Addressing Fairness in SMT Multicores with a Progress-Aware Scheduler

J. Feliu, J. Sahuquillo, S. Petit, and J. Duato

Universitat Politècnica de València

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

GAP Parallel Architectures Group

May 26th, 2015

Hyderabad, India

Introduction

- Simultaneous multithreading (SMT) multicores dominate the high-performance microprocessors market

- Two levels of shared resources:
  - Inter-core: uncore shared part of the systems (CMPs)
  - Intra-core: inside the core (SMTs)
- Applications compete among themselves at runtime for the shared resources

- Designing fair resource-sharing policies is challenging:
  - The applications present different requirements for the multiple shared resources
  - The shared use of a resource affects differently the performance of distinct applications

Introduction

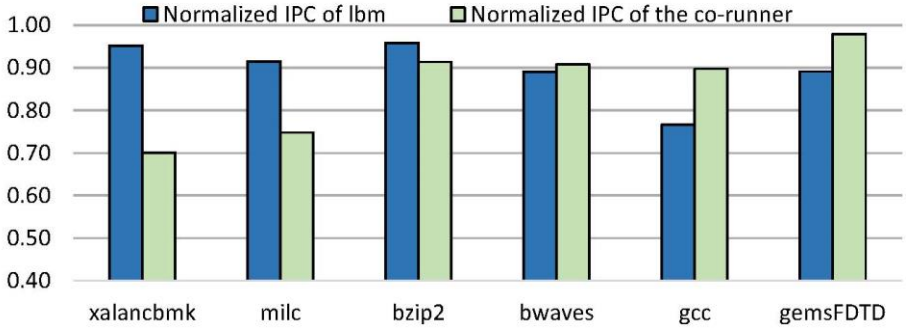
- We consider the system to be fair when all the running processes experience the same slowdown with respect to their isolated execution

- Unfairness causes important undesirable behaviors:
  - Complicates priority based scheduling or QoS
  - Difficult guaranteeing worst-case execution time (WCET)
  - Reduces performance predictability
  - Leads to unfairness billings in cloud computing services
  - Enables denial of service attacks

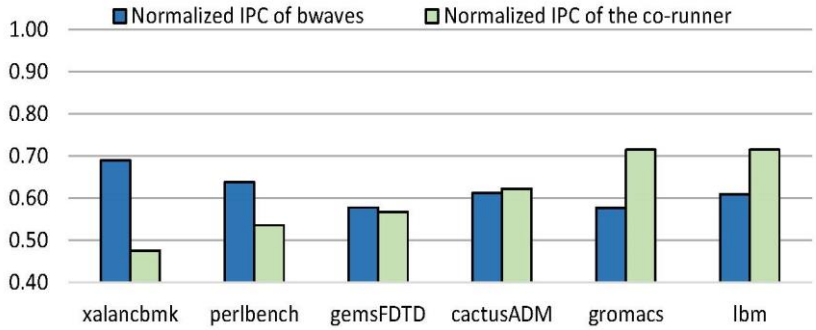
- Processors with heterogeneous cores or hardware accelerators will magnify the unfairness issues

Introduction

Are current SMT multicores fair?



(a) Running with *lbm* on different cores



(b) Running with *bwaves* on the same core

Fig 1. Normalized IPC of the benchmarks with respect to isolated execution

- Simple experiment running pairs of benchmarks:
  - (a) On different cores: inter-core interferences
  - (b) On the same core: intra-core interferences
- The processes progress at different paces depending on the co-runner
- Unfairness above 30%

Introduction

Key contributions

- We propose a way to accurately estimate the standalone performance of the processes in SMT multicores
  - Running in multiprogrammed workloads

- We can estimate the progress achieved by the processes
  - With respect to their isolated execution

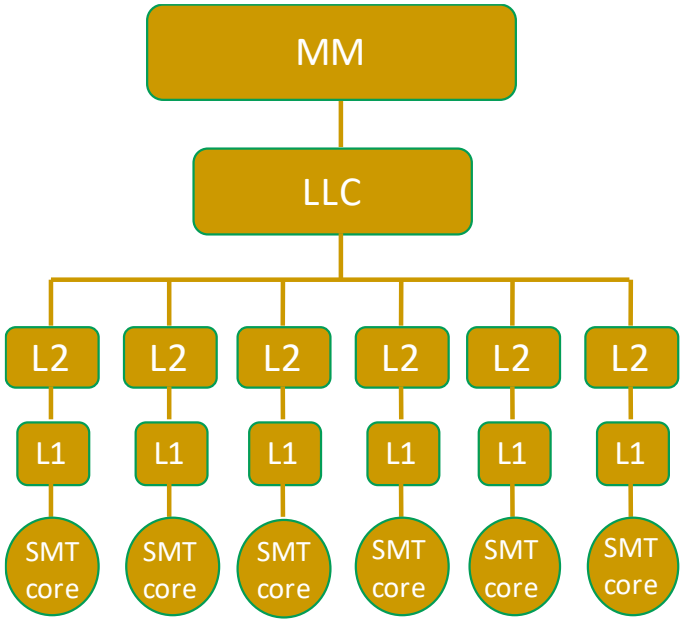
- We present the Progress-Aware scheduler to maximize fairness

Outline

- Introduction
- **Experimental platform**
- Estimating progress
- Progress-Aware scheduling
- Experimental evaluation
- Conclusions

Experimental platform

- All the experiments are performed on a real system:
  - Intel Xeon E5645 (Westmere-EP microarchitecture)
  - Linux with kernel 3.11.4
- SPEC CPU2006 benchmarks
  - Workload with more processes than execution contexts
- Libpfm 4.6 to manage performance counters:
  - Gather IPC and bandwidth utilization through the memory hierarchy



