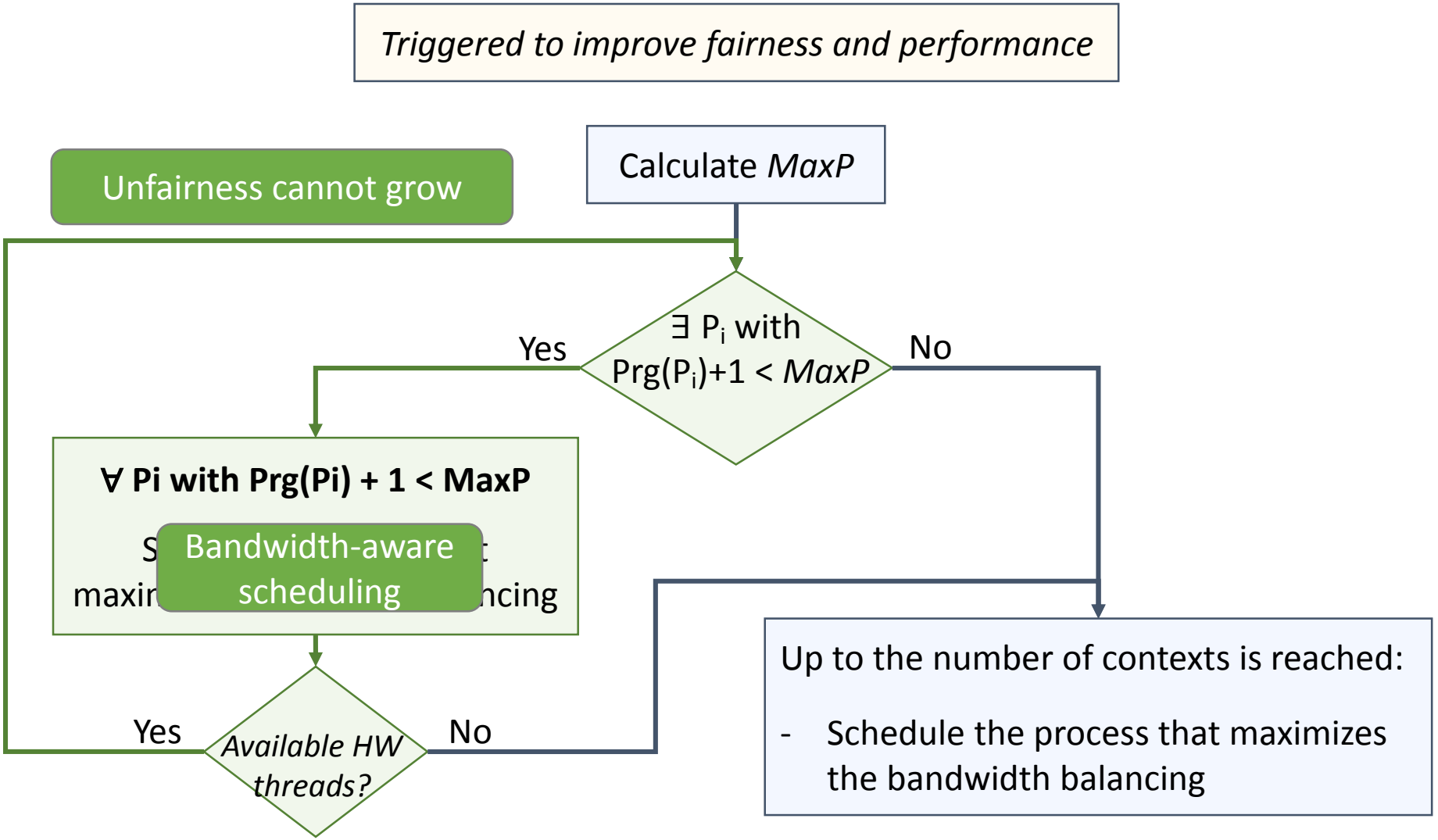
Progress-aware Perf&Fair scheduler

Performance- & Fairness- oriented mode



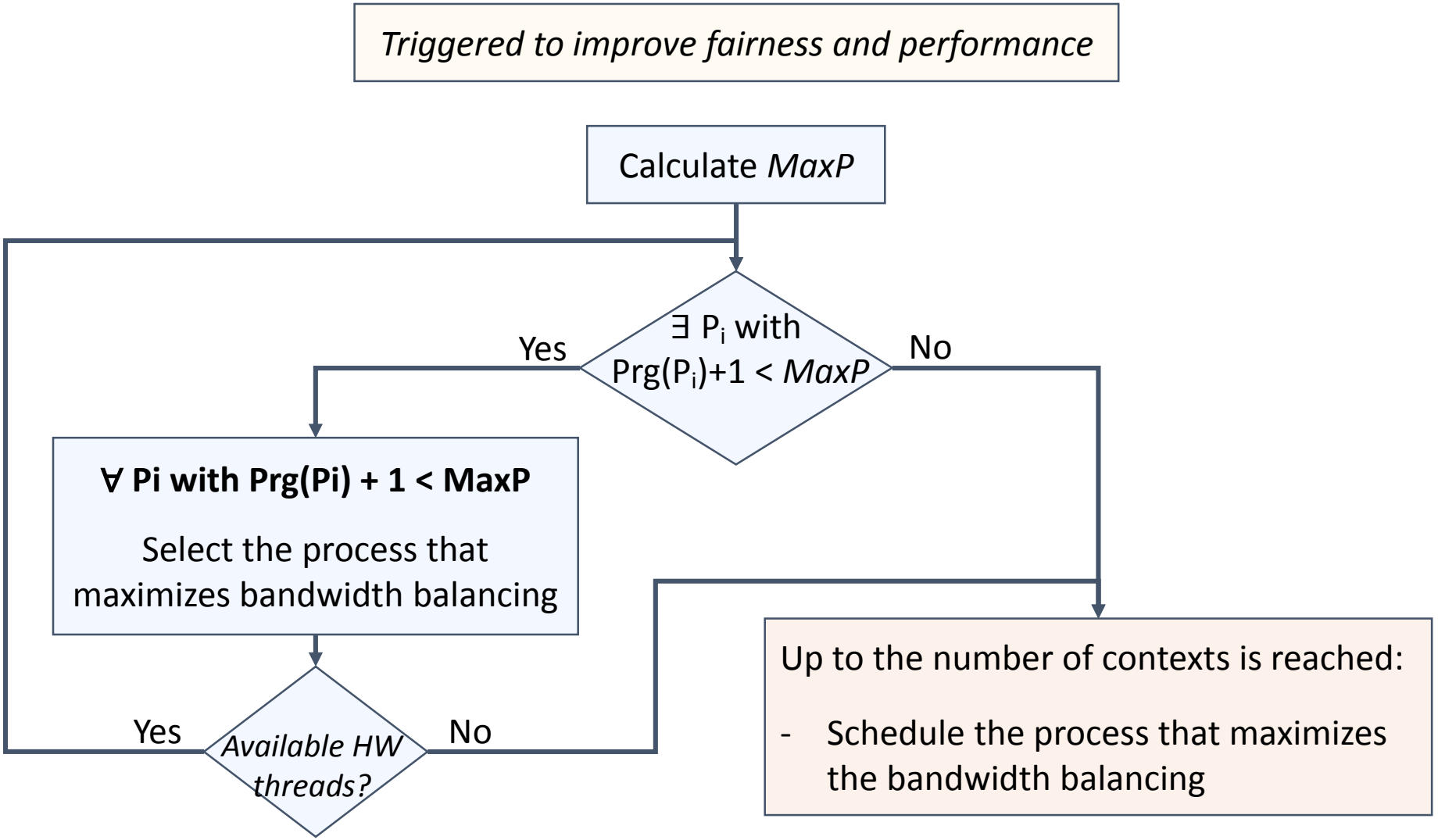
Progress-aware Perf&Fair scheduler

Performance- & Fairness- oriented mode



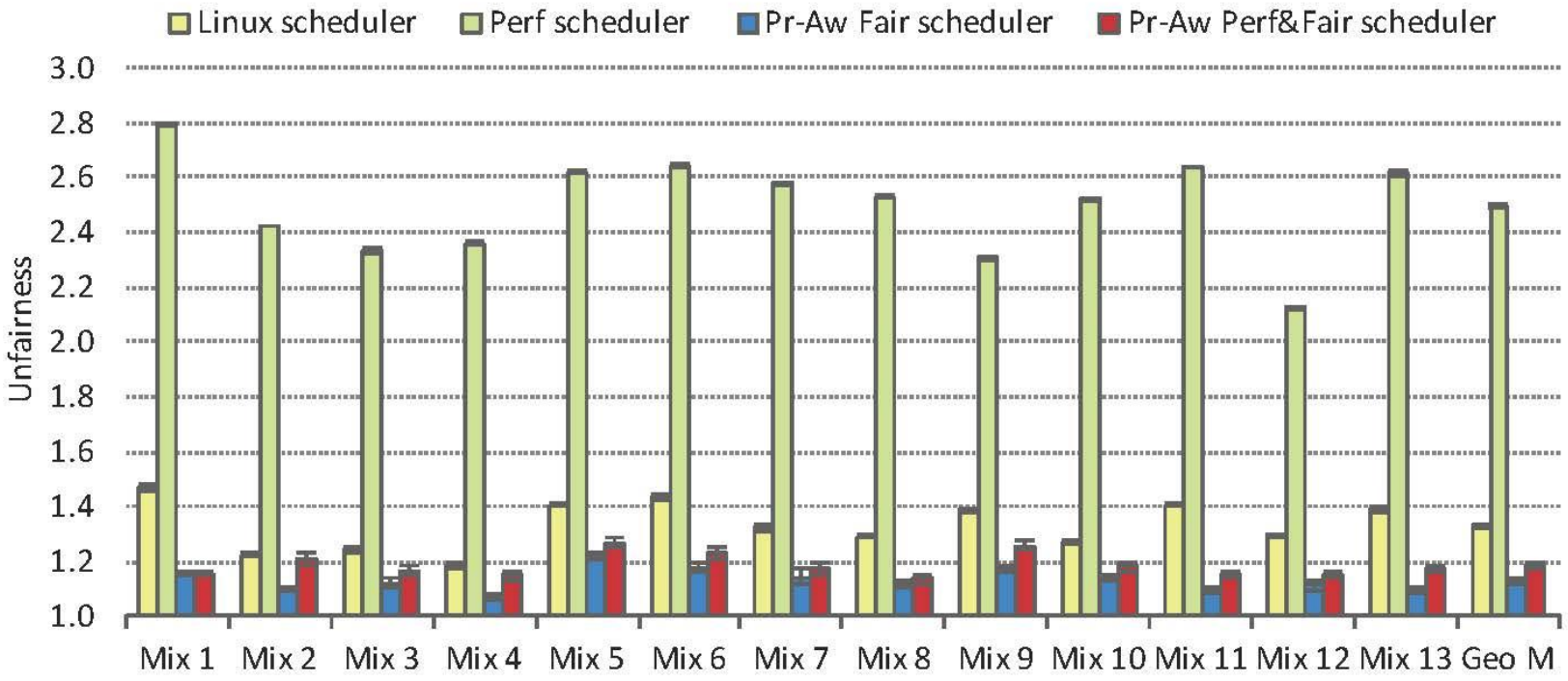
Progress-aware Perf&Fair scheduler

Performance- & Fairness- oriented mode



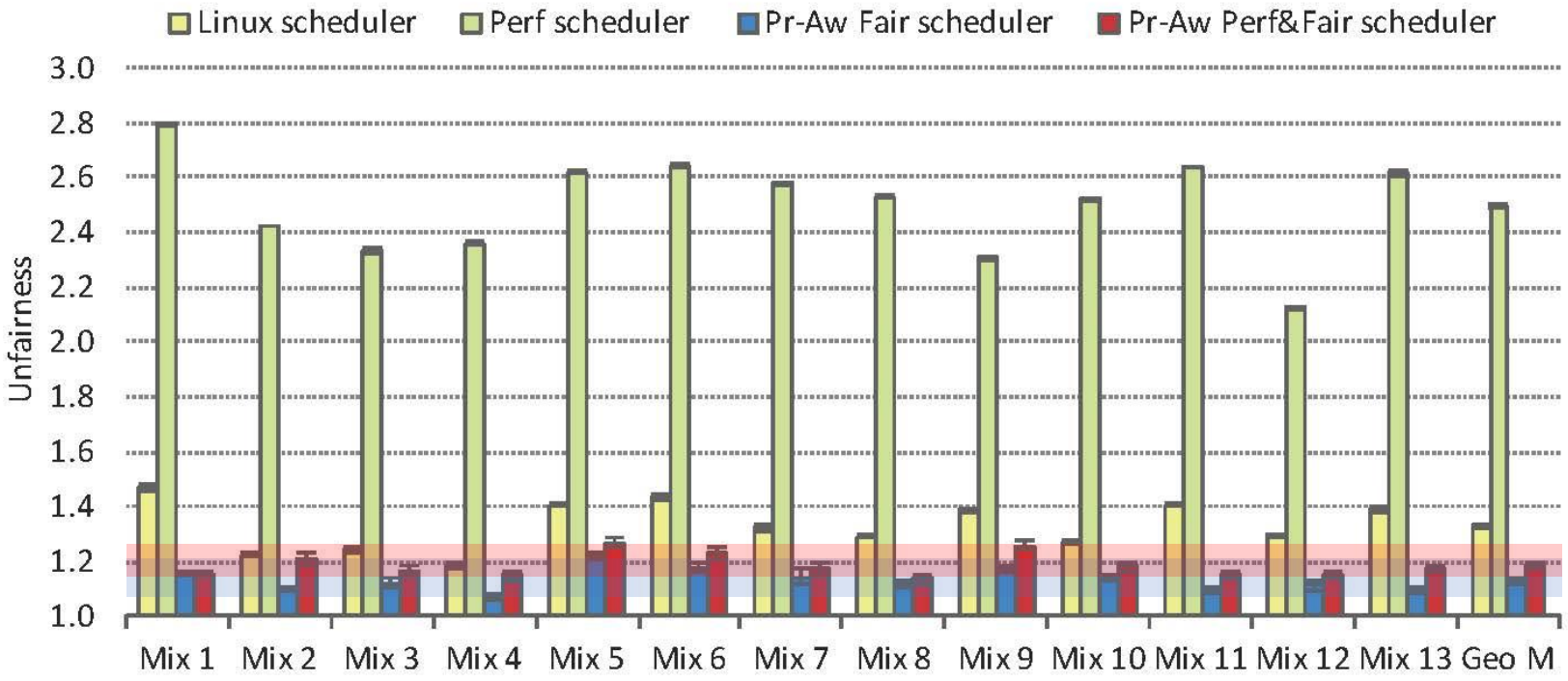
Experimental evaluation

Fairness (II)



Experimental evaluation

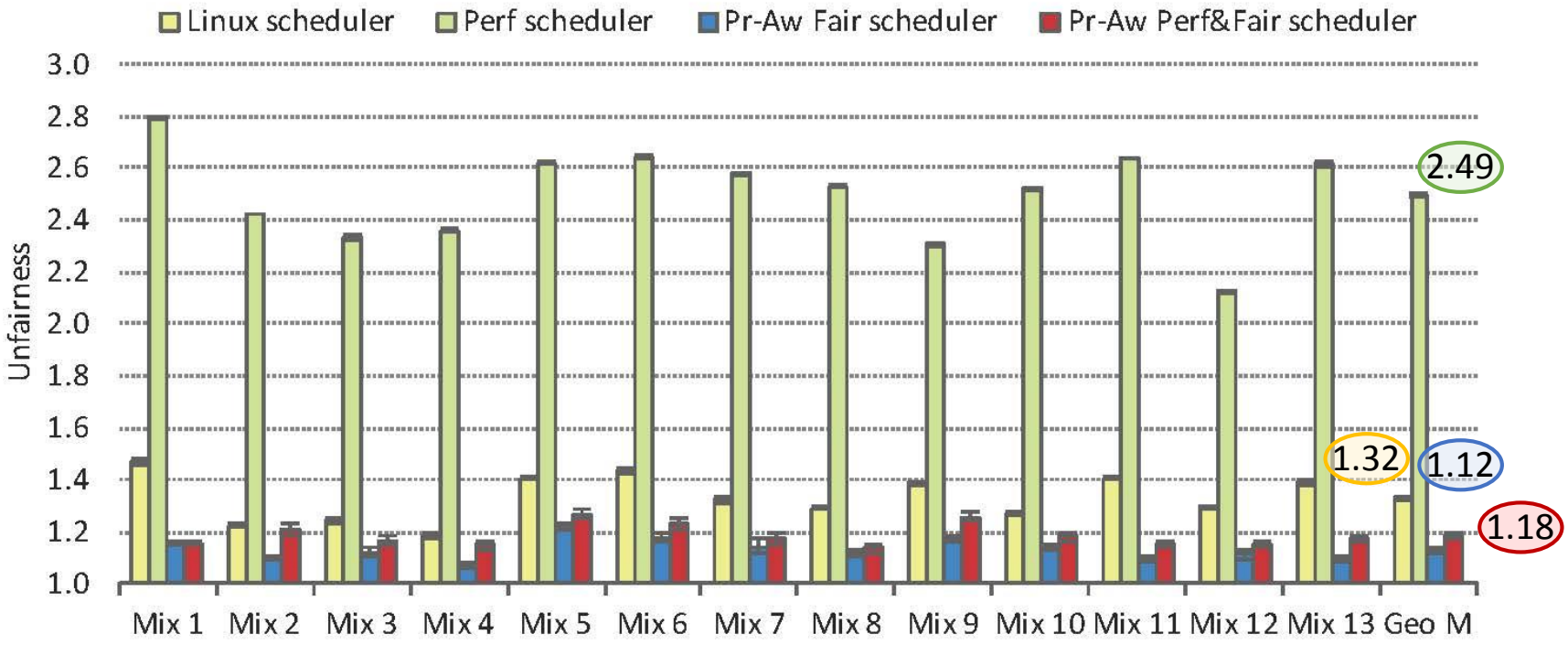
Fairness (II)



Unfairness with Perf&Fair slightly above that of *Fair*

Experimental evaluation

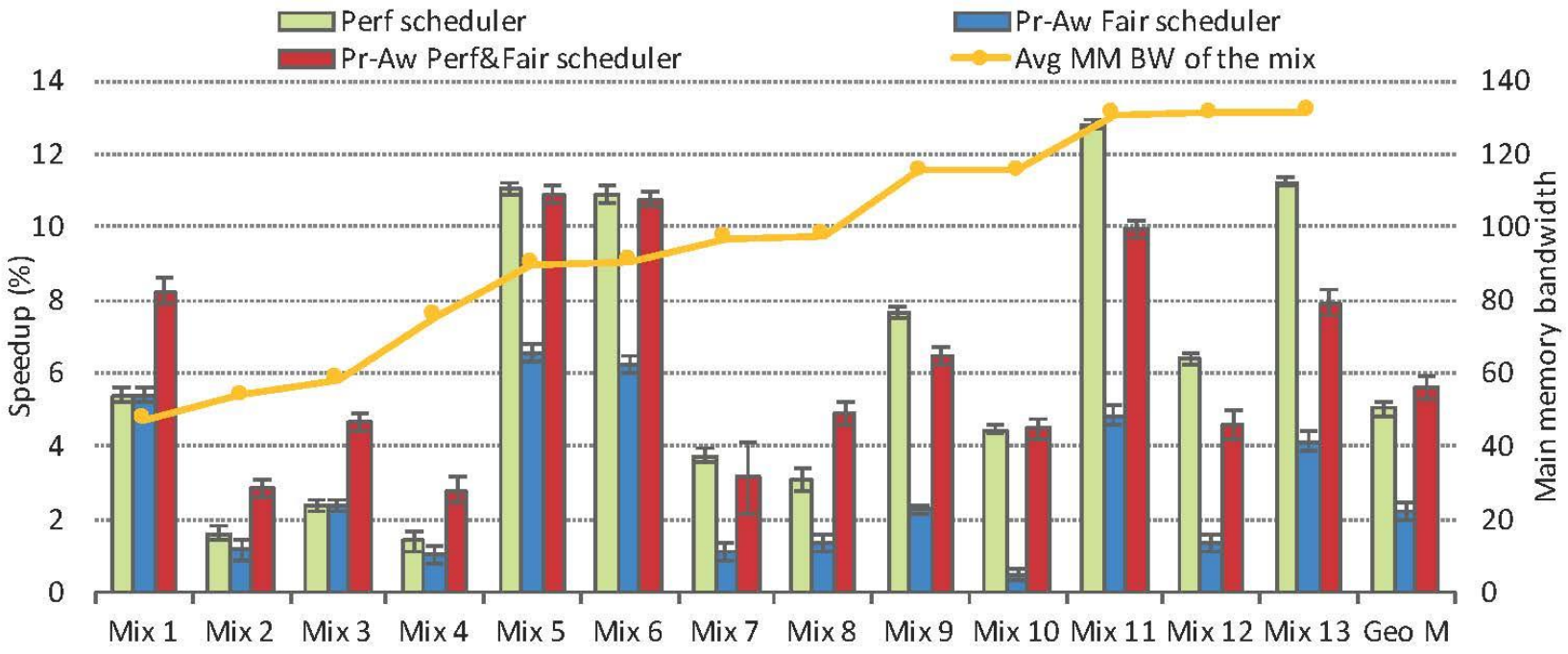
Fairness (II)