Main memory: 12 GB DDR3

Shared 12 MB LLC cache

Private 256KB L2 cache

Private 32KB L1 cache

Dual-thread cores

Fig 2. Memory hierarchy of Intel Xeon E5645

Outline

- Introduction
- Experimental platform
- **Estimating progress**
- Progress-Aware scheduling
- Experimental evaluation
- Conclusions

Estimating progress

- Accurately estimating how the processes progress is the key to devise fairness oriented schedulers
- Progress can be estimated as:

$$Progress\ (p) = \sum_{i=0}^{Q} \frac{IPC^{i}_{co-schedule}}{IPC^{i}_{alone}}$$

$IPC^{i}_{co-schedule}$: IPC of the process running in the co-schedule

- Directly calculated with performance counters measures

IPC^{i}_{alone}: IPC that the process would have achieved in isolation during the same quantum

- How it is obtained is the key

Estimating progress

- To obtain IPC^i_{alone} we propose to arrange a low-contention co-schedule:
  - Intra-core interferences are avoided, allocating the process alone on a core
  - Inter-core interferences are minimized, selecting co-runners that cause little interferences in the co-runners

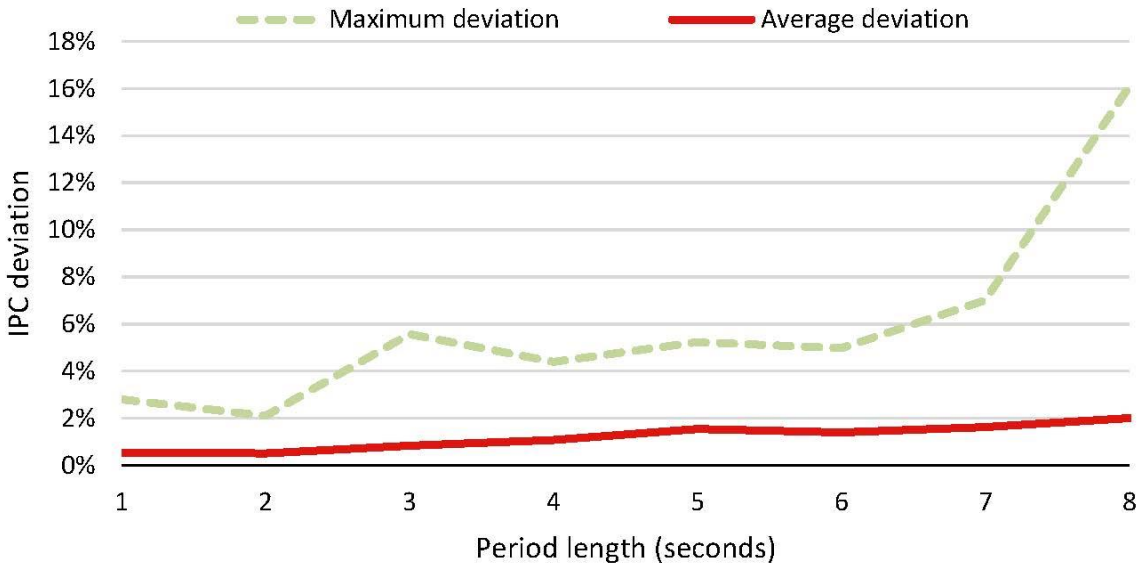
- The IPC of the target process is measured in the low-contention co-schedule
  - $IPC^i_{alone} \approx IPC^i_{co-schedule}$
  - Assumed valid for the *n* following quanta

- Two reasons can cause inaccuracy in the estimates:
  I. Standalone IPC assumed valid for a too long interval
  II. Process interferences are high in the low-contention co-schedule

Estimating progress

I. Period length between IPC estimates

- Tradeoff between accuracy and quanta used on estimates:
  - Long interval, inaccuracy could rise
  - Short interval, more quanta devoted to IPC estimates
- We compare the measured IPC of the benchmarks
  - Baseline: IPC measured each quantum
  - With periods from 1s to 8s between estimates



Maximum deviation: < 6% with periods below 6s

Average deviation < 2%

Fig 3. IPC deviation when increasing the period length between measures

Estimating progress

II. Process interferences in low-contention co-schedules

- Our goal is to classify the process two categories:
  - Heavy-sharing -> strong impact on co-runners
  - Light-sharing -> slight impact on co-runners