Perf&Fair reduces Linux unfairness close to a half

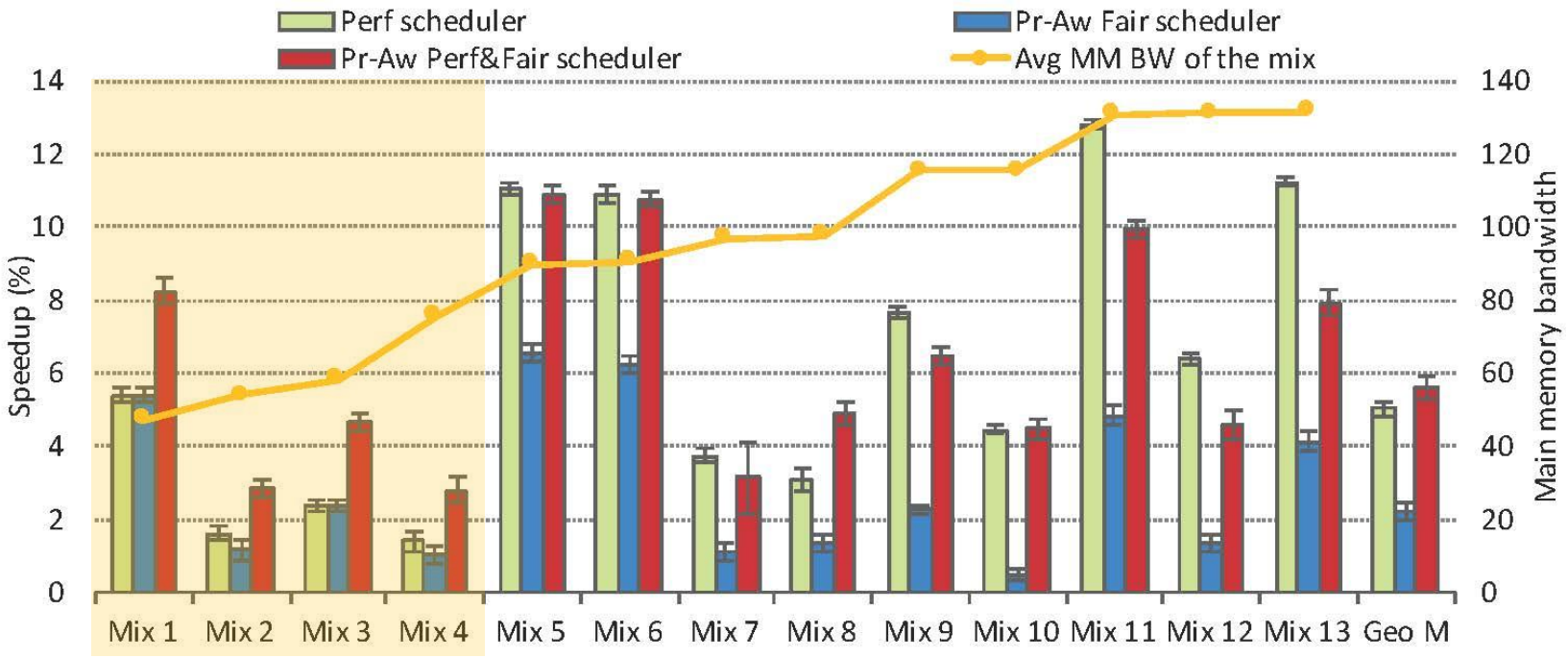
Experimental evaluation

Speedup of the turnaround time over Linux (II)



Experimental evaluation

Speedup of the turnaround time over Linux (II)

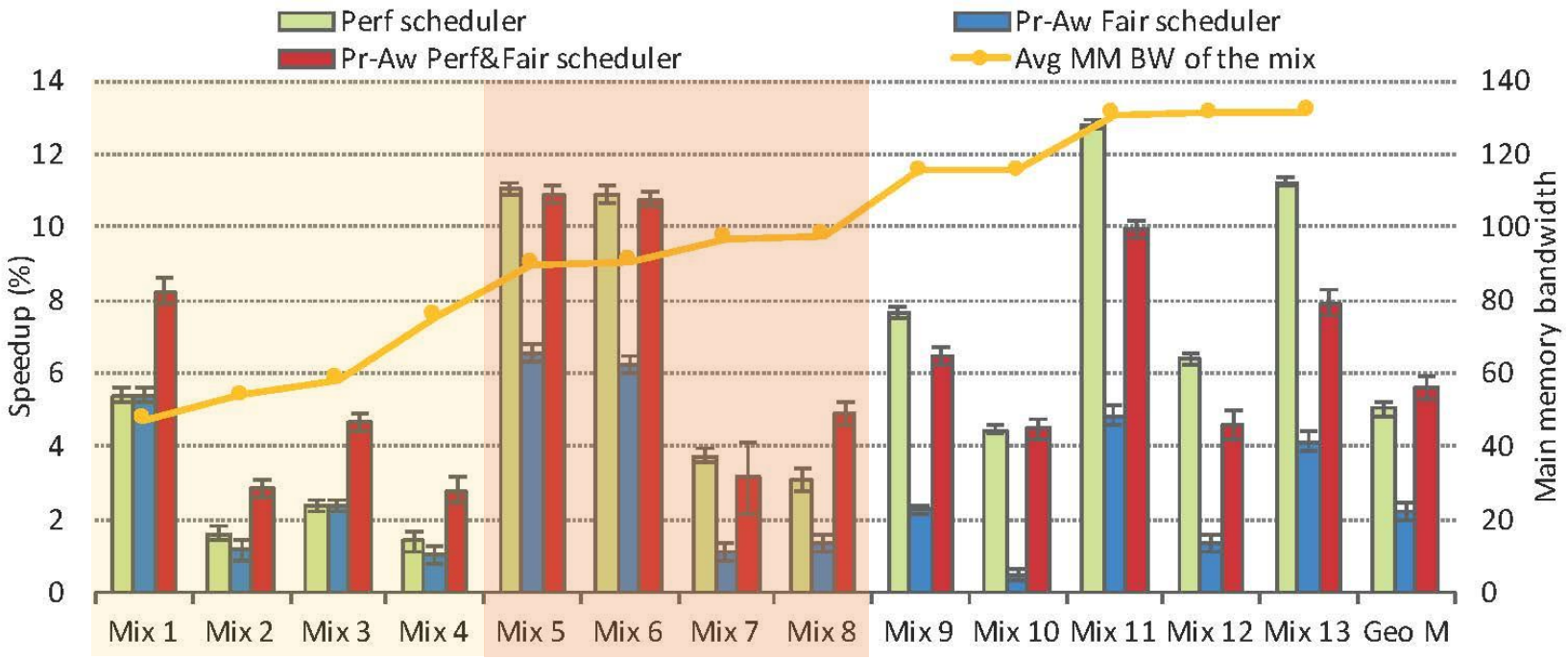


- Perf&Fair scheduler performance with respect to *Perf*:

Better with “low” bandwidth contention

Experimental evaluation

Speedup of the turnaround time over Linux (II)

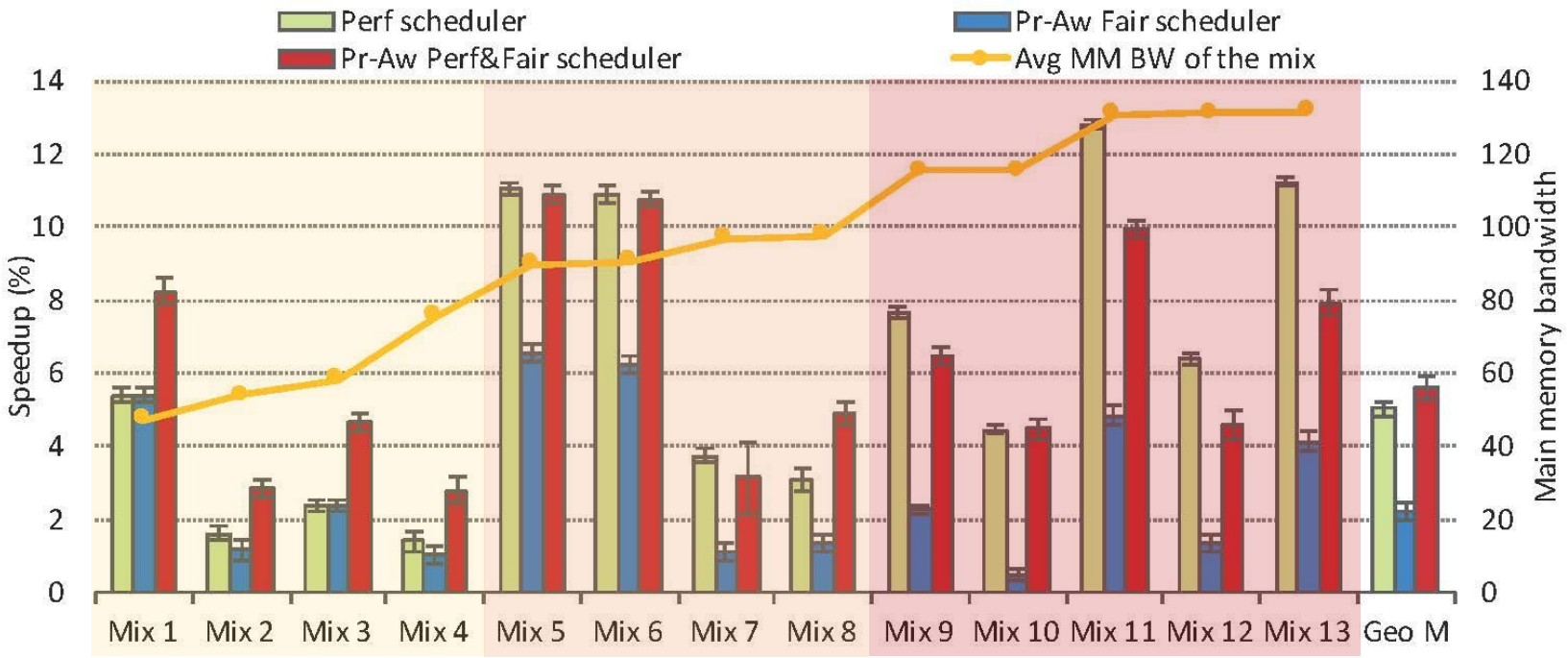


Better with “low” bandwidth contention

Similar with “medium” bandwidth contention

Experimental evaluation

Speedup of the turnaround time over Linux (II)



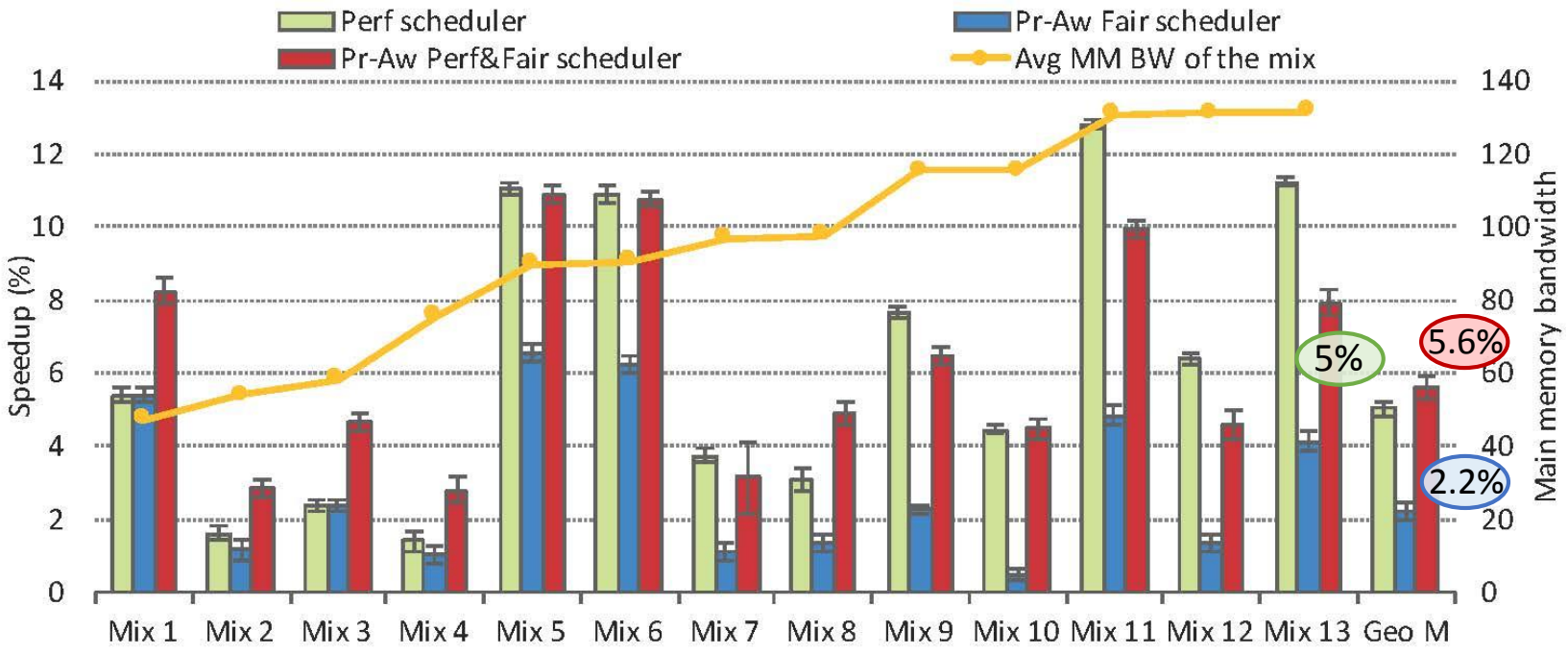
Better with “low” bandwidth contention

Similar with “medium” bandwidth contention

Worse with “high” bandwidth contention

Experimental evaluation

Speedup of the turnaround time over Linux (II)



Perf&Fair achieves, on average, better performance than *Perf* across all evaluated mixes

Outline

I. Bandwidth-Aware Scheduling on Multicores

II. Bandwidth-Aware Scheduling on SMT Multicores

III. Progress-Aware Scheduling on SMT Multicores

IV. Symbiotic Job Scheduling on the IBM POWER8

1. Introduction
2. Predicting job symbiosis
3. Symbiotic scheduling
4. Experimental evaluation

Symbiotic job scheduling

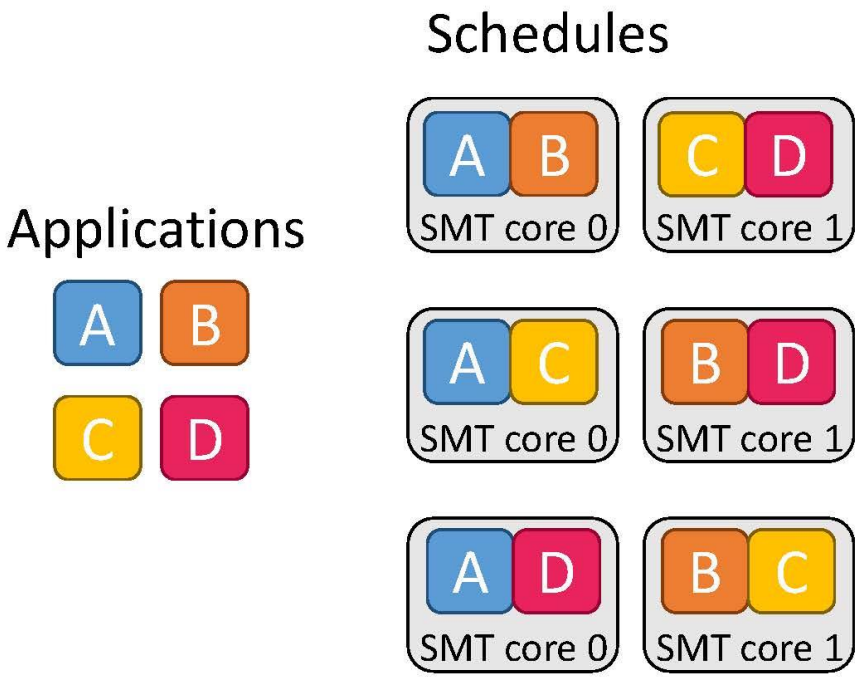
Introduction

- Symbiotic scheduler: based on a model that estimates job symbiosis
  - Predicts the slowdown of the applications on a schedule
  - It is fast, allowing to find a (close to) optimal schedule

Symbiotic job scheduling

Introduction

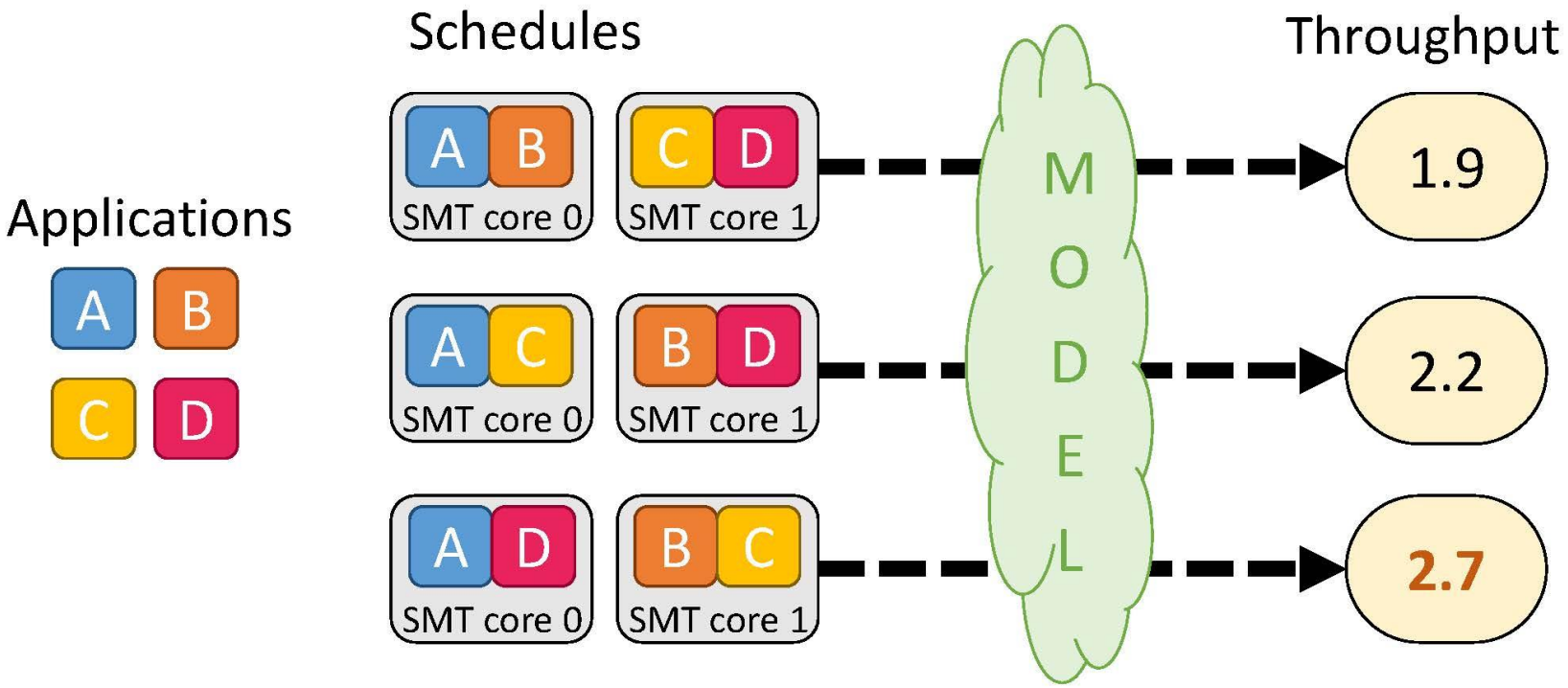
- Symbiotic scheduler: based on a model that estimates job symbiosis
  - Predicts the slowdown of the applications on a schedule
  - It is fast, allowing to find a (close to) optimal schedule



Symbiotic job scheduling

Introduction

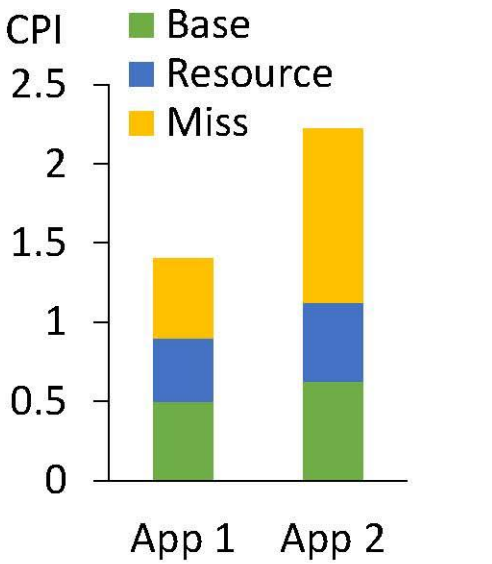
- Symbiotic scheduler: based on a model that estimates job symbiosis
  - Predicts the slowdown of the applications on a schedule
  - It is fast, allowing to find a (close to) optimal schedule



Predicting job symbiosis

Interference model

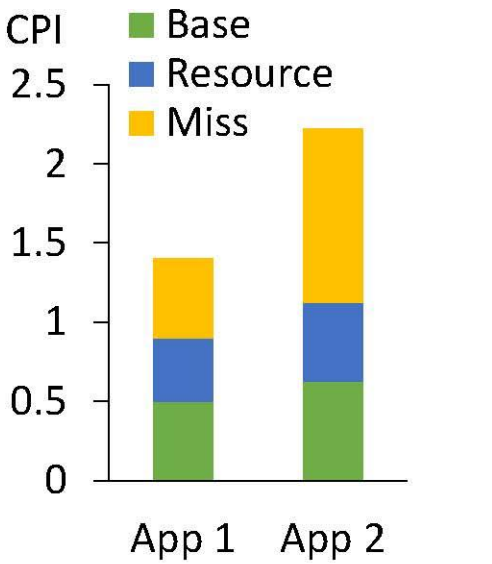
- The proposed model leverages CPI stacks to predict job symbiosis



Measured single-threaded **(ST)** CPI stacks

Predicting job symbiosis
Interference model

- The proposed model leverages CPI stacks to predict job symbiosis



Measured single-threaded **(ST)** CPI stacks

CPI Stacks

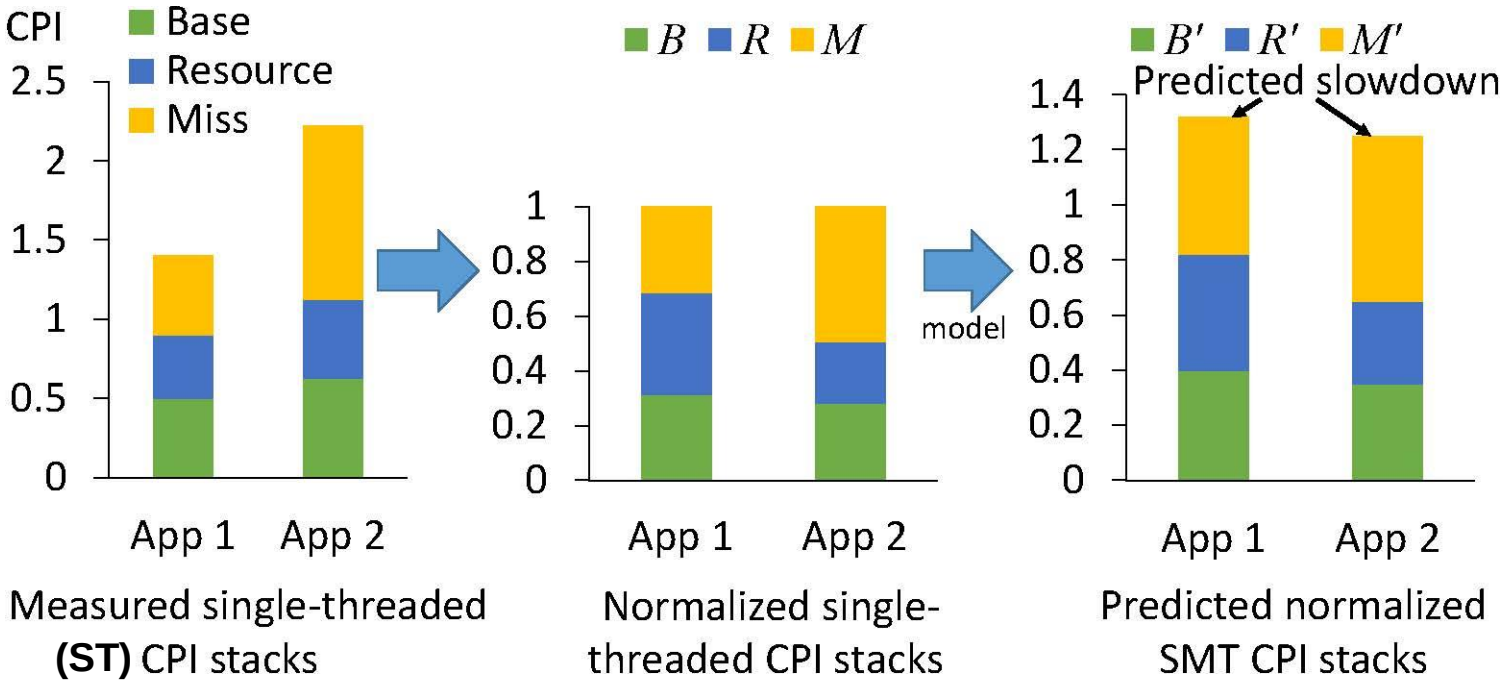
Divide the execution cycles into various components:

- **Base**: instructions are completed
- **Resource**: no instruction completed due to resource stall
- **Miss**: no instruction completed due to miss event

Predicting job symbiosis

Interference model

- The proposed model leverages CPI stacks to predict job symbiosis
- Model: estimates the slowdown
  - Interprets the normalized CPI components as probabilities
  - Calculates the probability of interference



Predicting job symbiosis

Model equation

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k$$

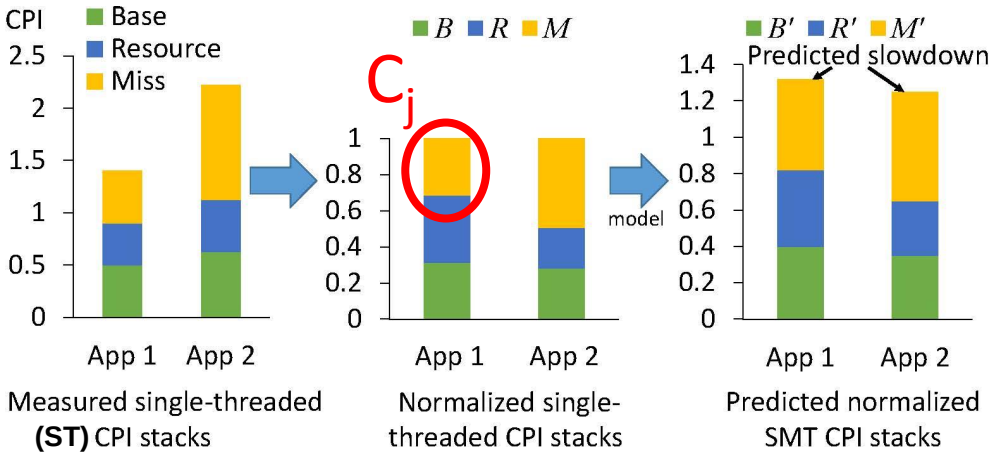
Predicting job symbiosis

Model equation

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k$$

Components

- C_j represents thread j own component in ST (single-threaded) mode
- C_k represents the ST component of the other threads in the schedule
- C'_j identifies the SMT component of thread j



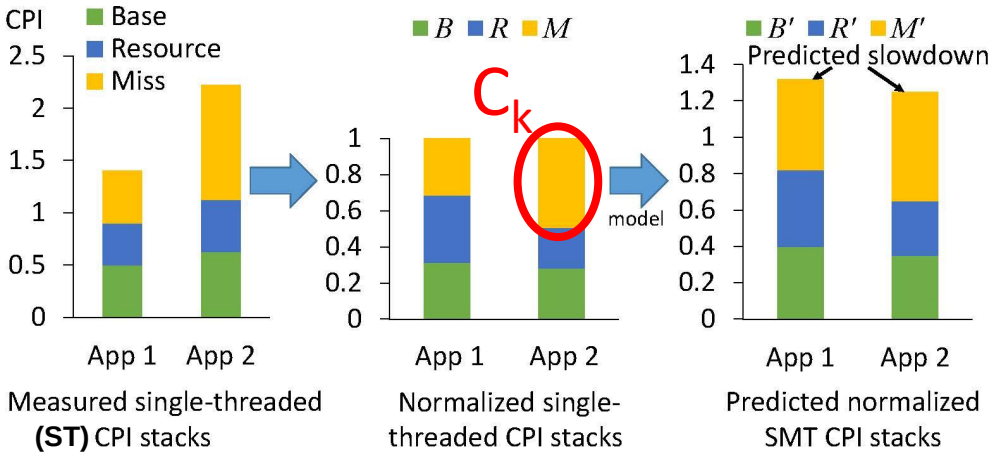
Predicting job symbiosis

Model equation

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k$$

Components

- C_j represents thread j own component in ST mode
- C_k represents the ST component of the other threads in the schedule
- C'_j identifies the SMT component of thread j



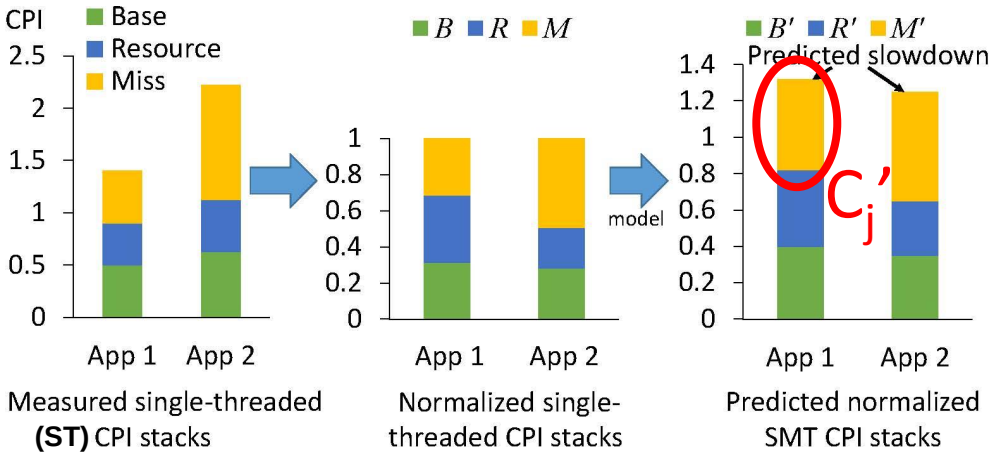
Predicting job symbiosis

Model equation

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k$$

Components

- C_j represents thread j own component in ST mode
- C_k represents the ST component of the other threads in the schedule
- C'_j identifies the SMT component of thread j



Predicting job symbiosis

Model equation

$$C'_j = \alpha_C + \beta_C C_j + \gamma_C \sum_{k \neq j} C_k + \delta_C C_j \sum_{k \neq j} C_k$$

Components

- C_j represents thread j own component in ST mode
- C_k represents the ST component of the other threads in the schedule
- C'_j identifies the SMT component of thread j

Parameters

- $\alpha_c, \beta_c, \gamma_c$, and δ_c
- Tied to specific components, not to applications

Predicting job symbiosis

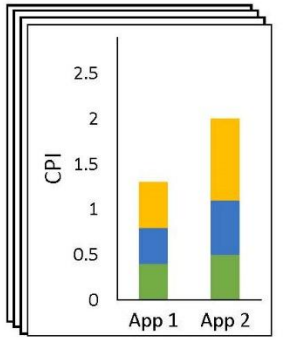
Model construction and slowdown estimation

- Model parameters determined by linear regression
  - One-time offline training
  - Based on experimental data
  - Tied to CPI components, not to applications -> no need to retrain

Predicting job symbiosis

Model construction and slowdown estimation

- Model parameters determined by linear regression
  - One-time offline training
  - Based on experimental data
  - Tied to CPI components, not to applications -> no need to retrain

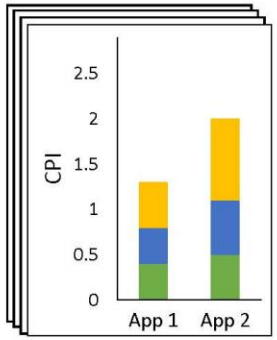


Measured ST CPI stacks (Alone)

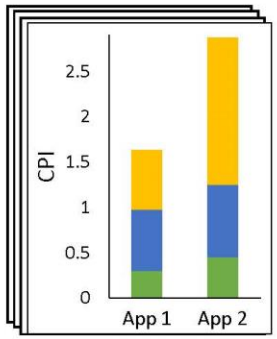
Predicting job symbiosis

Model construction and slowdown estimation

- Model parameters determined by linear regression
  - One-time offline training
  - Based on experimental data
  - Tied to CPI components, not to applications -> no need to retrain



Measured ST CPI stacks
(Alone)

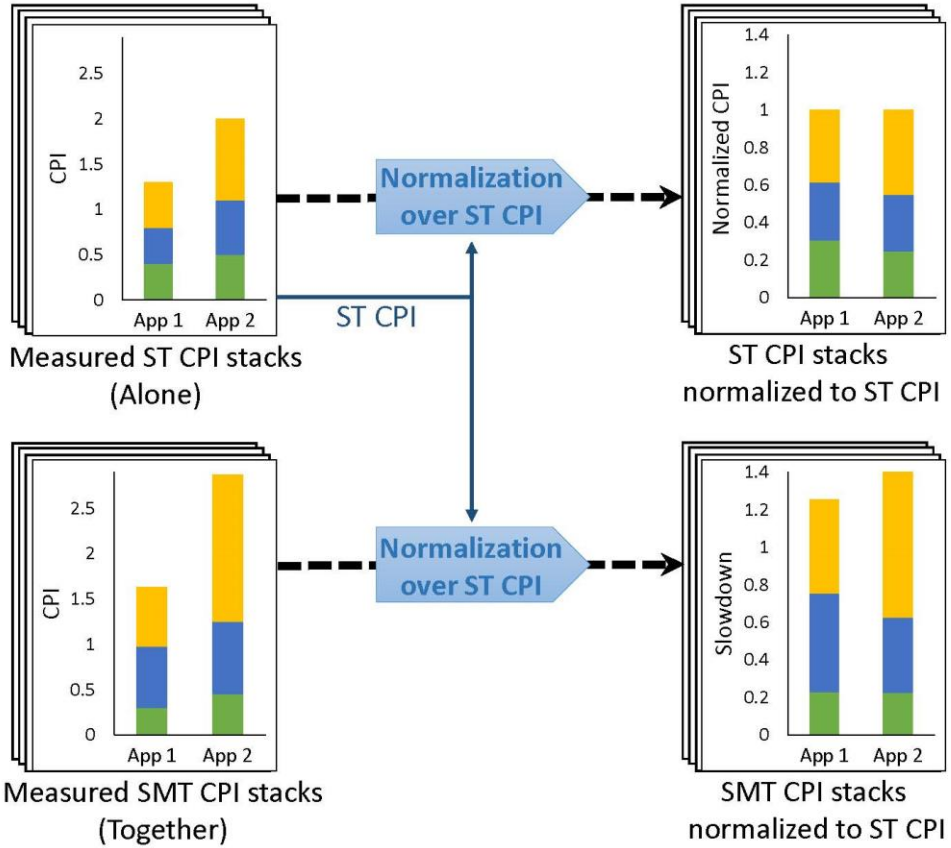


Measured SMT CPI stacks
(Together)

Predicting job symbiosis

Model construction and slowdown estimation

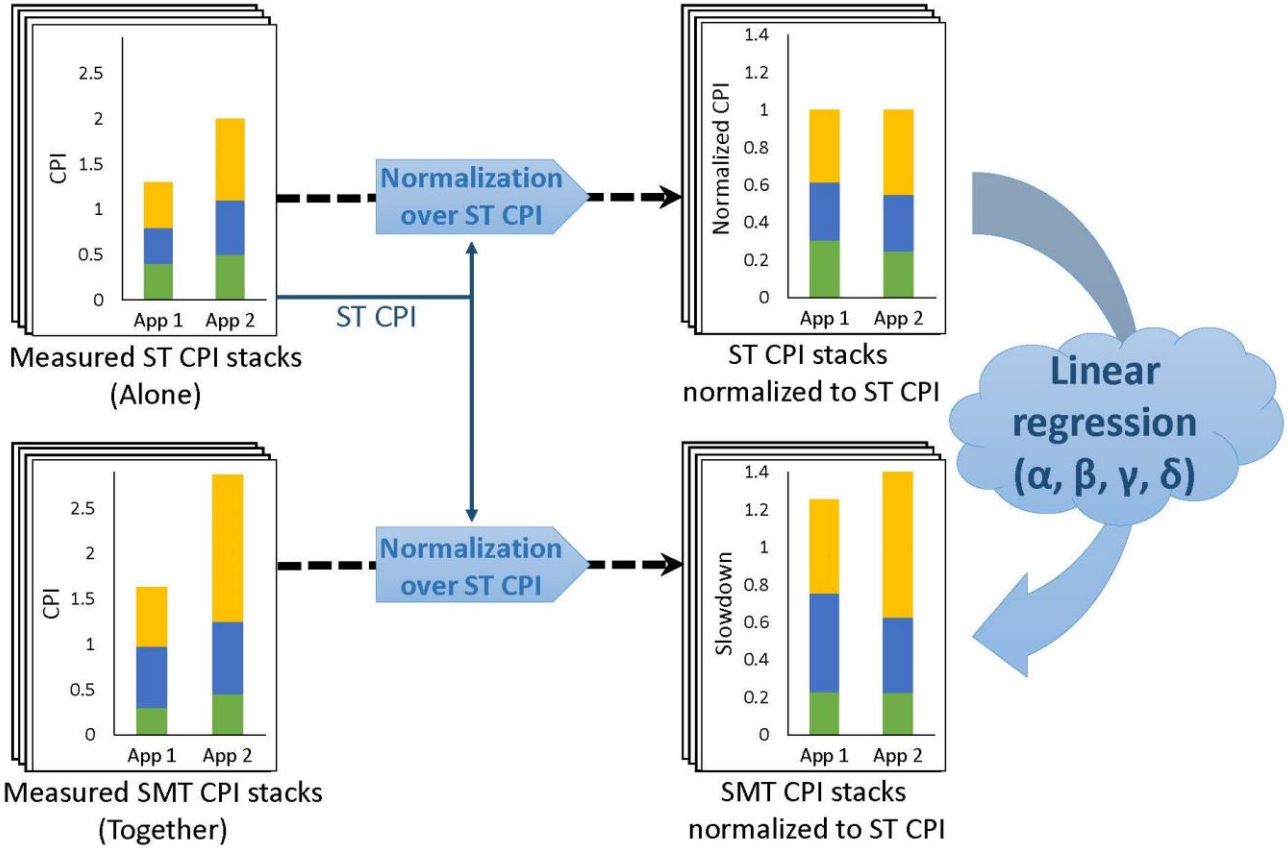
- Model parameters determined by linear regression
  - One-time offline training
  - Based on experimental data
  - Tied to CPI components, not to applications -> no need to retrain



Predicting job symbiosis

Model construction and slowdown estimation

- Model parameters determined by linear regression
  - One-time offline training
  - Based on experimental data
  - Tied to CPI components, not to applications -> no need to retrain



SMT interference-aware scheduling

Selection of the (optimal) schedule

- Too large number of different schedules
  - $\frac{n!}{c!\,(\frac{n}{c}!)^c}$ *n* application onto *c* cores
  - More than 2M schedules for 16 application in 8 cores!

SMT interference-aware scheduling

Selection of the (optimal) schedule

- Too large number of different schedules
  - $\frac{n!}{c!\,(\frac{n}{c}!)^c}$ *n* application onto *c* cores
  - More than 2M schedules for 16 application in 8 cores!
- SMT2 (2 threads per core) scheduling
  - Modeled as a minimum-weight perfect matching problem
  - Solved in polynomial time using the blossom algorithm
  - Select the optimal schedule

SMT interference-aware scheduling

Selection of the (optimal) schedule

- Too large number of different schedules
  - $\frac{n!}{c!\,(\frac{n}{c}!)^c}$ *n* application onto *c* cores
  - More than 2M schedules for 16 application in 8 cores!
- SMT2 (2 threads per core) scheduling
  - Modeled as a minimum-weight perfect matching problem
  - Solved in polynomial time using the blossom algorithm
  - Select the optimal schedule
- SMT4 (4 threads per core) scheduling
  - NP-complete problem
  - Hierarchical perfect matching algorithm (proposed by Jiang et al[1])
  - Select a close to optimal schedule

[1] Y. Jiang, X. Shen, J. Chen, and R. Tripathi. "Analysis and Approximation of Optimal Co-scheduling on Chip Multiprocessors", at PACT 2008.

SMT interference-aware scheduling

Scheduling steps

Forward model

Selection graph

Inverted model

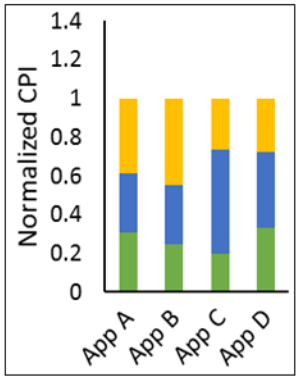
Blossom algorithm

Run!

SMT interference-aware scheduling

Scheduling steps

Measured SMT CPI stacks



Forward model

Selection graph

Inverted model

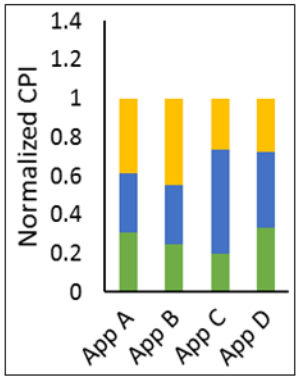
Blossom algorithm

Run!