Co-runner

Target process	perlbench	bzip2	gcc	mcf	gobmk	hmmer	sjeng	libquantum	h264ref	omnetpp	astar	xalancbmk	bwaves	gamess	milc	zeusMP	gromacs	cactusADM	leslie3d	namd	dealII	soplex	povray	gemsFDTD	lbm
perlbench	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	1%	0%	0%	1%	1%	0%	1%	0%	0%	1%	0%	0%	1%
bzip2	0%	0%	1%	6%	0%	0%	0%	8%	0%	4%	3%	3%	8%	0%	9%	3%	0%	2%	7%	0%	1%	6%	0%	8%	9%
gcc	0%	1%	3%	8%	1%	0%	1%	10%	1%	6%	4%	5%	11%	0%	11%	5%	1%	3%	9%	0%	1%	8%	0%	10%	10%
mcf	0%	0%	3%	24%	2%	2%	1%	28%	3%	15%	13%	17%	29%	0%	29%	4%	2%	6%	24%	0%	5%	13%	0%	28%	32%
gobmk	0%	0%	0%	1%	0%	0%	0%	4%	1%	1%	0%	2%	2%	0%	4%	2%	1%	2%	3%	1%	0%	3%	0%	4%	4%
hmmer	0%	0%	0%	1%	0%	0%	0%	2%	0%	1%	0%	0%	3%	0%	2%	1%	0%	0%	2%	0%	0%	1%	0%	1%	3%
sjeng	0%	0%	0%	1%	0%	0%	3%	6%	3%	1%	4%	4%	6%	3%	6%	4%	3%	4%	6%	3%	3%	5%	3%	6%	6%
libquantum	0%	0%	0%	0%	0%	0%	0%	1%	0%	2%	0%	0%	2%	0%	1%	1%	0%	0%	0%	0%	0%	2%	0%	4%	4%
h264ref	0%	0%	0%	4%	0%	0%	0%	6%	0%	2%	1%	3%	6%	0%	11%	2%	0%	1%	8%	0%	1%	6%	0%	10%	6%
omnetpp	1%	6%	7%	15%	3%	3%	3%	17%	5%	14%	10%	13%	17%	2%	18%	11%	4%	10%	16%	1%	4%	15%	0%	19%	19%
astar	1%	4%	5%	12%	2%	2%	3%	14%	3%	10%	17%	17%	14%	5%	22%	15%	7%	5%	21%	6%	3%	20%	1%	22%	23%
xalancbmk	0%	4%	7%	21%	2%	2%	2%	25%	3%	15%	14%	19%	28%	1%	28%	10%	2%	6%	23%	1%	4%	24%	0%	27%	30%
bwaves	0%	0%	0%	1%	0%	0%	0%	1%	0%	1%	1%	0%	9%	8%	9%	8%	8%	8%	9%	8%	8%	1%	8%	9%	9%
gamess	0%	0%	0%	1%	0%	1%	0%	1%	0%	0%	0%	1%	1%	0%	1%	1%	0%	0%	0%	0%	0%	1%	0%	1%	1%
milc	0%	0%	0%	1%	0%	0%	0%	2%	0%	1%	1%	1%	2%	0%	25%	24%	24%	24%	3%	24%	24%	24%	24%	25%	25%
zeusMP	1%	1%	1%	2%	0%	0%	1%	2%	0%	1%	1%	1%	2%	0%	2%	10%	8%	8%	9%	8%	8%	9%	0%	9%	9%
gromacs	0%	0%	2%	2%	0%	0%	1%	2%	0%	1%	2%	1%	2%	0%	2%	1%	1%	1%	3%	0%	1%	1%	0%	3%	3%
cactusADM	0%	1%	3%	8%	1%	0%	0%	9%	0%	6%	4%	5%	9%	0%	9%	5%	0%	4%	9%	0%	1%	8%	0%	10%	10%
leslie3d	0%	0%	0%	1%	0%	0%	0%	1%	0%	0%	0%	0%	1%	0%	1%	0%	0%	0%	20%	18%	18%	19%	18%	20%	20%
namd	0%	0%	0%	1%	0%	0%	0%	1%	0%	0%	0%	0%	1%	0%	1%	0%	0%	0%	1%	0%	0%	1%	0%	2%	1%
dealII	0%	0%	0%	1%	0%	0%	0%	2%	0%	1%	1%	0%	2%	0%	1%	0%	0%	0%	1%	0%	1%	1%	0%	2%	2%
soplex	2%	4%	6%	19%	3%	3%	3%	20%	5%	13%	11%	15%	21%	1%	19%	11%	2%	7%	17%	1%	4%	20%	0%	21%	24%
povray	0%	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	1%	0%	1%	0%	0%	0%	0%	0%	0%	1%	1%	0%	1%
gemsFDTD	0%	0%	0%	1%	0%	0%	0%	1%	0%	0%	0%	0%	1%	0%	1%	0%	0%	0%	0%	0%	0%	1%	0%	2%	2%
lbm	0%	4%	23%	6%	0%	0%	0%	8%	0%	3%	2%	5%	11%	0%	9%	2%	0%	1%	7%	0%	1%	8%	0%	11%	36%

Fig 5. Performance degradation due to inter-core interferences running pairs of benchmarks on different cores

Estimating progress

II. Process interferences in low-contention co-schedules

- Our goal is to classify the process two categories:
  - Heavy-sharing -> strong impact on co-runners
  - Light-sharing -> slight impact on co-runners
- Inter-core interferences caused by LLC and MM contention
  - Thresholds -> MM: 3.5t/µs LLC: 19t/µs
  - If a process reaches any of the two thresholds -> heavy-sharing