SMT interference-aware scheduling

Scheduling steps

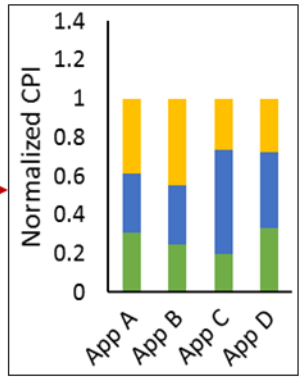
Measured SMT CPI stacks



Forward model

Inverted model

Estimated ST CPI stacks

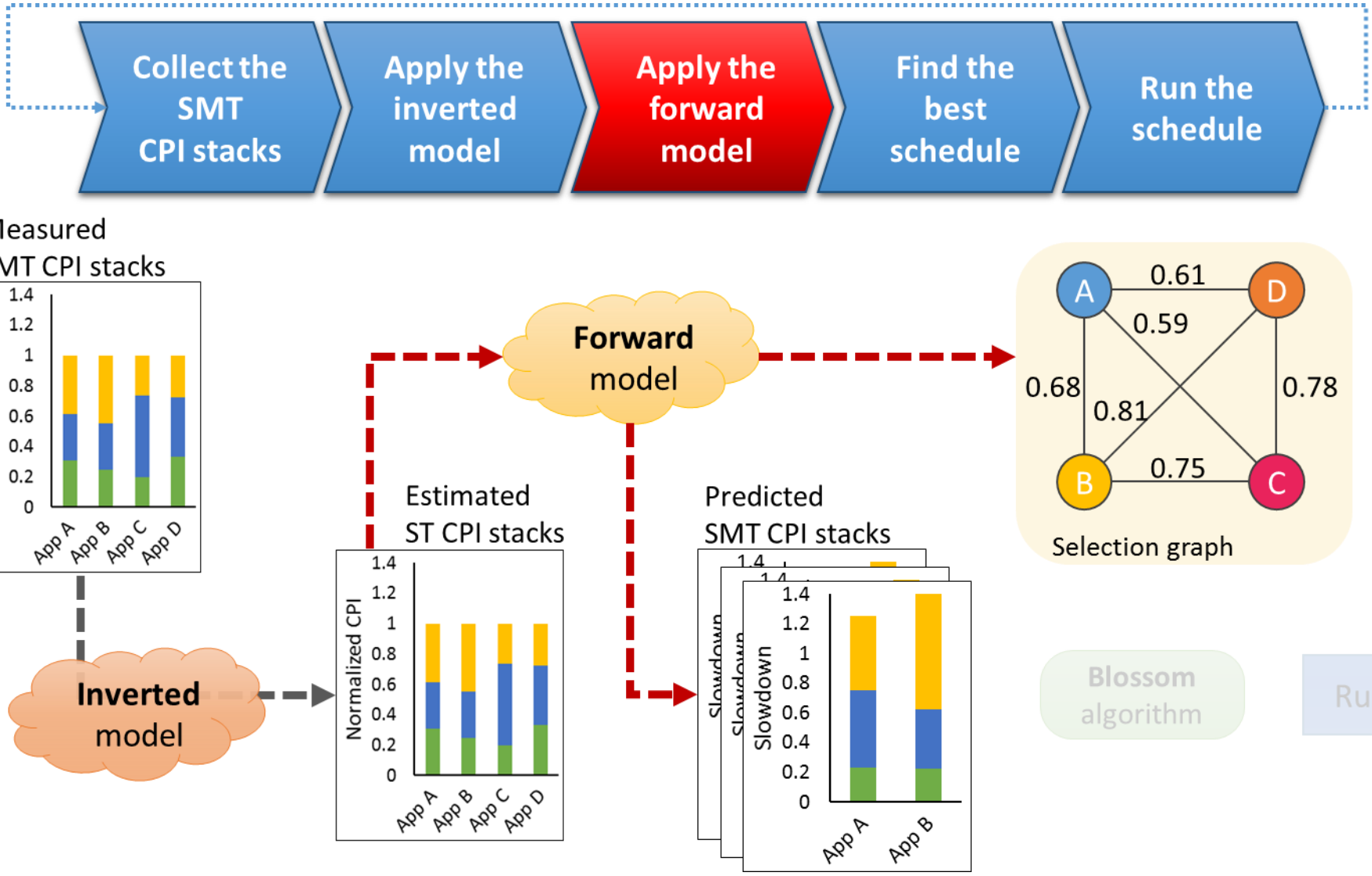


Blossom algorithm

Run!

SMT interference-aware scheduling

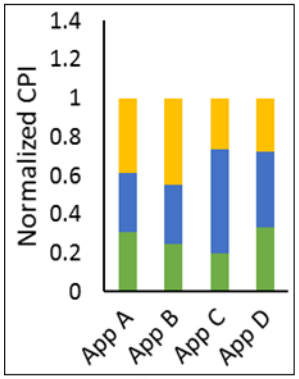
Scheduling steps



SMT interference-aware scheduling

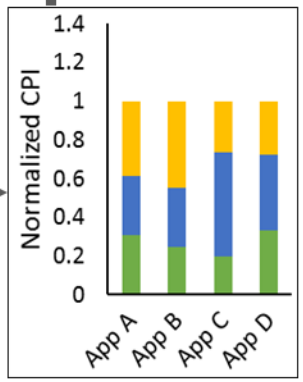
Scheduling steps

Measured SMT CPI stacks



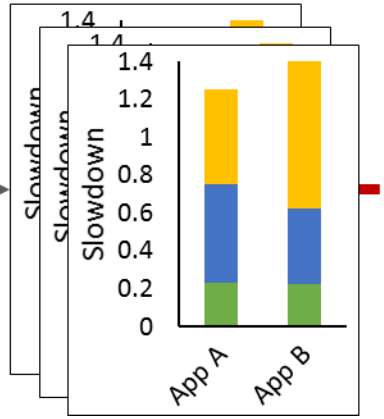
Inverted model

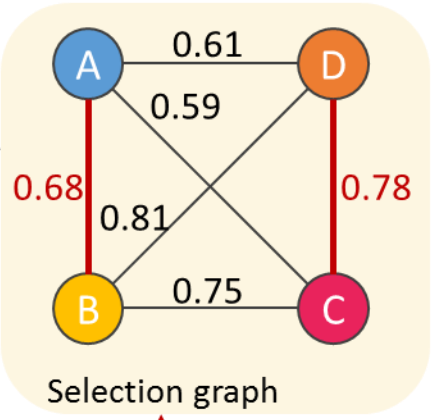
Estimated ST CPI stacks



Forward model

Predicted SMT CPI stacks



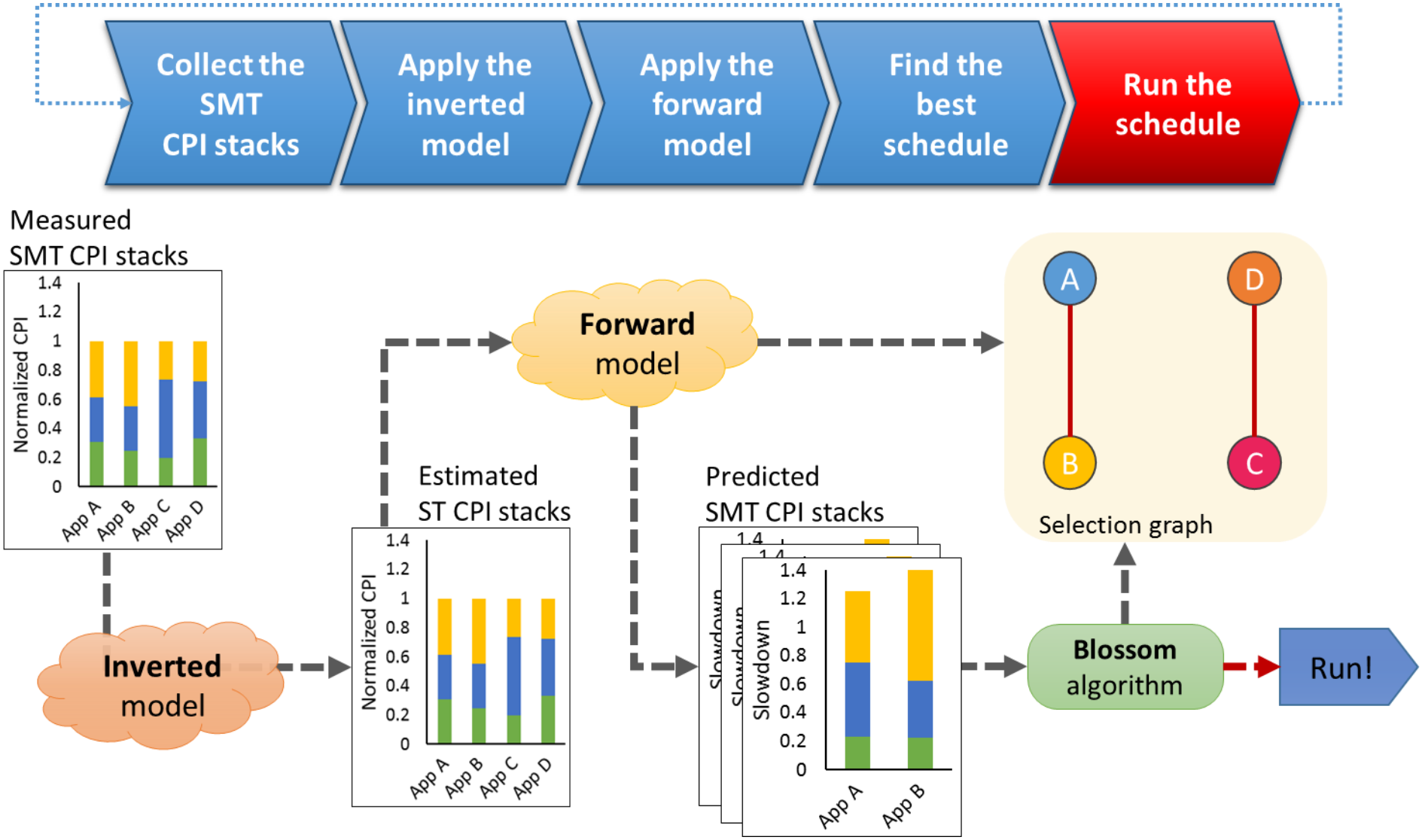


Blossom algorithm

Run!

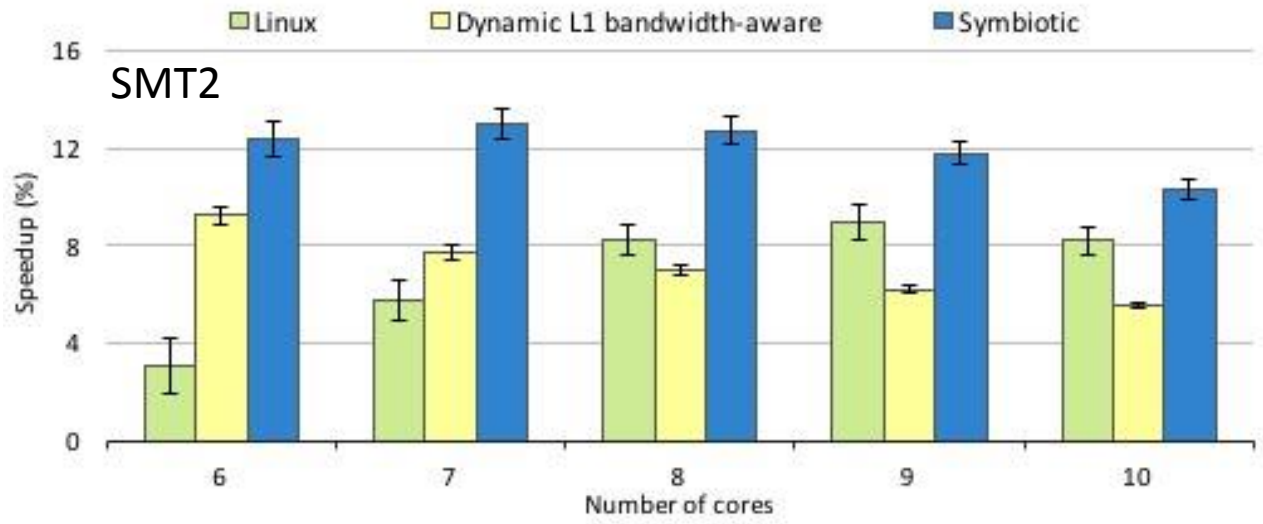
SMT interference-aware scheduling

Scheduling steps



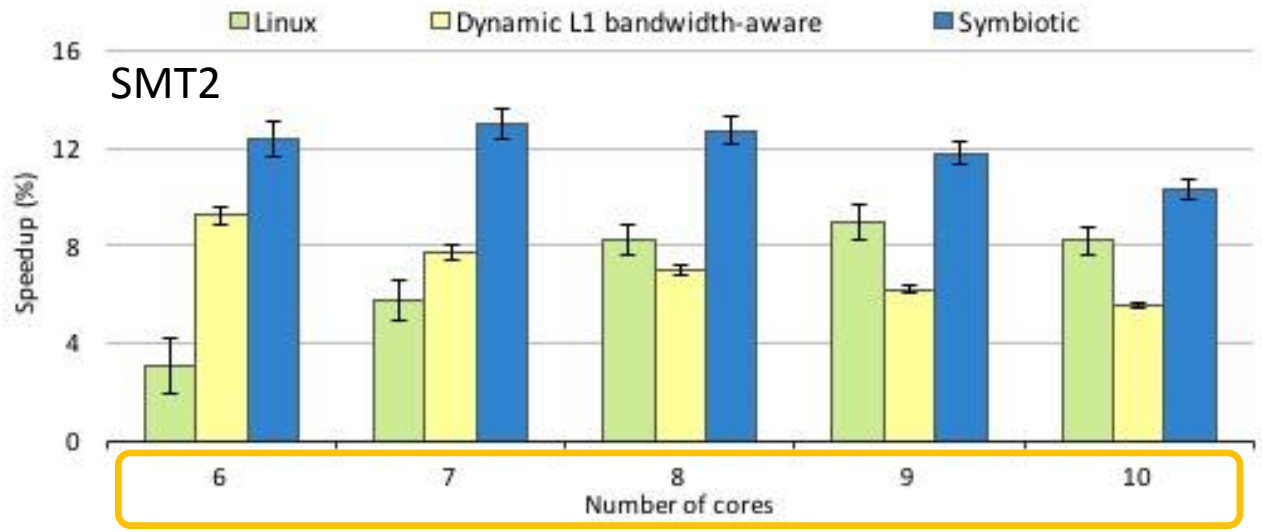
Experimental evaluation

System throughput



Experimental evaluation

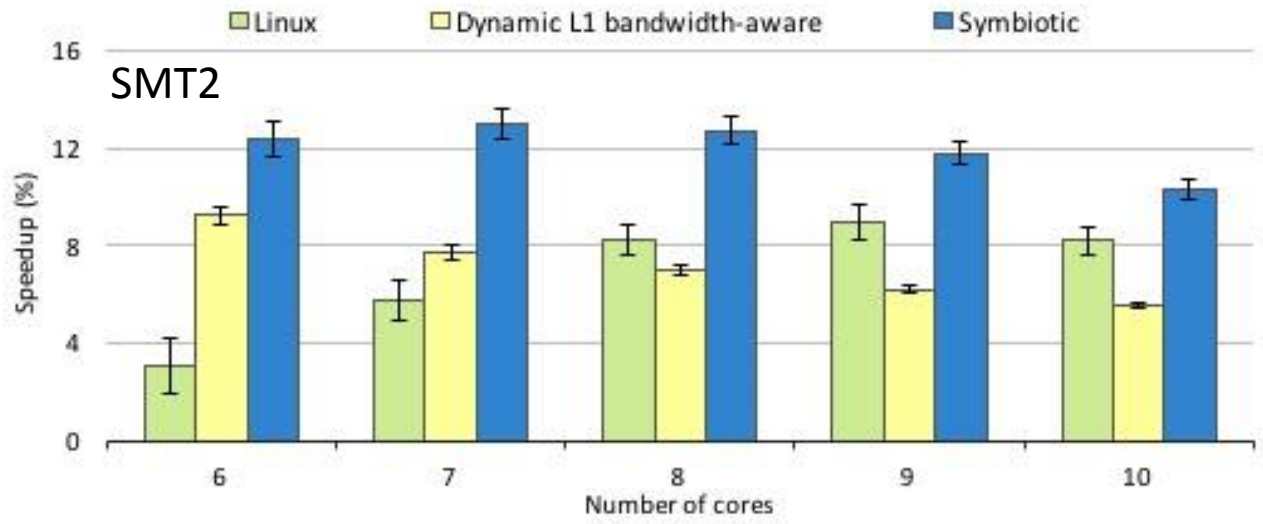
System throughput



- 15 mixes for each number of cores
- Each mix includes 2 applications per core

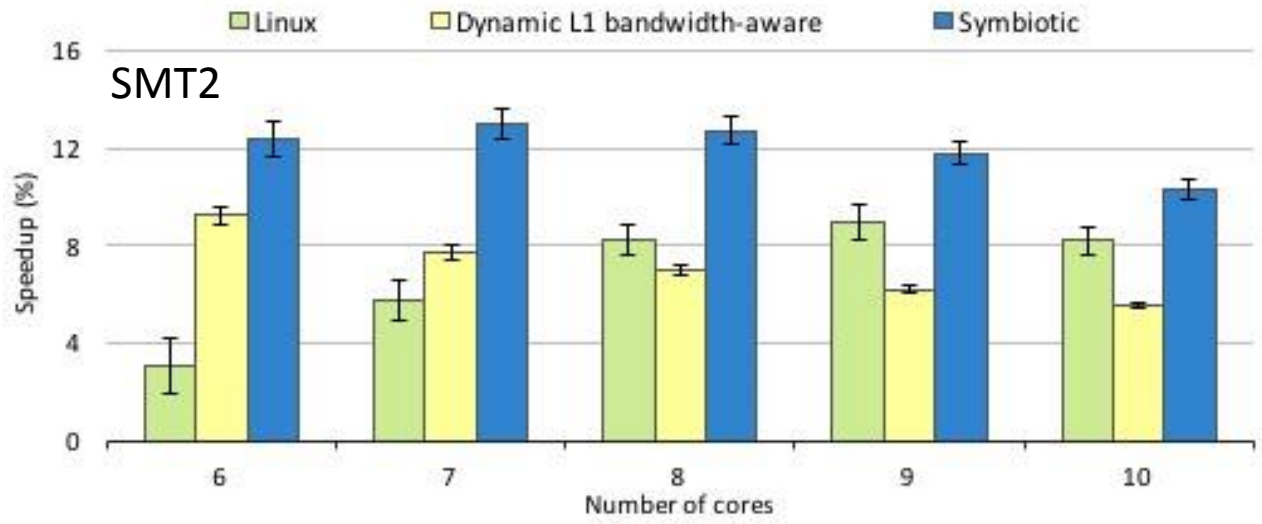
Experimental evaluation

System throughput



Experimental evaluation

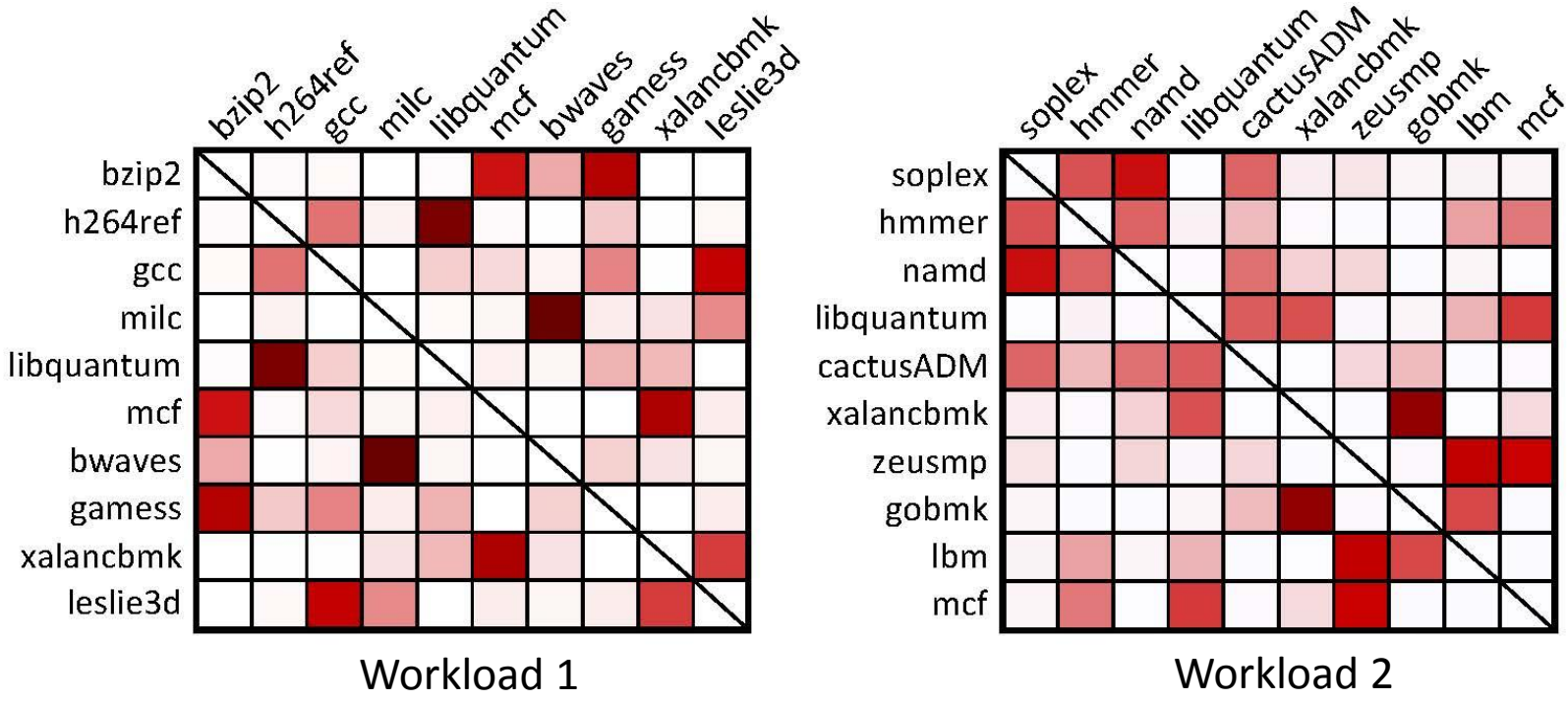
System throughput



Avg speedup w.r.t. Linux
5.2%

Experimental evaluation

Symbiosis patterns

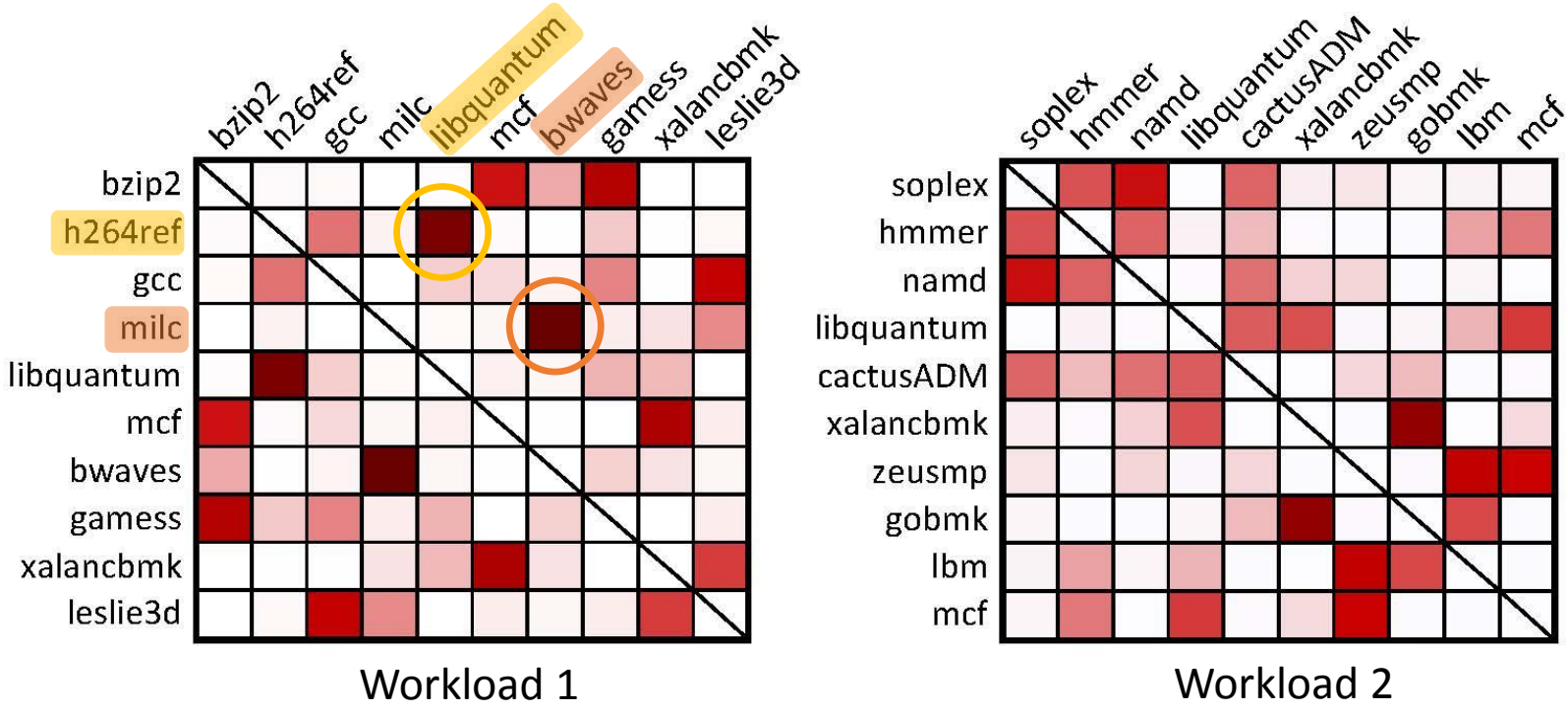


Frequency matrices of job coschedules for two workloads

- The darker the color, the more frequently the couple is scheduled together on an SMT core

Experimental evaluation

Symbiosis patterns



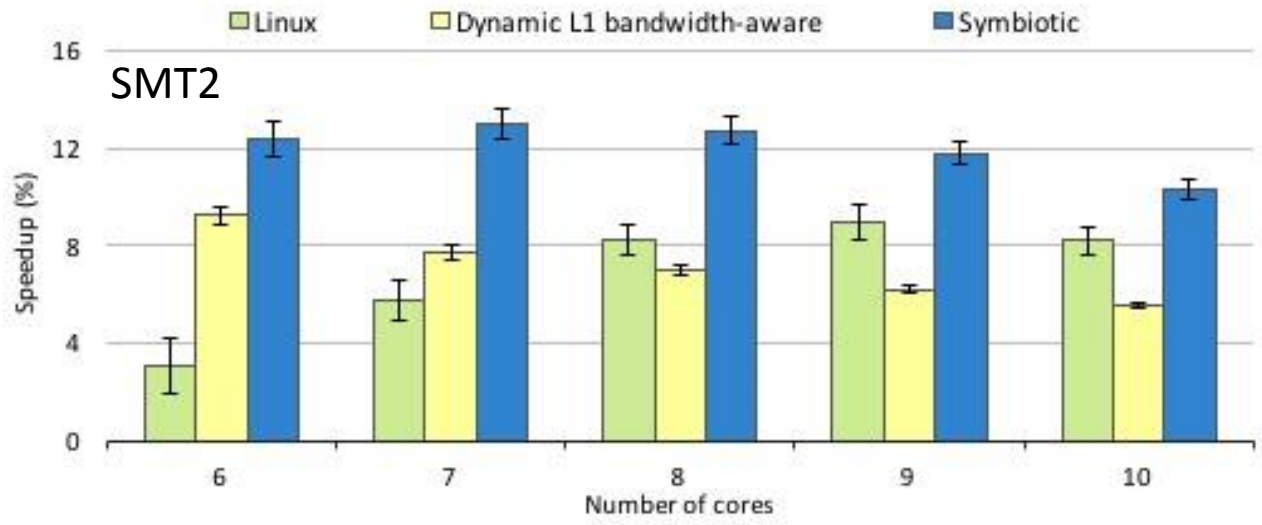
Couples scheduled very frequently

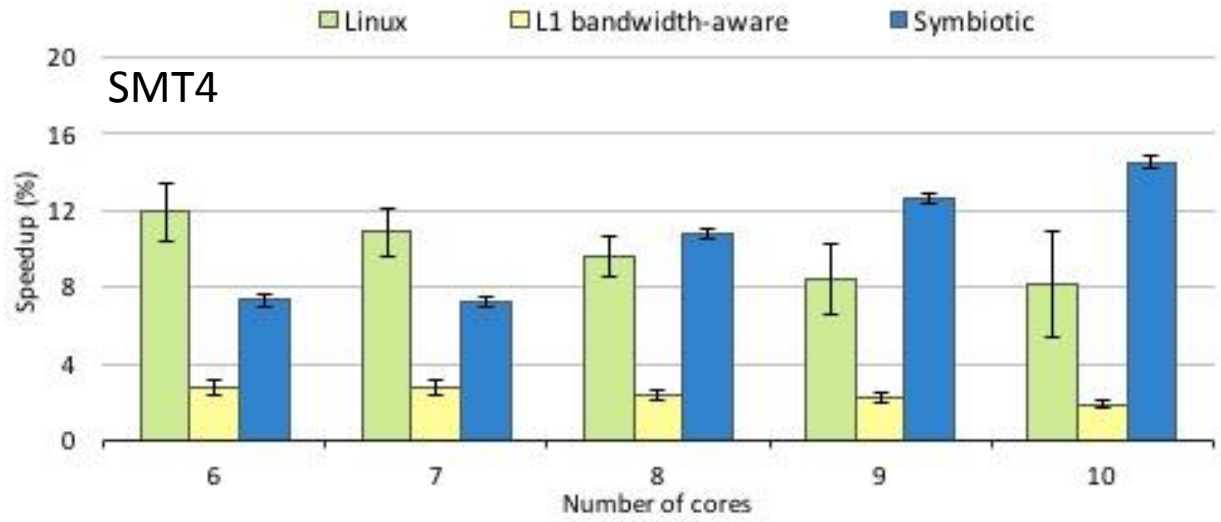
Frequency matrices of job coschedules for two workloads

- The darker the color, the more frequently the couple is scheduled together on an SMT core

Experimental evaluation

System throughput

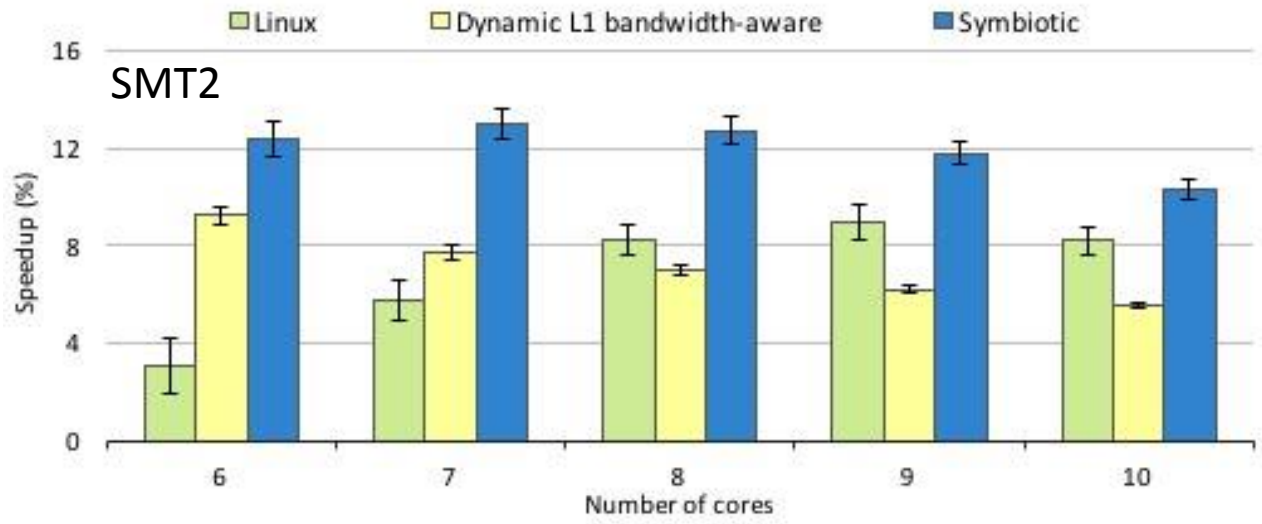




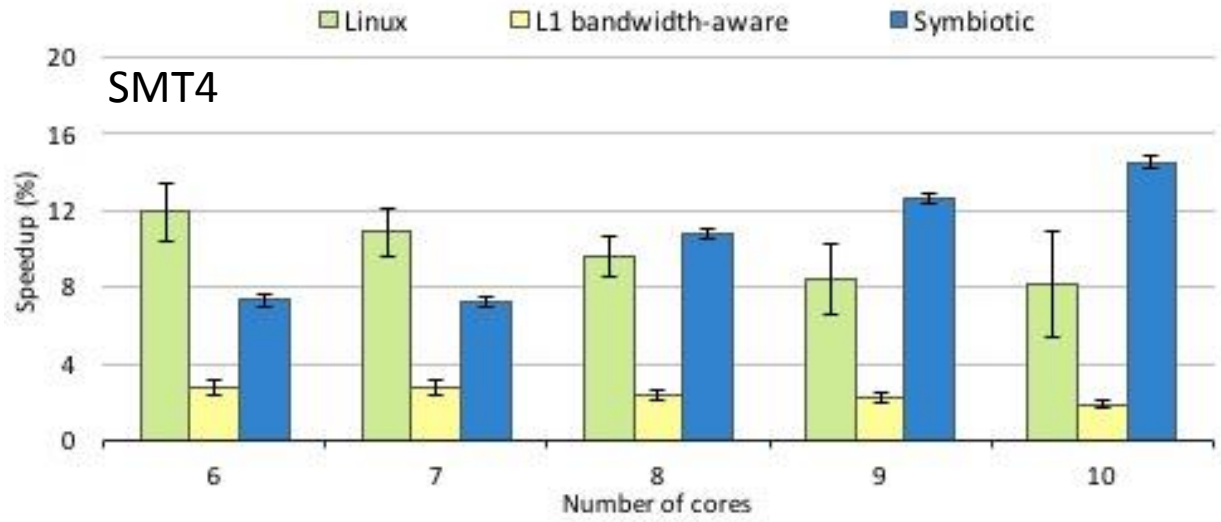
Avg speedup w.r.t. Linux
5.2%

Experimental evaluation

System throughput



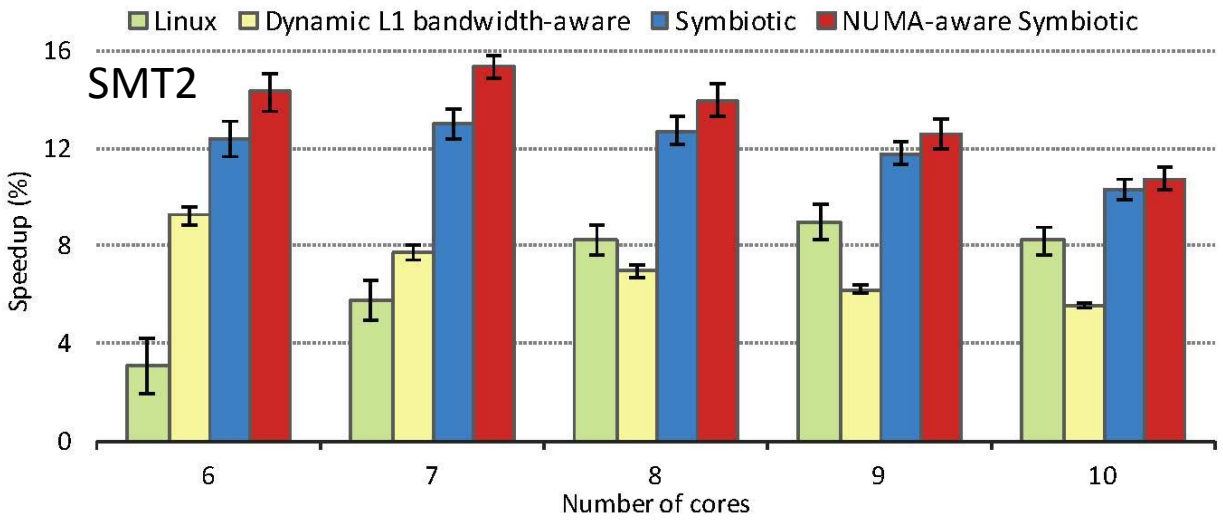
Avg speedup w.r.t. Linux
5.2%



Avg speedup w.r.t. Linux
0.7%

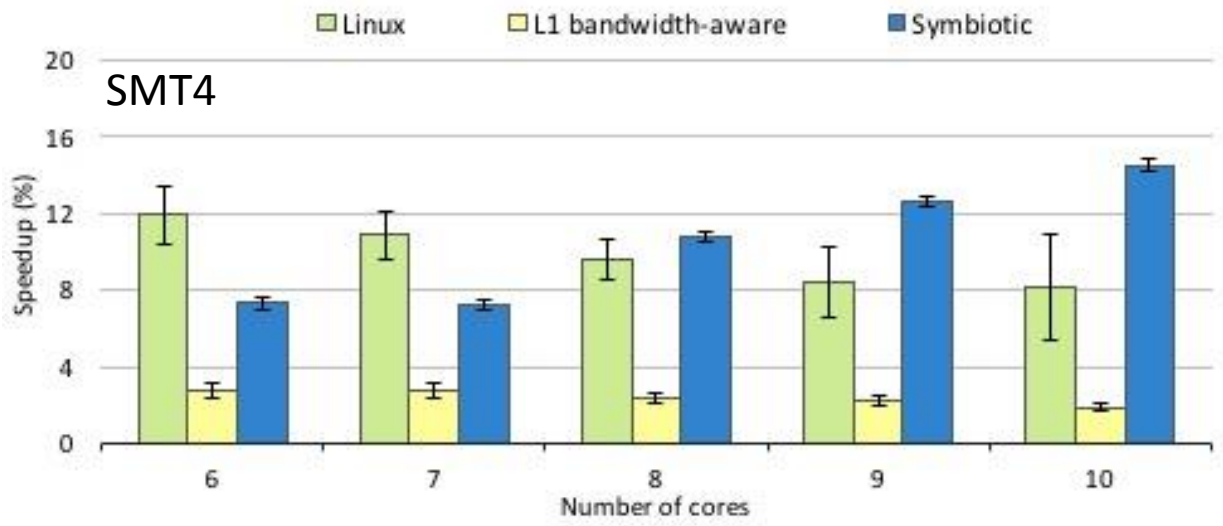
Experimental evaluation

System throughput



Avg speedup w.r.t. Linux
5.2%

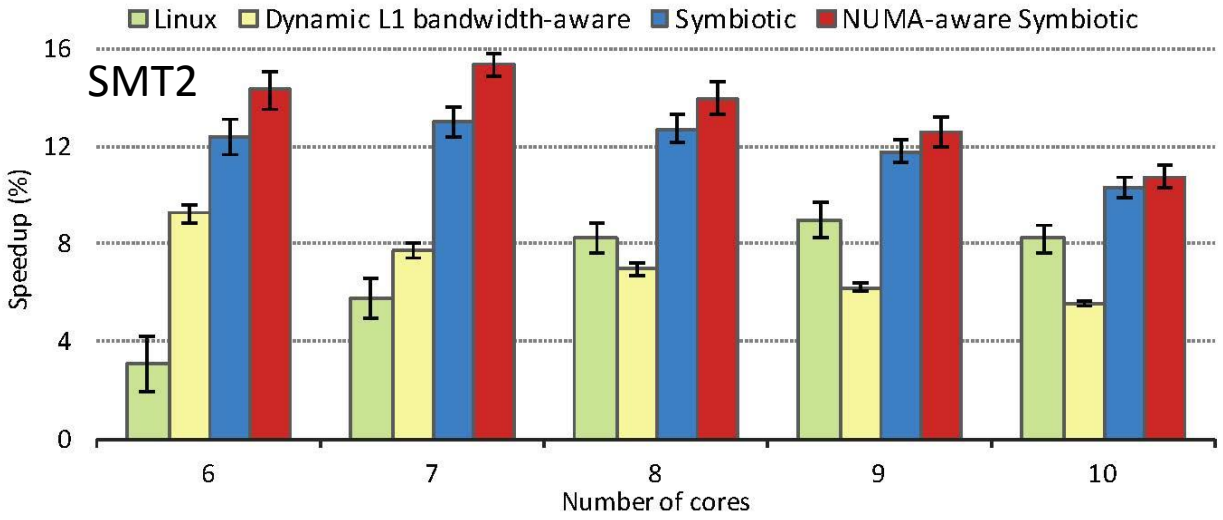
Avg speedup w.r.t. Linux
6.7%



Avg speedup w.r.t. Linux
0.7%

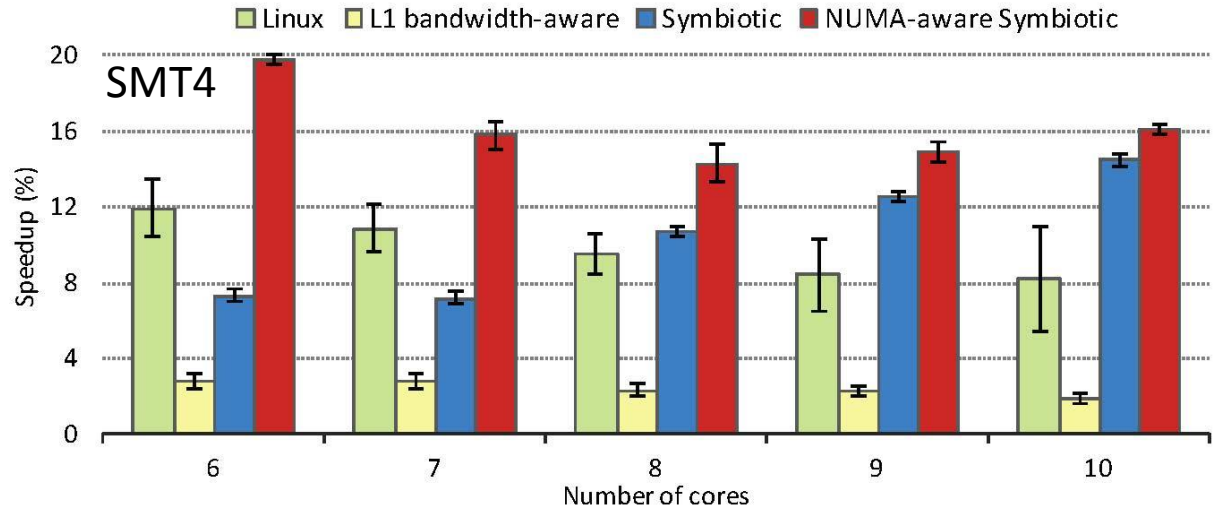
Experimental evaluation

System throughput



Avg speedup w.r.t. Linux
5.2%

Avg speedup w.r.t. Linux
6.7%



Avg speedup w.r.t. Linux
0.7%

Avg speedup w.r.t. Linux
5.9%

Outline

I. Bandwidth-Aware Scheduling on Multicores

II. Bandwidth-Aware Scheduling on SMT Multicores

III. Progress-Aware Scheduling on SMT Multicores

IV. Symbiotic Job Scheduling on the IBM POWER8

V. Conclusions

I. Main contribution

II. Future directions

III. Publications

Conclusions
Main contributions

1st contribution

We propose the **memory-hierarchy bandwidth-aware scheduling** algorithm for single-threaded multicores

Bandwidth contention can rise at any level of the memory hierarchy

The proposed algorithm:

- Minimizes cache-hierarchy bandwidth contention balancing the requests that each cache receives
- Processes most sensitive to contention are scheduled in favorable scenarios