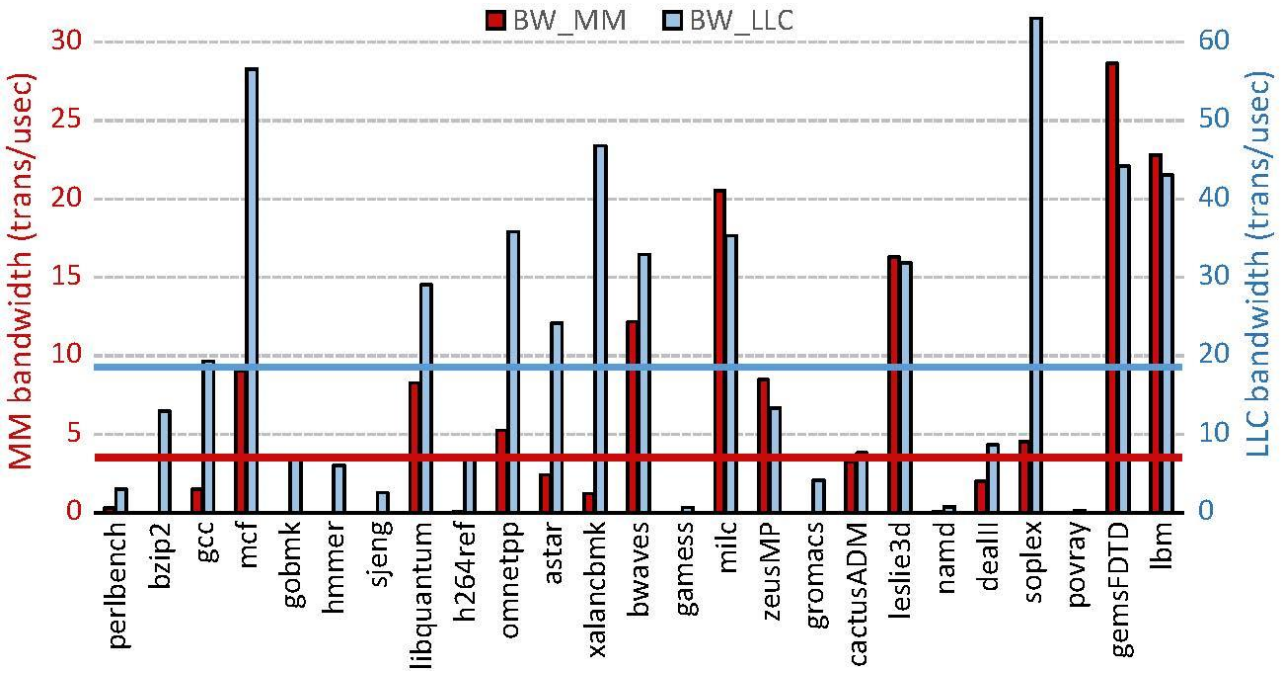


Fig 6. Average main memory and LLC bandwidth

Estimating progress

II. Process interferences in low-contention co-schedules

- Evaluate the performance degradation on all possible co-schedules from three to six light-sharing processes
  - The figure presents the frequency on which the performance degradation in the co-schedules falls in the intervals
  - Average and maximum IPC degradation among the processes of each co-schedule

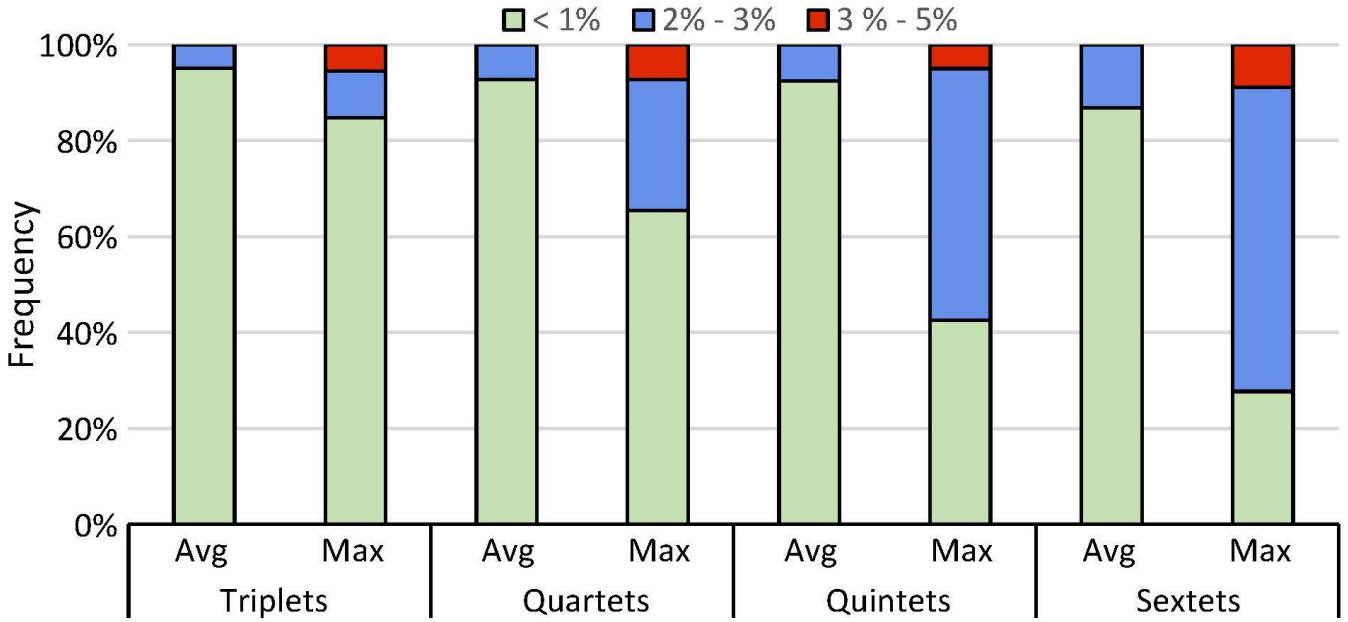
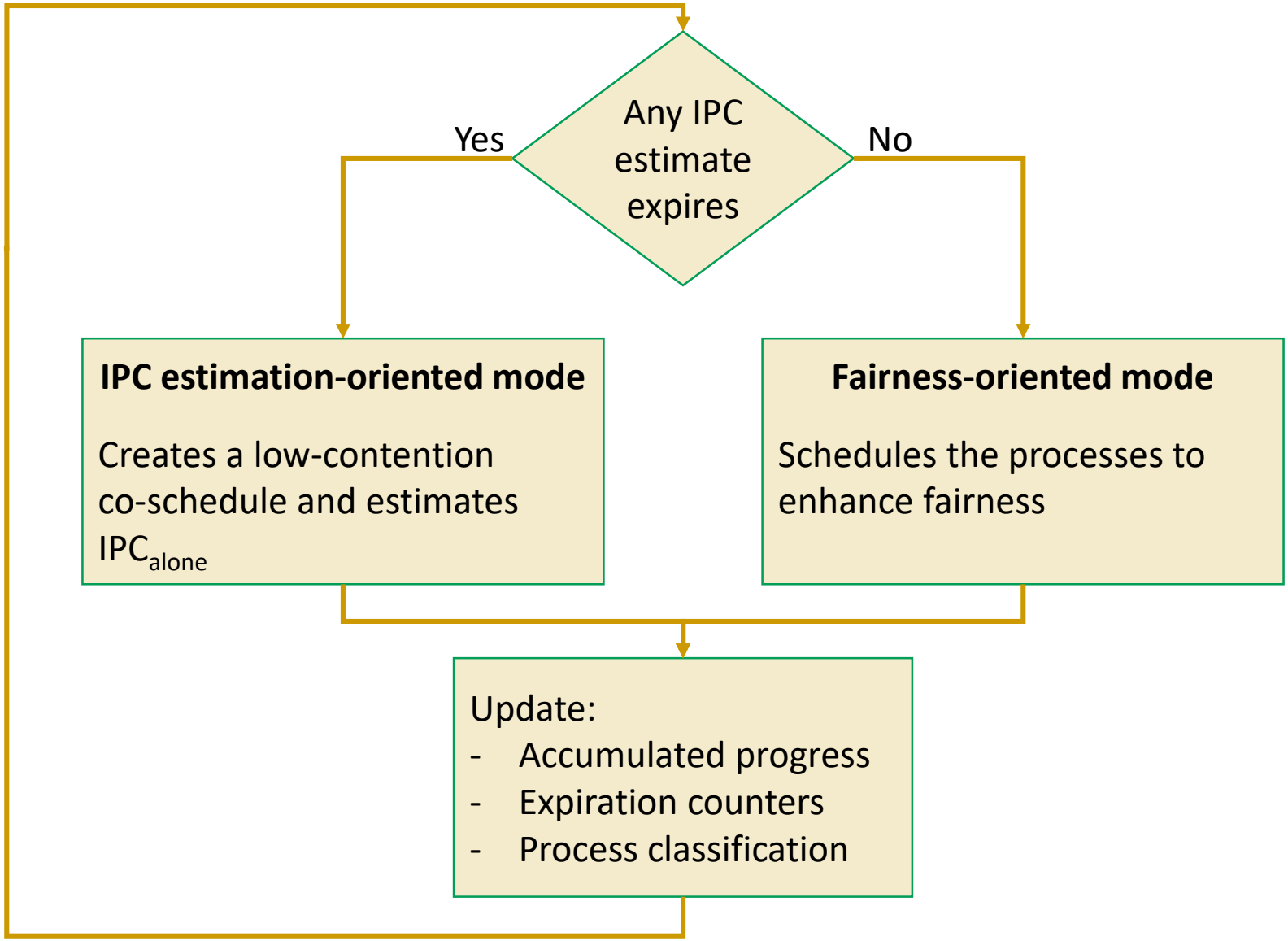