Speeded of the turnaround time 6.6% with respect to Linux

Conclusions
Main contributions

2nd contribution

We propose the **bandwidth-aware scheduling** algorithm for SMT multicores

L1 caches become shared

- Bandwidth contention at the L1 caches rises

The proposed algorithm

- Uses the **dynamic L1 bandwidth-aware process allocation policy** (minimizes L1 bandwidth contention)
- Mitigates main memory bandwidth contention without preliminary information

Improves average IPC by 4.6% with respect to Linux

Conclusions
Main contributions

3rd contribution

We propose **progress-aware scheduling algorithms to deal with fairness** in SMT multicores

Approach to measure the progress of the processes at runtime

- Based on estimates of the IPC_{alone} in low-contention schedules

Proposed progress-aware algorithms:

- *Fair:* minimizes unfairness
- Perf&Fair: reduces unfairness and increases performance

Fair reduces Linux unfairness to a third

Perf&Fair reduces Linux unfairness to a half and improves turnaround time by 5.6%

Conclusions

Main contributions

4th contribution

We propose a **symbiotic job scheduler**

SMT interference model based on CPI stacks

- Predicts the performance of any combination of applications
- Considers contention in all the shared resources of SMT cores

Scheduling modeled as a graph problem

- SMT2: optimal schedule with the blossom algorithm
- SMT4: near optimal schedule with the hierarchical perfect matching algorithm

STP increases by 6.7% (SMT2) and 5.9% (SMT4) over Linux

Conclusions

Future directions

Scheduling is going to be a hot topic in the next years

- Scheduling parallel applications
  - Optimal number of threads, cores, or SMT mode for each application
  - Concurrent execution of multiple parallel applications
- Scheduling to make the best use of new architectures and features
  - Cache partitioning capabilities
  - Tune prefetching
  - Heterogeneous systems
- Scheduling in mobile devices
  - Improving performance/watt
  - Real time constraints

Conclusions

Publications

International journals:

- J. Feliu, S. Petit, J. Sahuquillo, and J. Duato. Cache-Hierarchy Contention Aware Scheduling in CMPs. *IEEE Transactions on Parallel and Distributed Systems* (**TPDS**), volume 25, issue 3, pages 581-590, 2014

- J. Feliu, J. Sahuquillo, S. Petit, and J. Duato. Bandwidth-Aware On-Line Scheduling in SMT Multicores. *IEEE Transactions on Computers* (**TC**), volume 65, issue 2, pages 422-434, 2016

- J. Feliu, J. Sahuquillo, S. Petit, and J. Duato. Perf&Fair: a Progress-Aware Scheduler to Enhance Performance and Fairness in SMT Multicores. *IEEE Transactions on Computers* (**TC**), DOI:10.1109/TC.2016.2620977

Conclusions

Publications

International conferences:

- J. Feliu, S. Eyerman, J. Sahuquillo, and S. Petit. “Symbiotic Job Scheduling on the IBM POWER8”. In *Proceedings of the 22nd International Symposium on High Performance Computer Architecture* (**HPCA**), pages 669-680, 2016.
- J. Feliu, J. Sahuquillo, S. Petit, and J. Duato. “L1-Bandwidth Aware Thread Allocation in Multicore SMT Processors”. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques* (**PACT**), pages 123-132, 2013.
- J. Feliu, J. Sahuquillo, S. Petit, and J. Duato. “Addressing Fairness in SMT Multicores with a Progress-Aware Scheduler”. In *Proceedings of the 29th International Parallel and Distributed Processing Symposium* (**IPDPS**), pages 187-196, 2015.
- J. Feliu, J. Sahuquillo, S. Petit, and J. Duato. “Understanding Cache Hierarchy Contention in CMPs to Improve Job Scheduling”. In *Proceedings of the 26th International Parallel and Distributed Processing Symposium* (**IPDPS**), pages 508-519, 2012.
- J. Feliu, J. Sahuquillo, S. Petit, and J. Duato. “Addressing Bandwidth Contention in SMT Multicores Through Scheduling”. In *Proceedings of the 28th International Conference on Supercomputing* (**ICS**), page 167, 2014.
- J. Feliu, J. Sahuquillo, S. Petit, and J. Duato. “Using Huge Pages and Performance Counters to Determine the LLC Architecture”. In *Proceedings of the International Conference on Computational Science* (**ICCS**), pages 2557-2560, 2013.

Conclusions
Publications

International summer school:

- J. Feliu, S. Eyerman, J. Sahuquillo, and S. Petit. “Improving Throughput on the IBM POWER8 with a Symbiotic Scheduler”. In *Proceedings of the 12th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems* (**ACACES**), pages 201-204, Fiuggi, Italy, 2016.

Conclusions
Publications

Domestic conferences:

- J. Feliu, S. Eyerman, J. Sahuquillo, and S. Petit. “Planificación Simbiótica de Procesos en el IBM POWER8”. In *Actas de las XXVII Jornadas de Paralelismo* (**JP**), pages 315-324, Salamanca, Spain, 2016.

- J. Feliu, J. Sahuquillo, S. Petit, and J. Duato. “Planificación Orientada a Equidad Considerando el Progreso en Multinúcleos SMT”. In *Actas de las XXVI Jornadas de Paralelismo* (**JP**), pages 118-126, Córdoba, Spain, 2015.

- J. Feliu, J. Sahuquillo, S. Petit, and J. Duato. “Ubicación de Procesos Considerando el Ancho de Banda de L1 en Procesadores Multinúcleo SMT”. In *Actas de las XXV Jornadas de Paralelismo* (**JP**), pages 343-352, Valladolid, Spain, 2014.

- J. Feliu, J. Sahuquillo, S. Petit, and J. Duato. “Planificación Considerando Degradación de Prestaciones por Contención”. In *Actas de las XXIV Jornadas de Paralelismo* (**JP**), pages 62-67, Madrid, Spain, 2013.

- J. Feliu, J. Sahuquillo, S. Petit, and J. Duato. “Planificación Considerando el Ancho de Banda de la Jerarquía de Cache”. In *Actas de las XXIII Jornadas de Paralelismo* (**JP**), pages 472-477, Elx, Spain, 2012.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA

Contention-Aware Scheduling for SMT Multicore Processors

Author:

Josué Feliu Pérez

Advisors:

Julio Sahuquillo Borrás

Salvador V. Petit Martí

València, February 22, 2017

Methodology

Process selection methodology

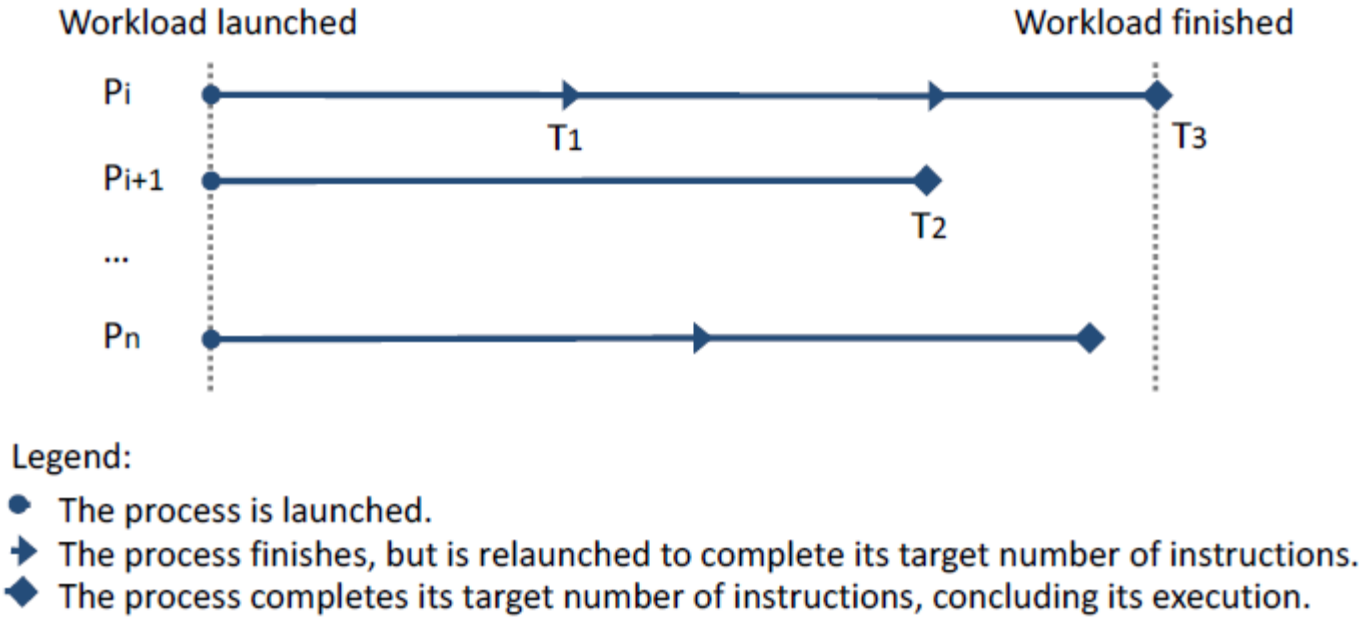


FIGURE 3.5: Timing chart under the process selection methodology.

Methodology

Process allocation methodology

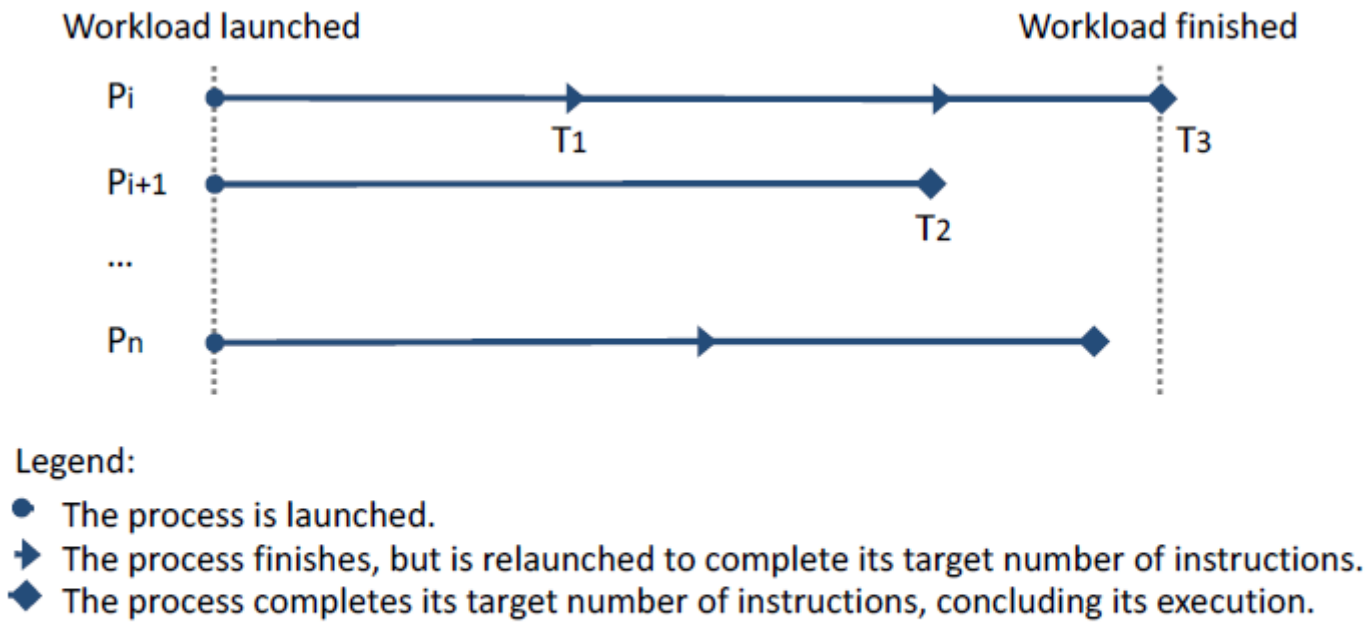
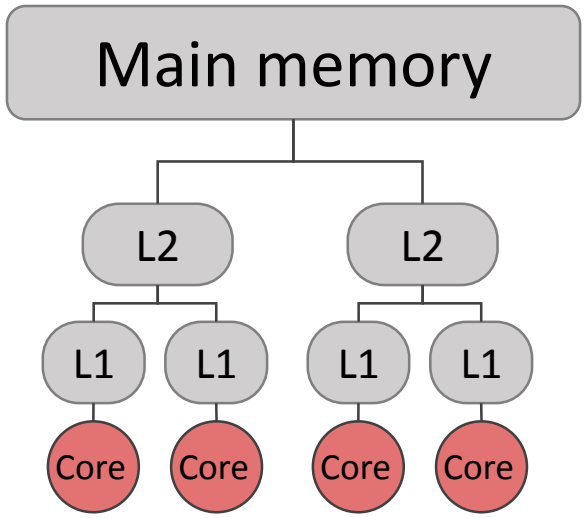


FIGURE 3.6: Timing chart under the process selection methodology.

1st contribution - Experimental evaluation

Setup

- Intel Xeon X3320
  - 4 cores, no SMT support
  - Two L2 caches, shared by a pair of cores
- 10 multiprogram workloads
  - 8 applications per workload
  - Varying bandwidth requirements at main memory and the L2 cache
  - MM BW between 20 and 40 t/usec
- Metric
  - Turnaround time



- Evaluated schedulers:
  - Linux, default Completely Fair Scheduler (CFS)
  - Baseline main memory bandwidth-aware scheduler (MMaS)
  - Memory-hierarchy bandwidth-aware scheduler (MHaS)
  - IPC-degradation memory-hierarchy bandwidth-aware scheduler (IDaS)

2nd contribution - Experimental evaluation

Setup

- 6-core Intel Xeon E5645
  - Supports simultaneous multithreading (SMT)

Process allocation

- **Multiprogram workloads**
  - **2 benchmarks per core, from 2 to 6 cores**
  - **Balanced and non-balanced mixes**
- Metrics
  - Average IPC
  - Harmonic mean of IPC speedup
- Evaluated algorithms
  - Random
  - Linux
  - Dynamic L1 bandwidth-aware
  - Static L1 bandwidth-aware

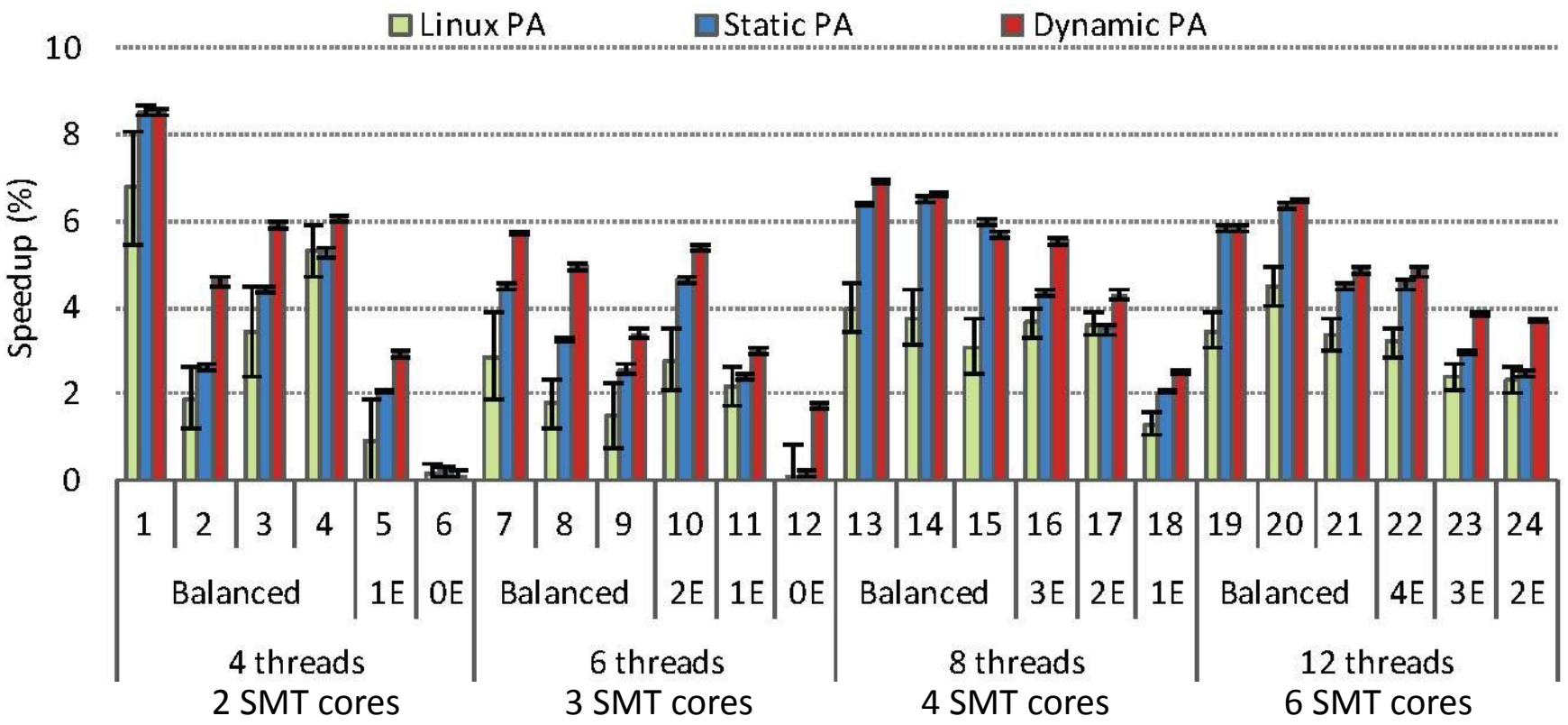
    (Uses de offline average bandwidth)

Scheduler

- **Multiprogram workloads**
  - **Double number of benchmark than hardware threads**
  - **Random workload**
- Metrics
  - Average IPC
  - Harmonic mean of IPC speedup
- Evaluated algorithms

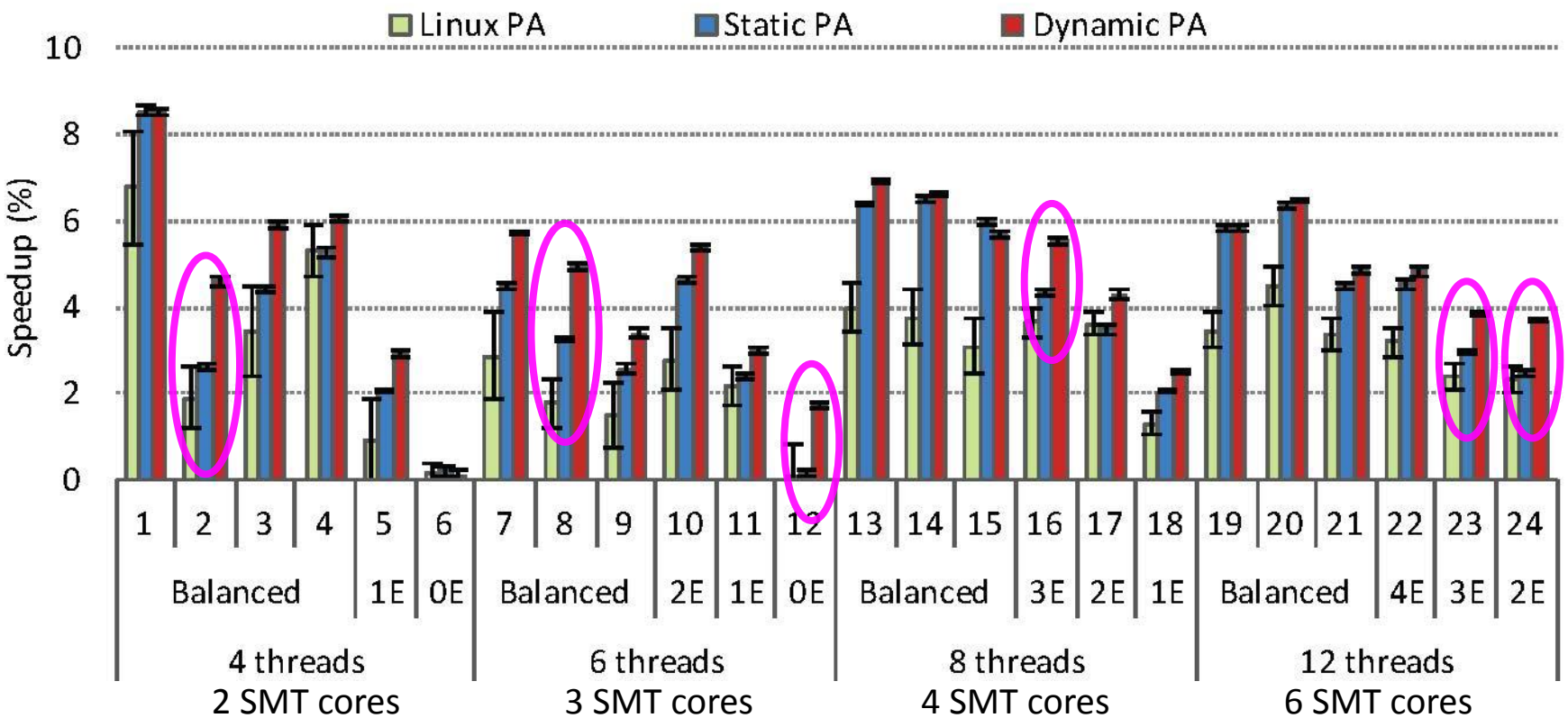
Experimental evaluation

Dynamic process allocation policy – Harmonic mean of IPC speedups



Experimental evaluation

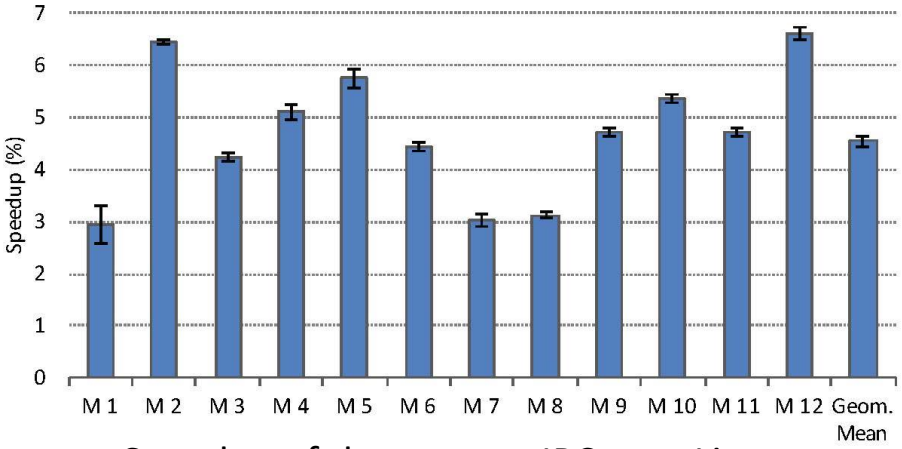
Dynamic process allocation policy – Harmonic mean of IPC speedups