Fig 7. Performance degradation on light-sharing co-schedules

Outline

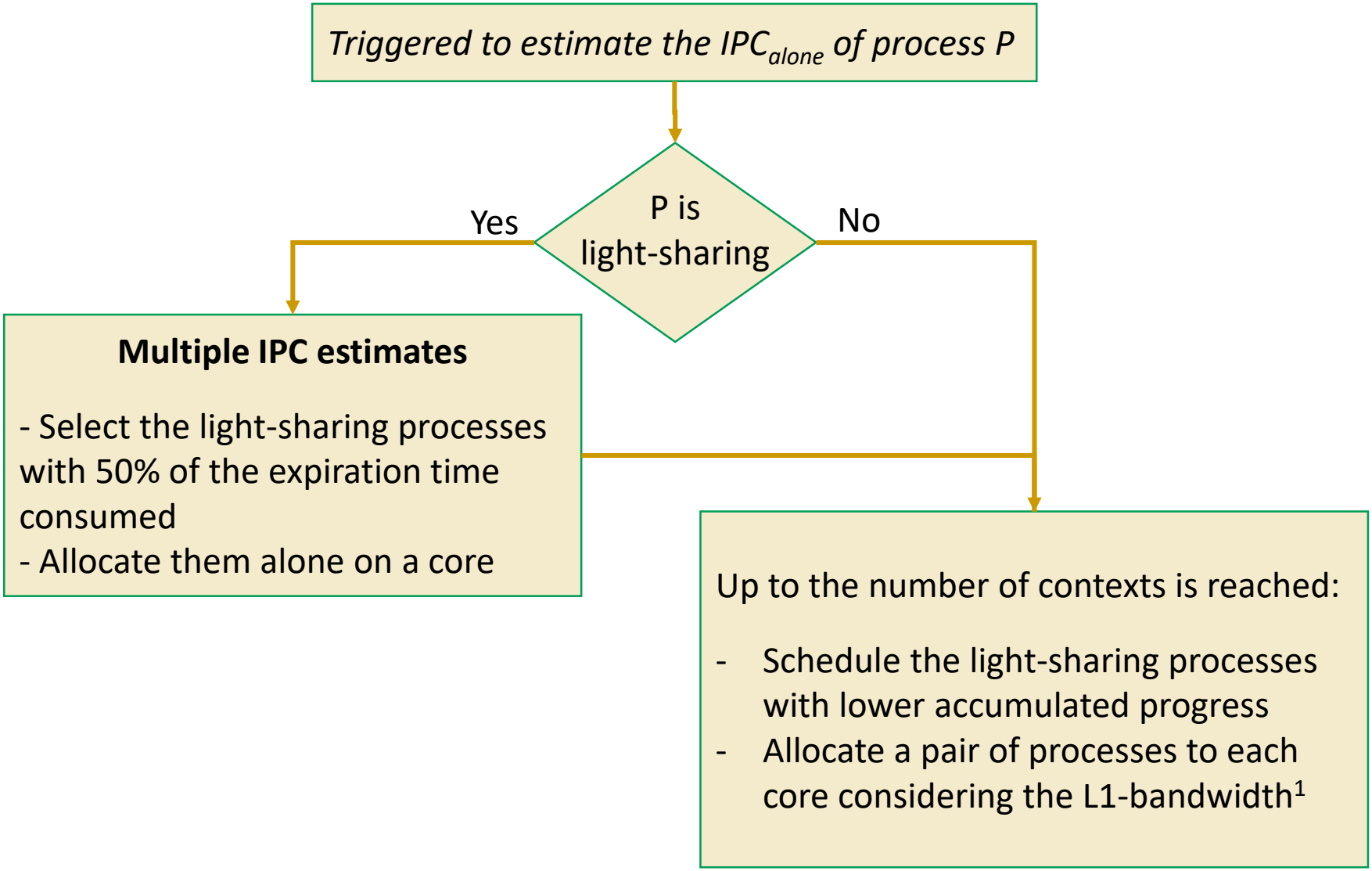
- Introduction
- Experimental platform
- Estimating progress
- **Progress-Aware scheduling**
- Experimental evaluation
- Conclusions

Progress-Aware Scheduling



Progress-Aware Scheduling

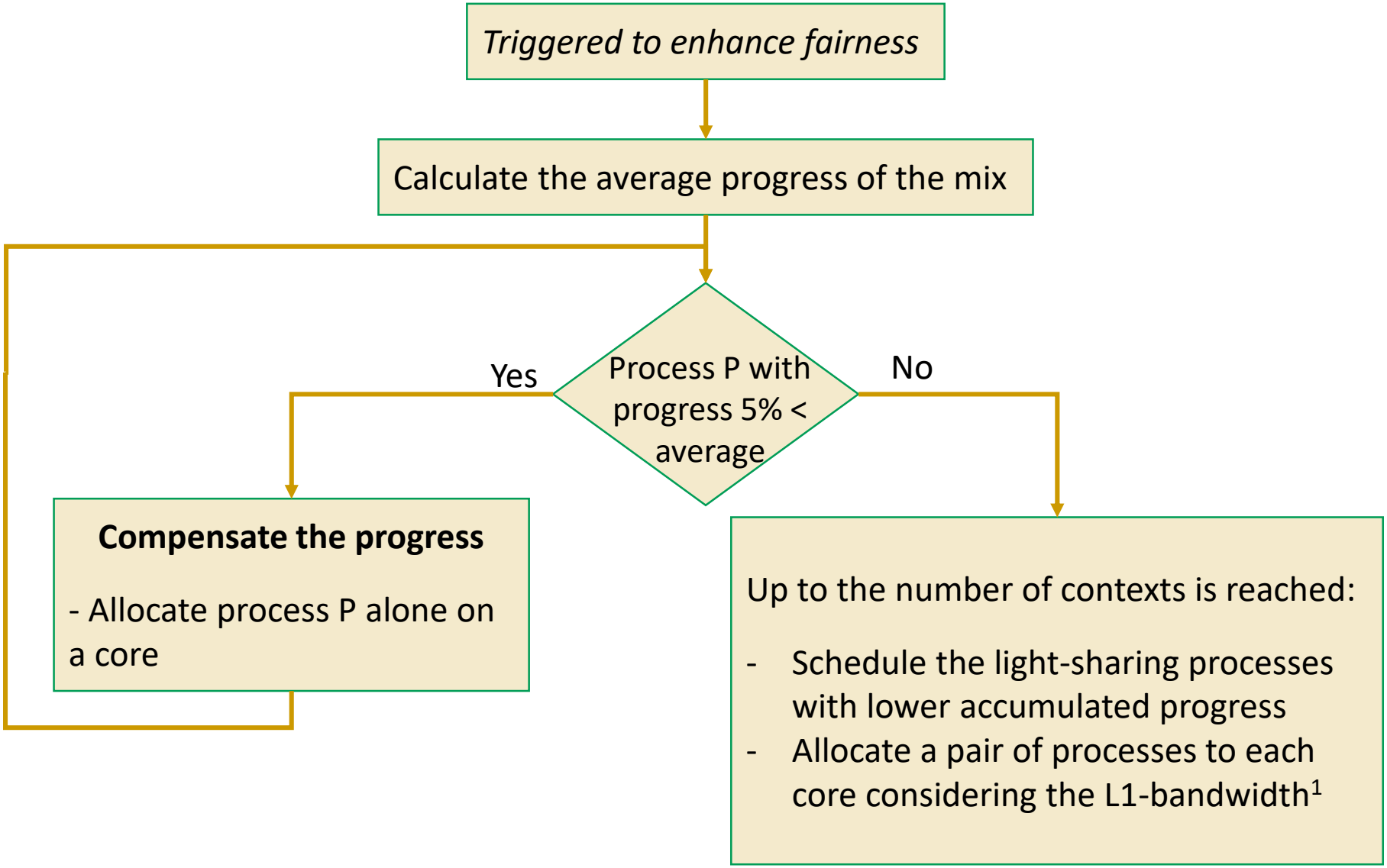
IPC estimation-oriented



[1] J. Feliu, J. Sahuquillo, S. Petit, and J. Duato, "L1-Bandwidth Aware Thread Allocation in Multicore SMT Processors", PACT'13

Progress-Aware Scheduling

Fairness-oriented mode



Outline

- Introduction
- Experimental platform
- Estimating progress
- Progress-Aware scheduling
- **Experimental evaluation**
- Conclusions

Experimental evaluation

Evaluation methodology

- The algorithm is implemented in a user-level scheduler, using:
  - System calls: determine which processes run each quantum
  - Thread-to-core affinity attribute: determine on which core each process runs
  - Performance counters: update IPC and bandwidth

  ➢ Negligible overhead of scheduling, below 0.1% of the quantum length (200ms)