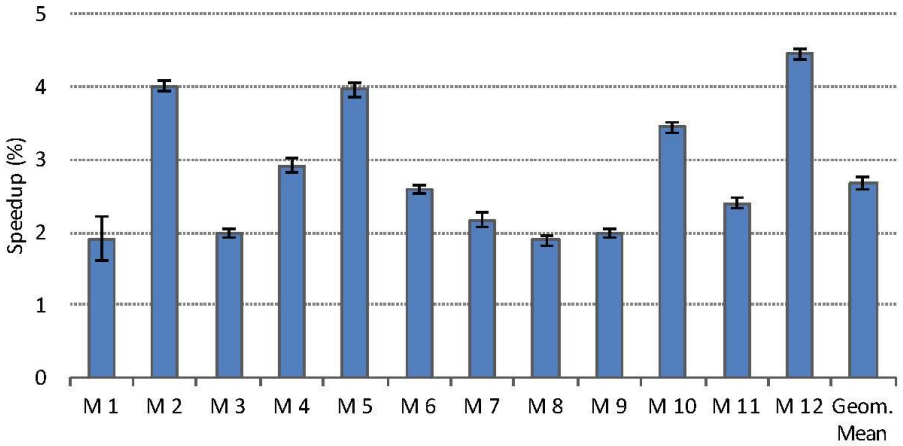
Higher between the static and dynamic policies

Experimental evaluation

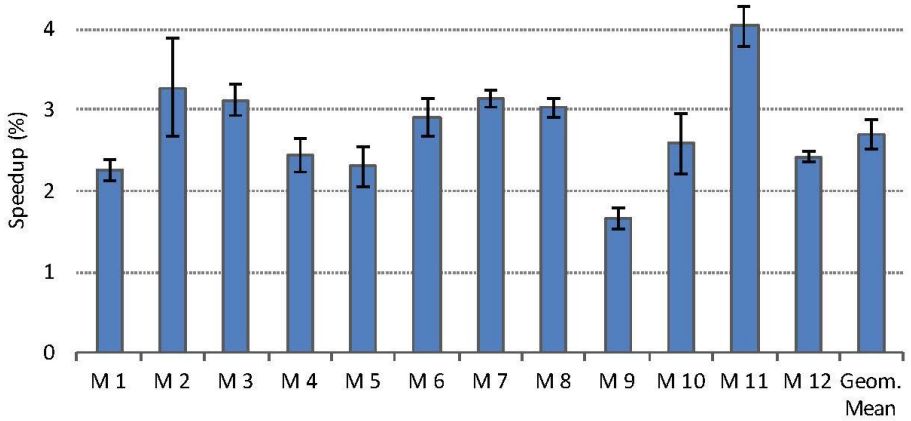
SMT bandwidth-aware scheduler – Speedups



Speedup of the average IPC over Linux



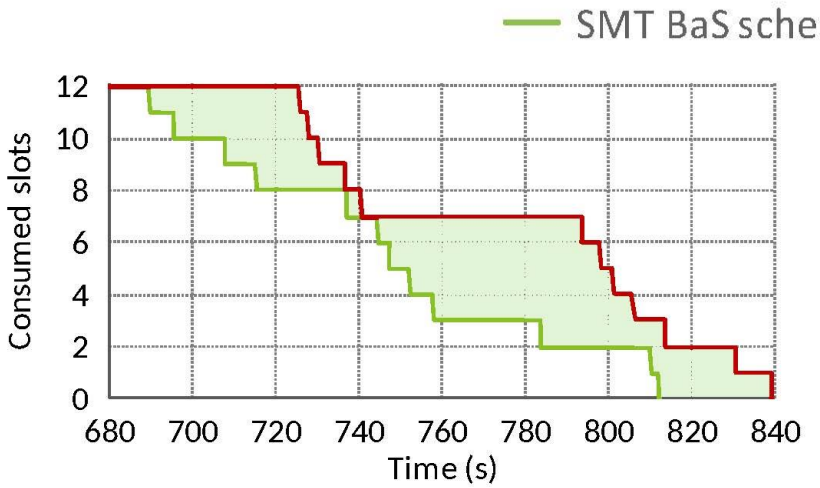
Speedup of the harmonic mean of IPC speedups over Linux



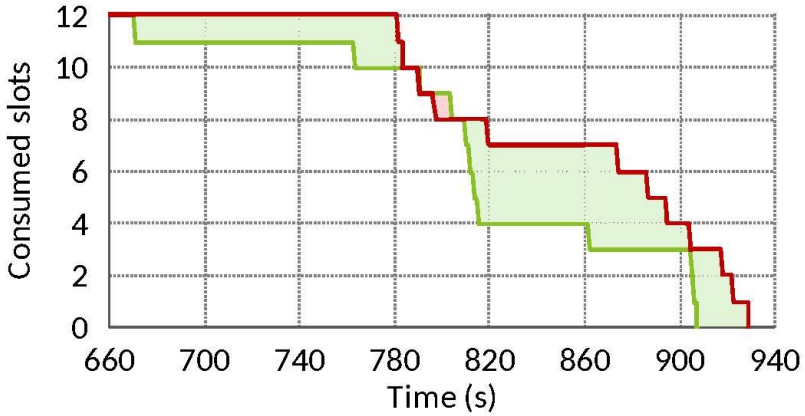
Speedup of the turnaround time over Linux

Experimental evaluation

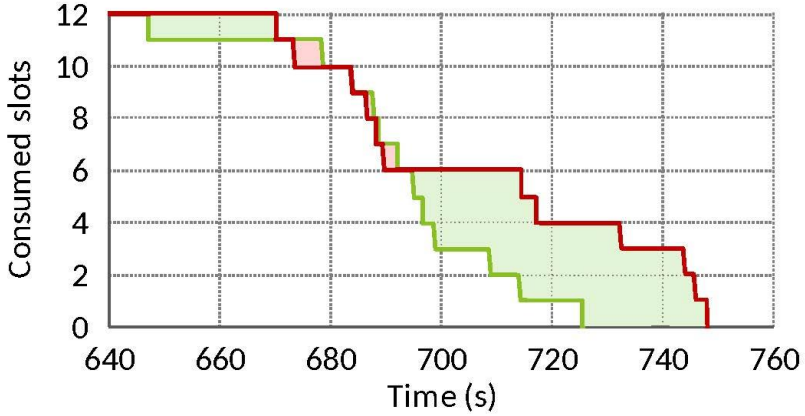
SMT bandwidth-aware scheduler – Consumed slots



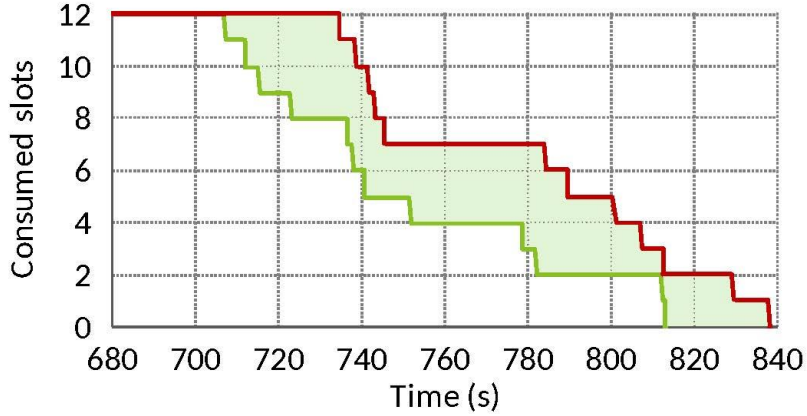
Consumed slots in workload 2



Consumed slots in workload 5



Consumed slots in workload 8



Consumed slots in workload 10

3rd contribution - Experimental evaluation

Setup

- 6-core Intel Xeon E5645
  - Supports simultaneous multithreading (SMT)

	Progress-aware schedulers
Multiprogram workloads	• **24 applications per workload** • **Average TRMM of the workloads from 50 to 130 t/usec**
Metrics	• Turnaround time • Unfairness
Evaluated algorithms	• Random • Linux • SMT bandwidth-aware (*Perf*) • Progress-aware *Fair* • Progress-aware Perf&Fair

Experimental evaluation

Progress evolution over time

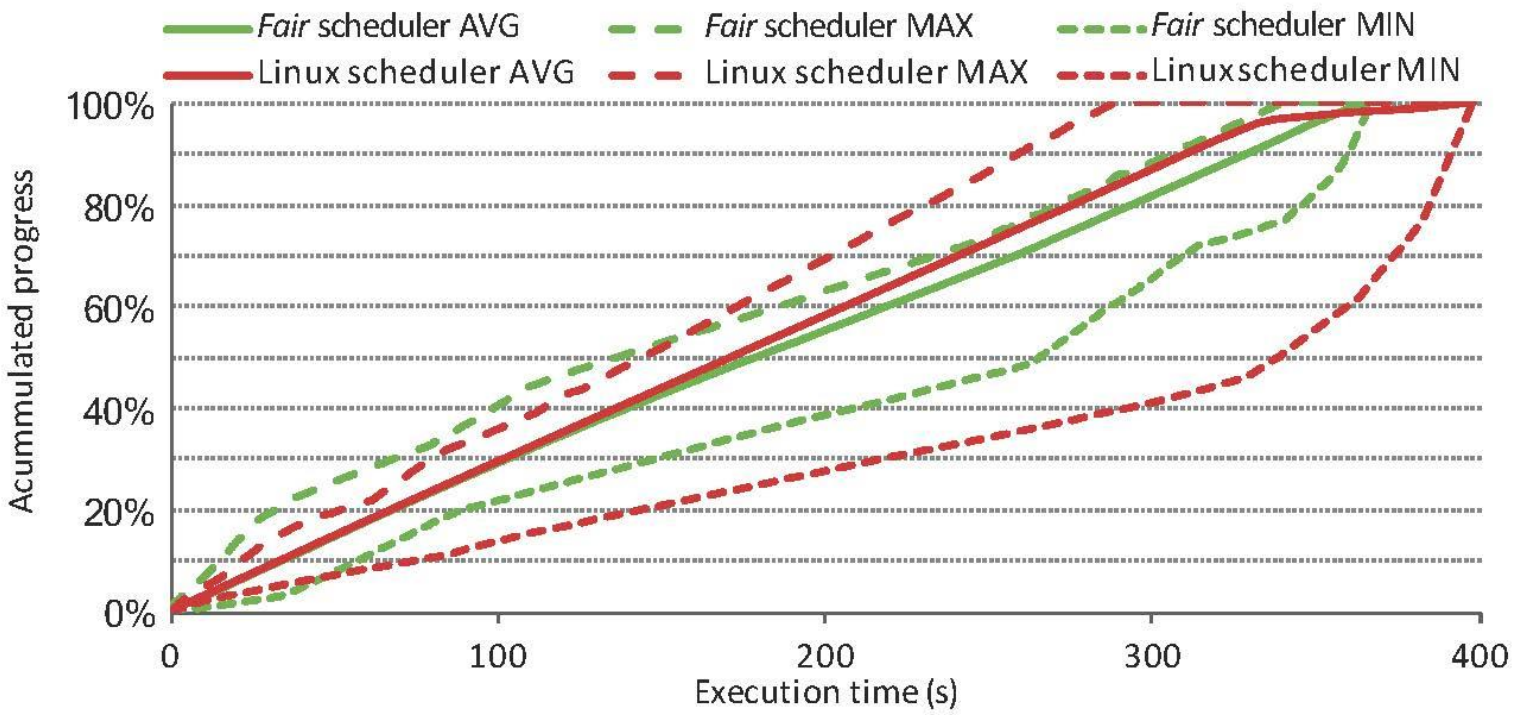


Fig X. Dynamic progress of processes in mix M7 with the Fair and Linux schedulers.

Experimental evaluation

Progress evolution over time

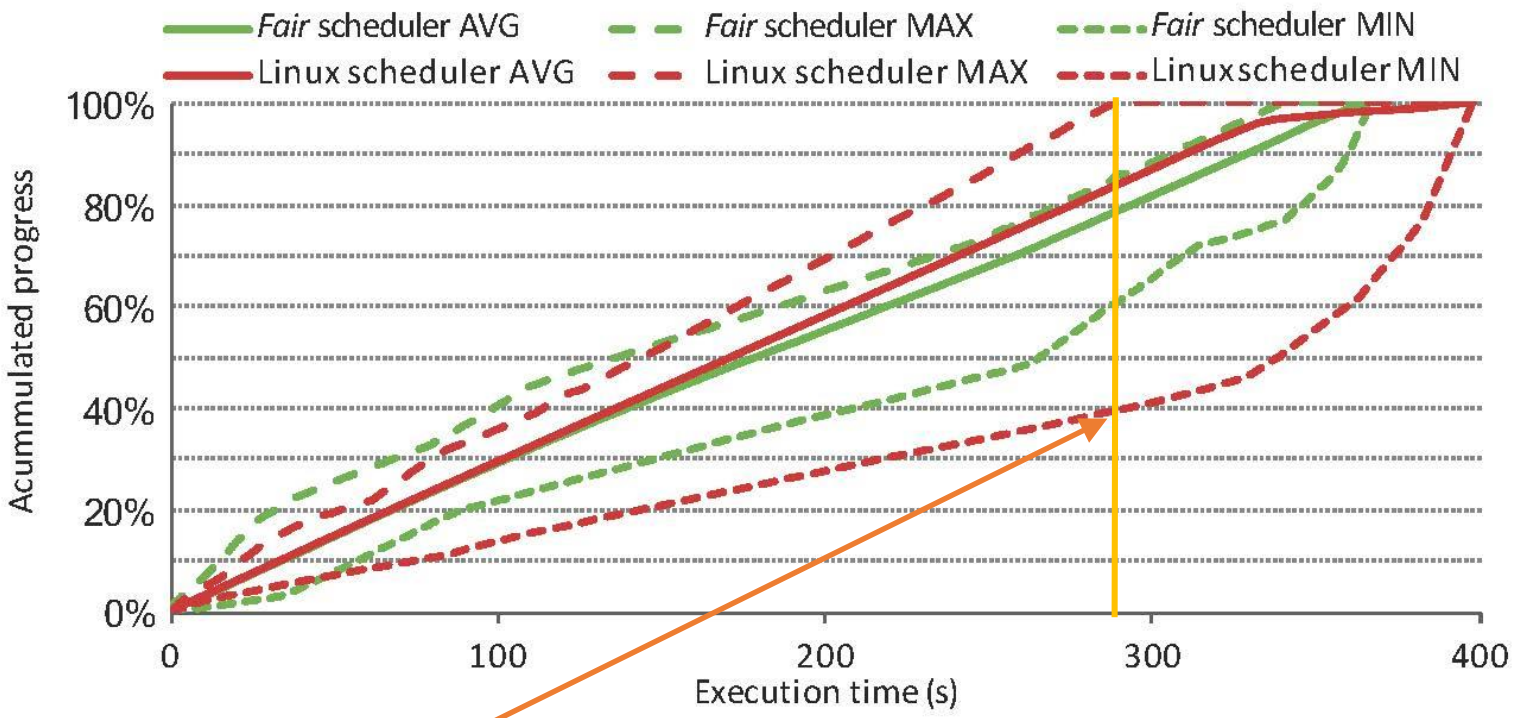


Fig X. Dynamic progress of processes in mix M7 with the Fair and Linux schedulers.

Minimum progress by 40%

Experimental evaluation

Progress evolution over time

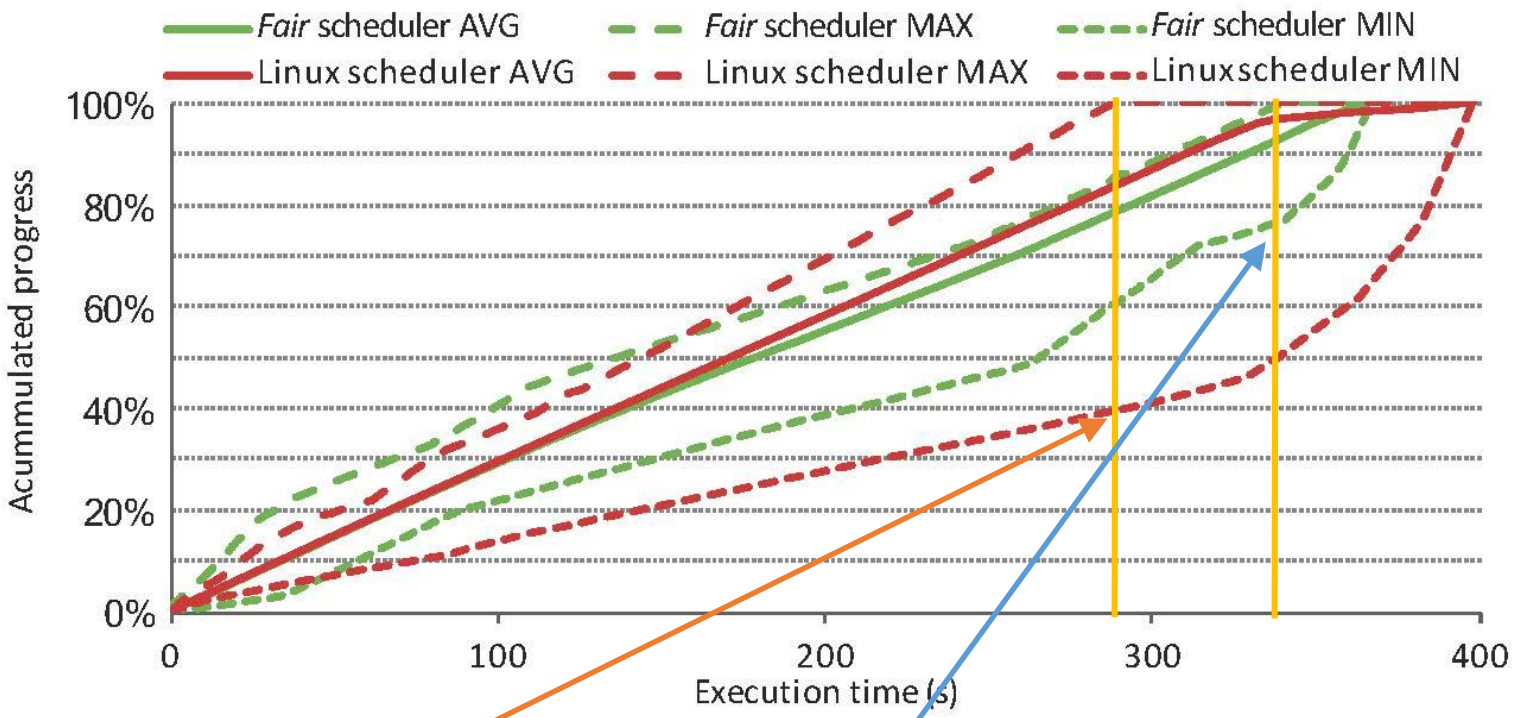


Fig X. Dynamic progress of processes in mix M7 with the Fair and Linux schedulers.