- Fourteen mixes of 24 benchmarks
- Fairness metric:

$$Unfairness = \frac{Max\ Slowdown}{Min\ Slowdown}\ \forall\ P\ \varepsilon\ \{1, N\}$$

$$Slowdown = \frac{T_{co-schedule}^{running} + T_{co-schedule}^{waiting}}{T_{alone}}$$

- Performance metrics
  - Turnaround time of the mix
  - System throughput = $\sum_{i=0}^{N} \frac{IPC_{co-schedule}^{i}}{IPC_{alone}^{i}}$

Experimental evaluation

Fairness

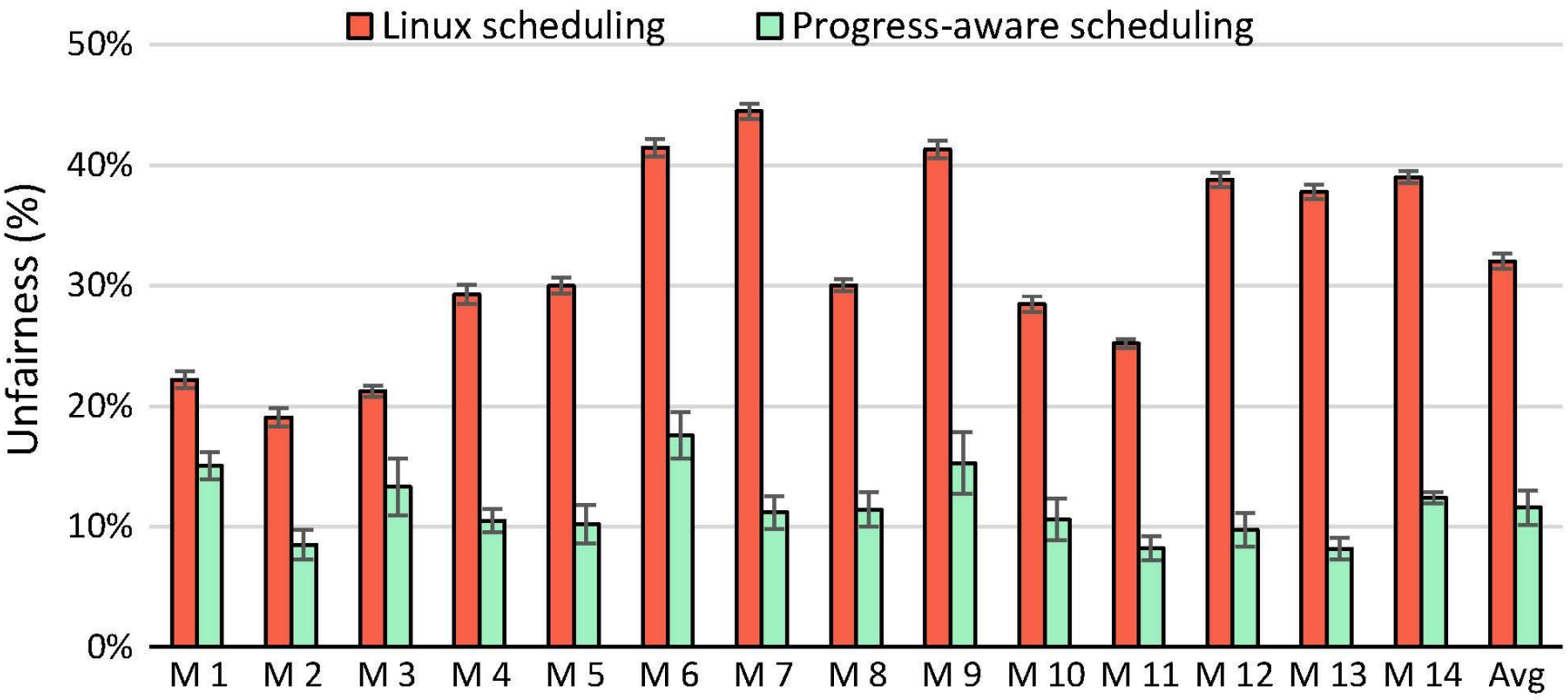


Fig 8 Unfairness (lower is better)

- The Progress-Aware scheduler performs fairer than Linux:
- Unfairness is reduced to a third on some mixes
- More steady results with the Progress-Aware scheduler