Minimum progress by 40%

Minimum progress by 75%

Experimental evaluation

Progress evolution over time

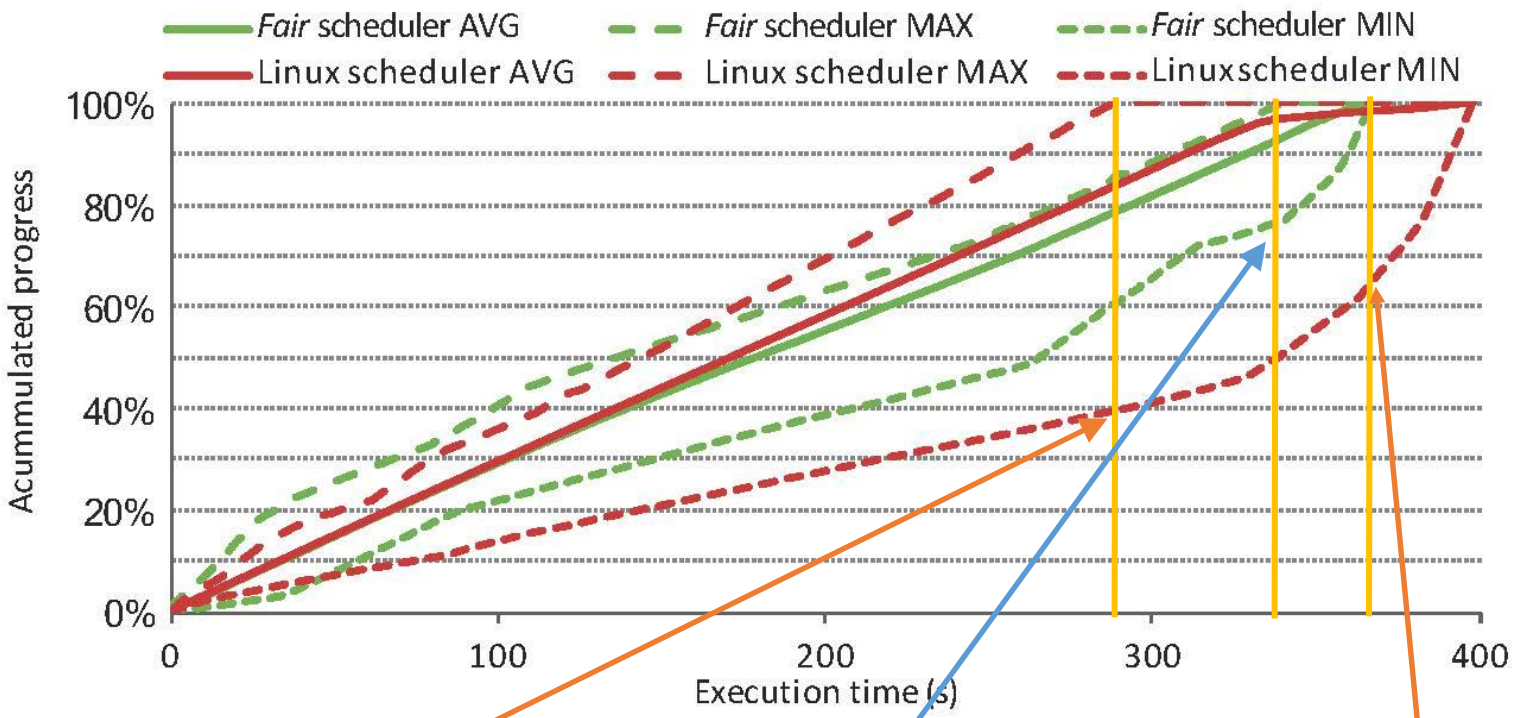


Fig X. Dynamic progress of processes in mix M7 with the Fair and Linux schedulers.

Minimum progress by 40%

Minimum progress by 75%

Minimum progress by 68%

Experimental evaluation

System throughput

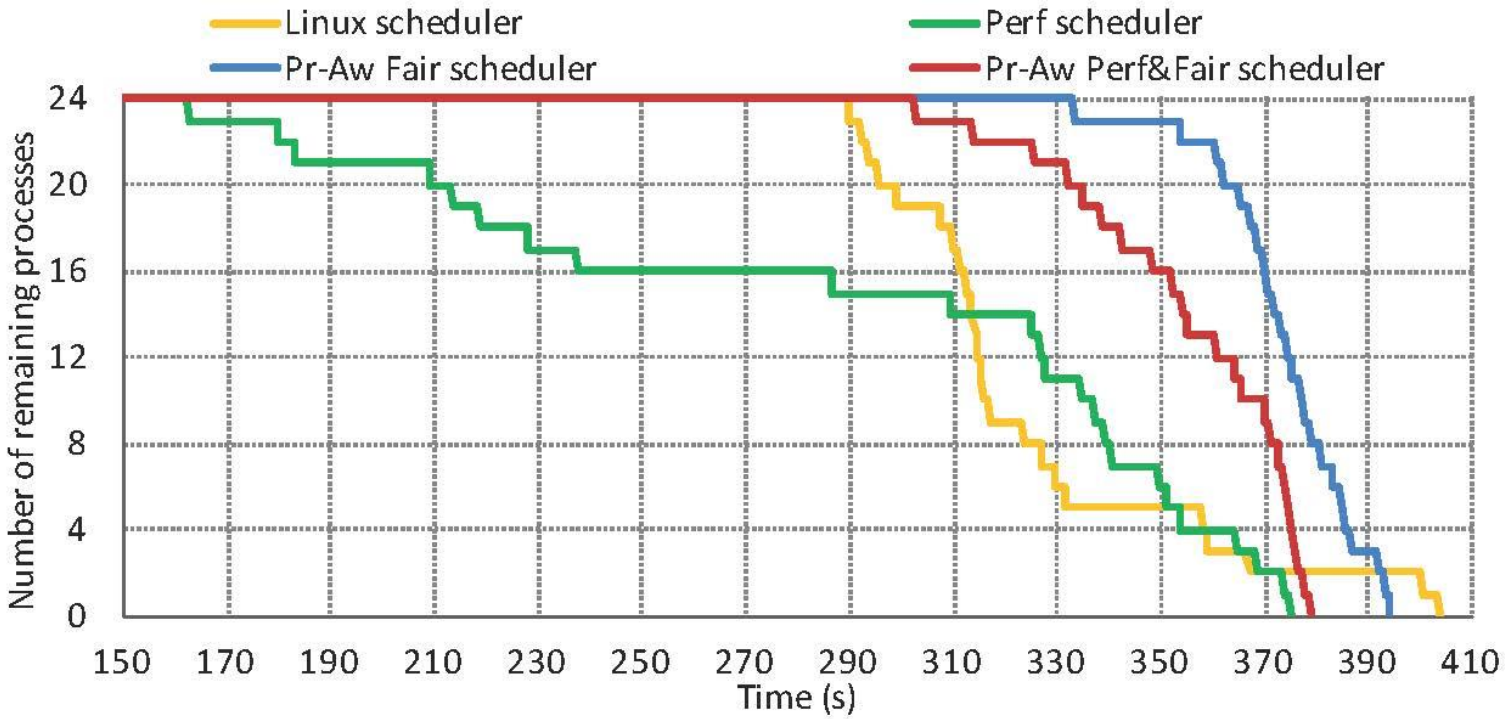


Fig X. Number of remaining processes along the execution of mix 9 with the studied schedulers.

Experimental evaluation

System throughput

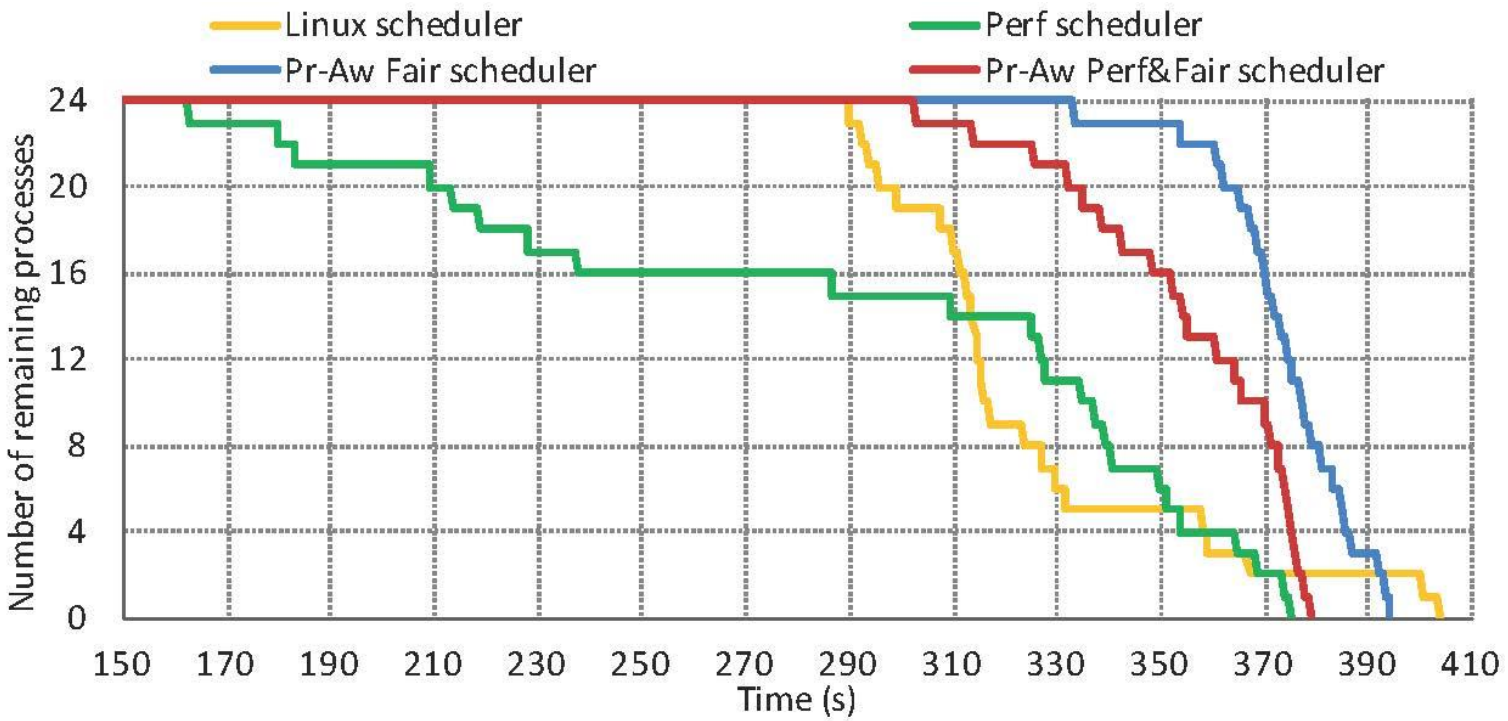


Fig X. Number of remaining processes along the execution of mix 9 with the studied schedulers.

Perf
- Best turnaround time
- Worst fairness

Fair
- "Bad" turnaround time
- Best fairness

Experimental evaluation

System throughput

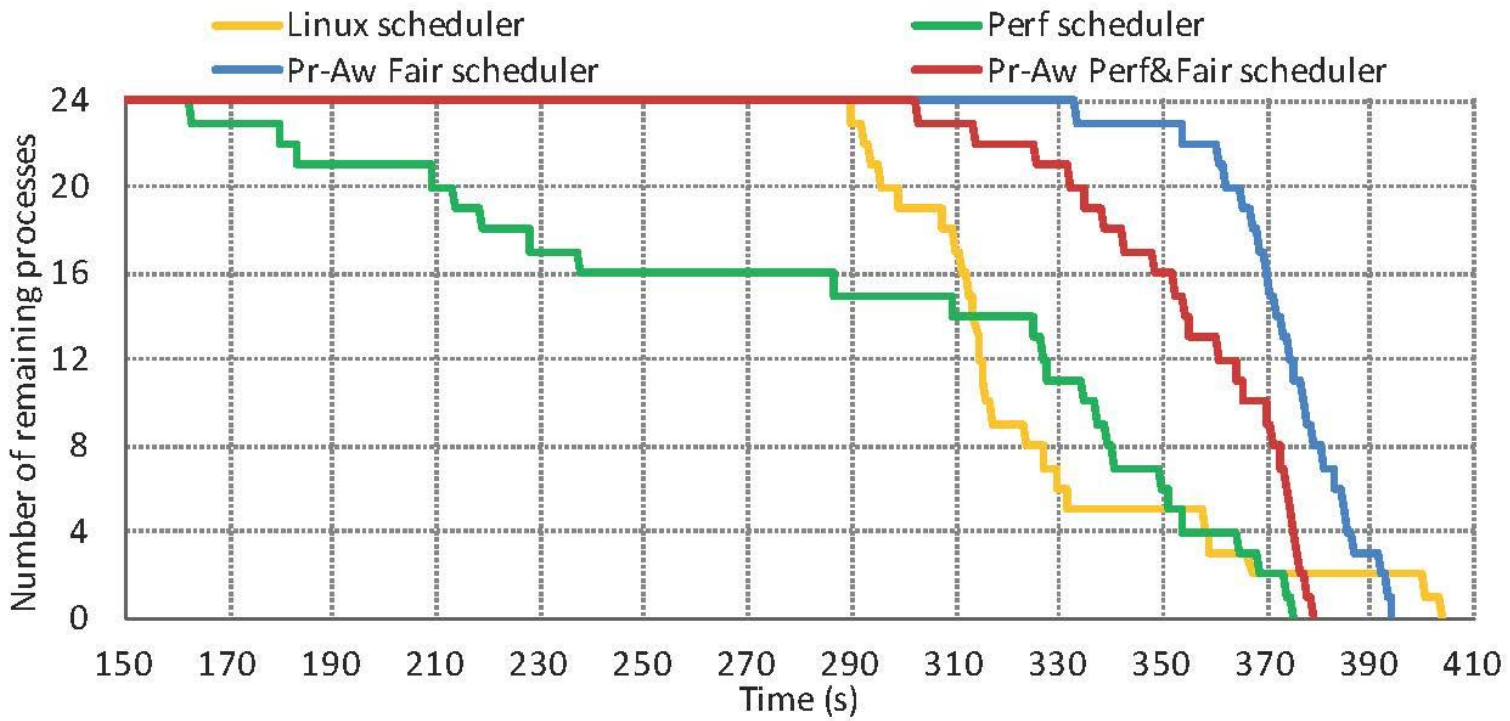


Fig X. Number of remaining processes along the execution of mix 9 with the studied schedulers.

Perf
- Best turnaround time
- Worst fairness

Fair
- "Bad" turnaround time
- Best fairness

Perf&Fair
- Best turnaround time
- Good fairness

SMT interference-aware scheduling

Reduction of the CPI stack components

- 45 events form the full CPI stack of the IBM POWER8
- 6 thread-level counters are implemented (4 programmable)
  - Structural conflicts on some events that cannot be measured together
  - 19 time slices required to build the full CPI stack
- Unacceptable for scheduling
  - Obtaining an updated CPI stack is not possible
- Fortunately, the CPI stack model is build hierarchically
  - Top level with 5 components
  - The model accuracy is reduced, but it has lower complexity and use updated CPI stacks

4th contribution - Experimental evaluation

Setup

- 10-core IBM POWER8
- 100 random multiprogram workloads
  - From 6- to 10-core workloads
  - 2 and 4 applications per core for SMT2 and SMT4
- Metrics
  - System throughput (STP)
  - Average normalized turnaround time (ANTT)
- Evaluated schedulers:
  - Random
  - Linux, default Completely Fair Scheduler (CFS)
  - Dynamic L1 bandwidth-aware scheduler
  - Symbiotic scheduler
  - NUMA-aware Symbiotic scheduler

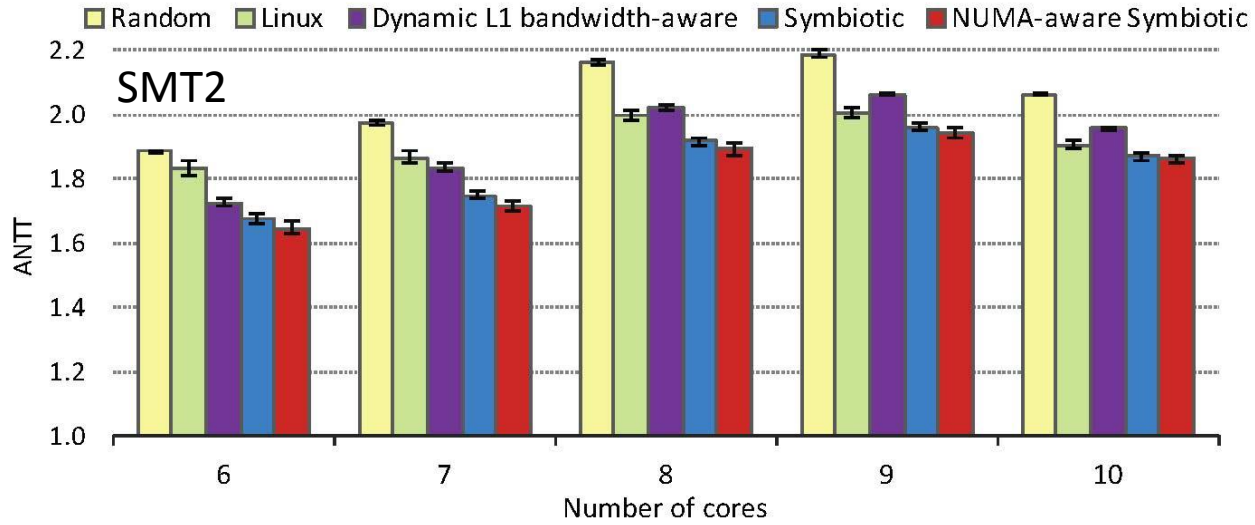
Experimental evaluation

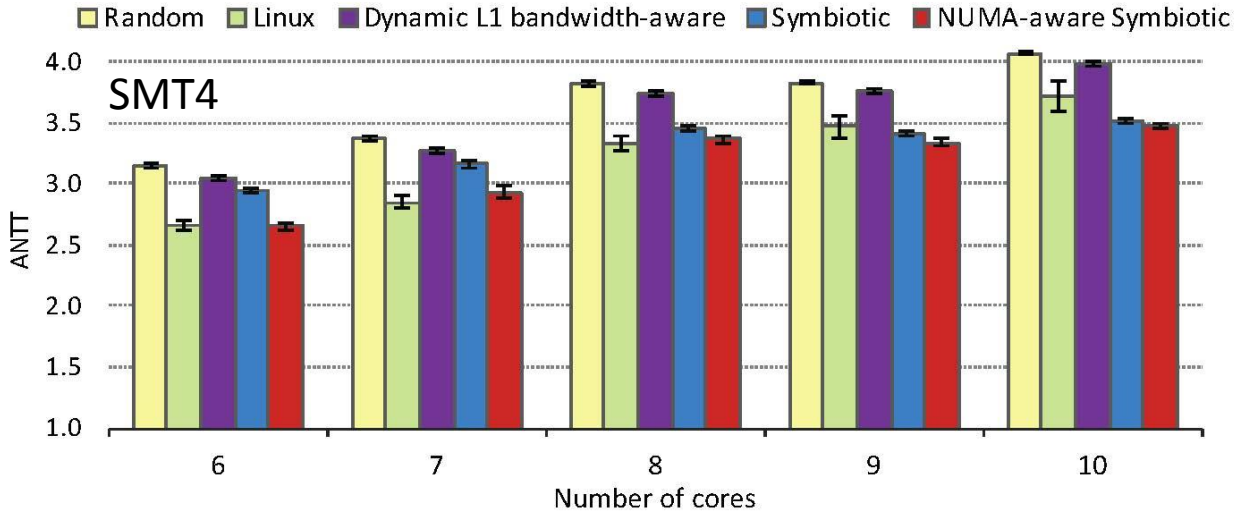
Setup

- 10-core IBM POWER8
- 100 random multiprogram workloads
  - From 6- to 10-core workloads
  - 2 and 4 applications per core for SMT2 and SMT4
- Metrics
  - System throughput (STP)
  - Average normalized turnaround time (ANTT)
- Evaluated schedulers:
  - Random
  - Linux, default Completely Fair Scheduler (CFS)
  - Dynamic L1 bandwidth-aware scheduler
  - Symbiotic scheduler
  - NUMA-aware Symbiotic scheduler

Experimental evaluation

Average normalized turnaround time





Symbiosis patterns
SMT4

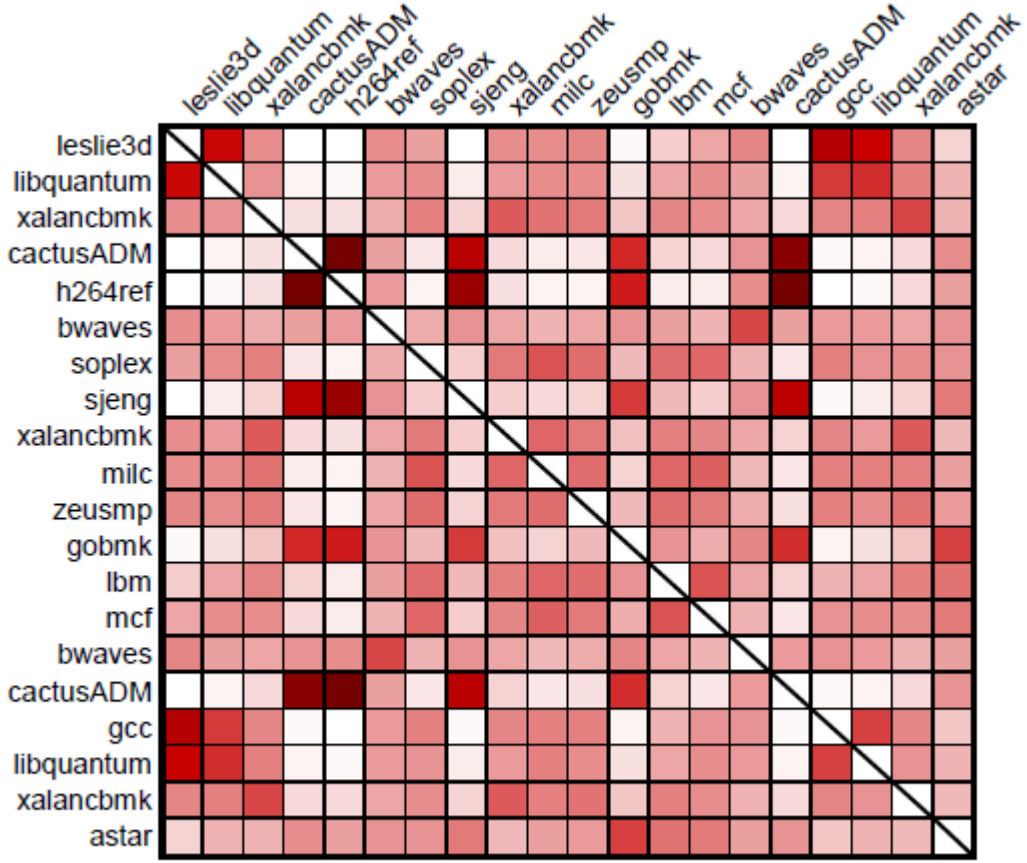


FIGURE 7.12: Frequency matrix for a 5-core workload running in SMT4 mode.