Experimental evaluation

Fairness

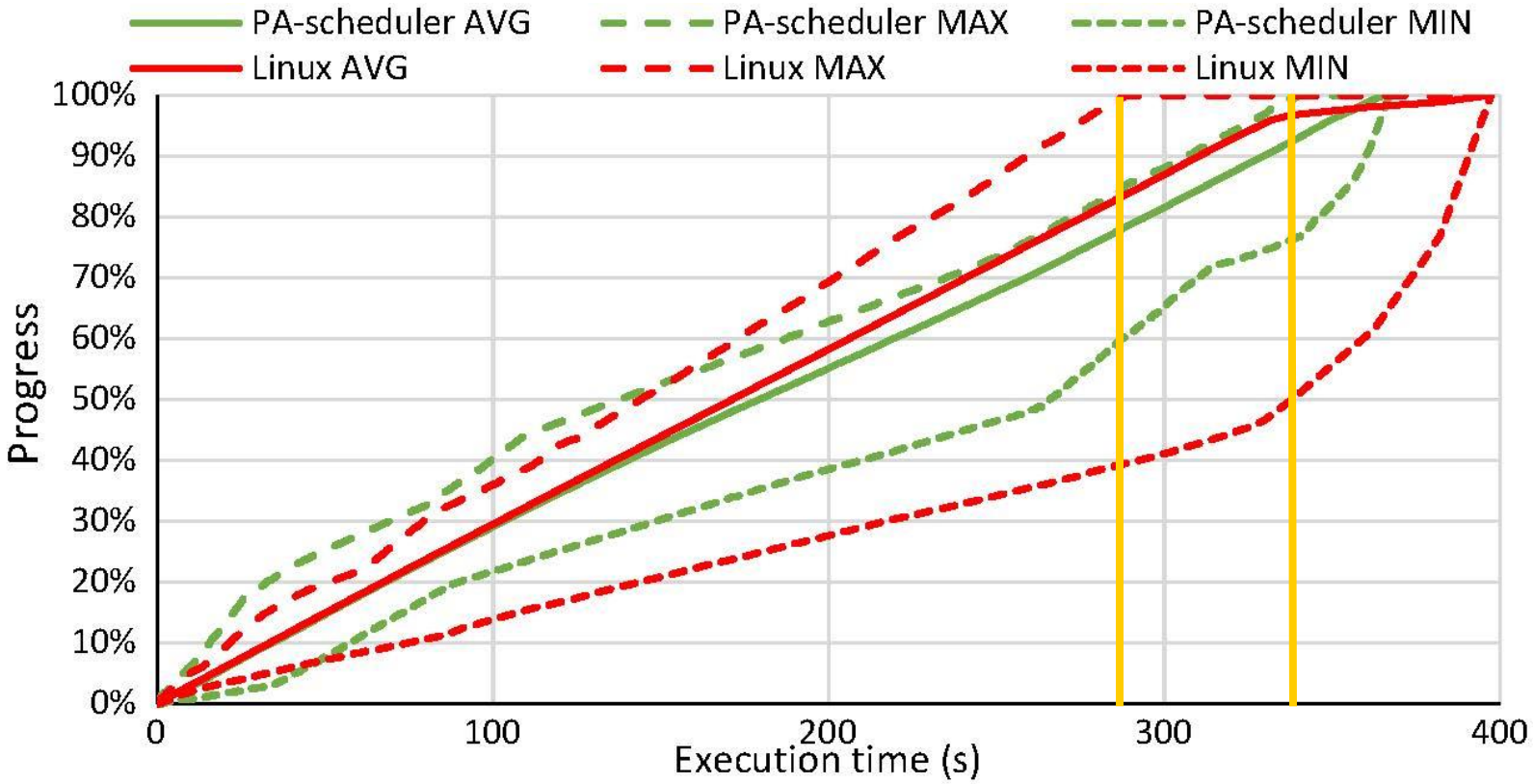


Fig 9. Dynamic progress of processes of mix 7

- The plot shows how unfairness evolves over the mix execution
- When the first processes finishes, the process with minimum progress:
  - With Linux: has completed 40% of its execution
  - With the PA scheduler: has completed 80% of its execution

Experimental evaluation

Performance

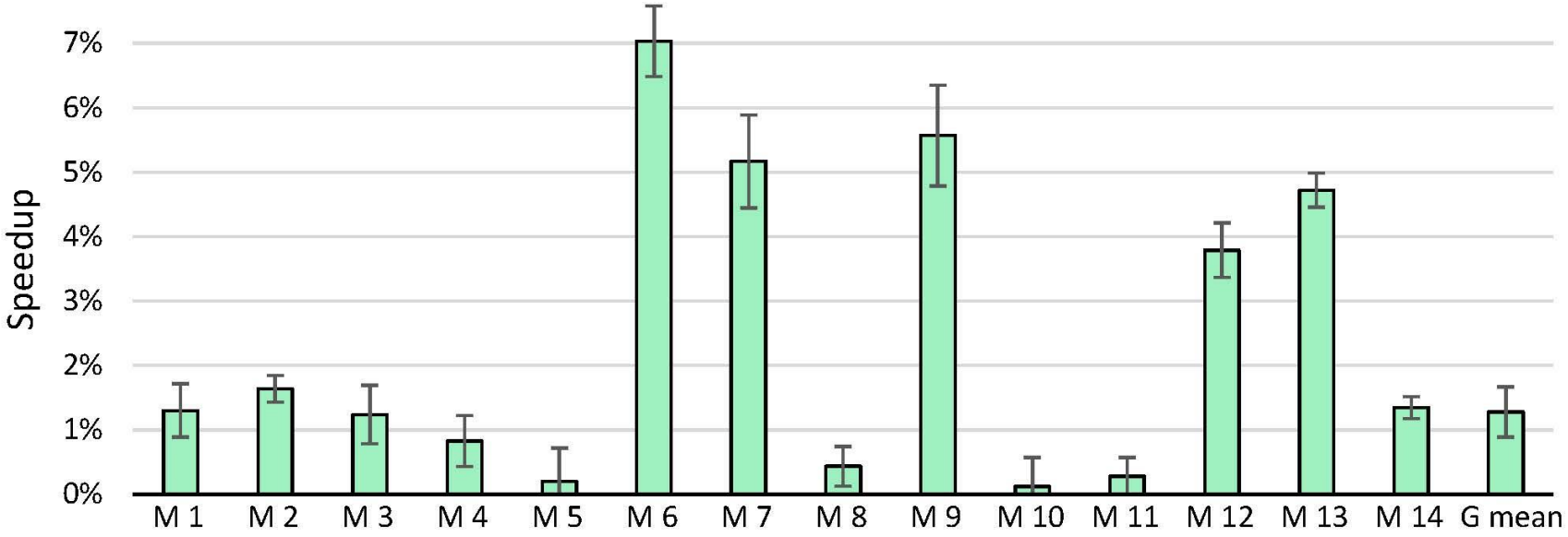


Fig 10. Speedup of the turnaround time over the Linux scheduler

- Turnaround time is not negatively affected, despite fairness-oriented scheduling
- In fact, the Progress-Aware scheduler improves Linux turnaround time in all the mixes.

Experimental evaluation

Performance

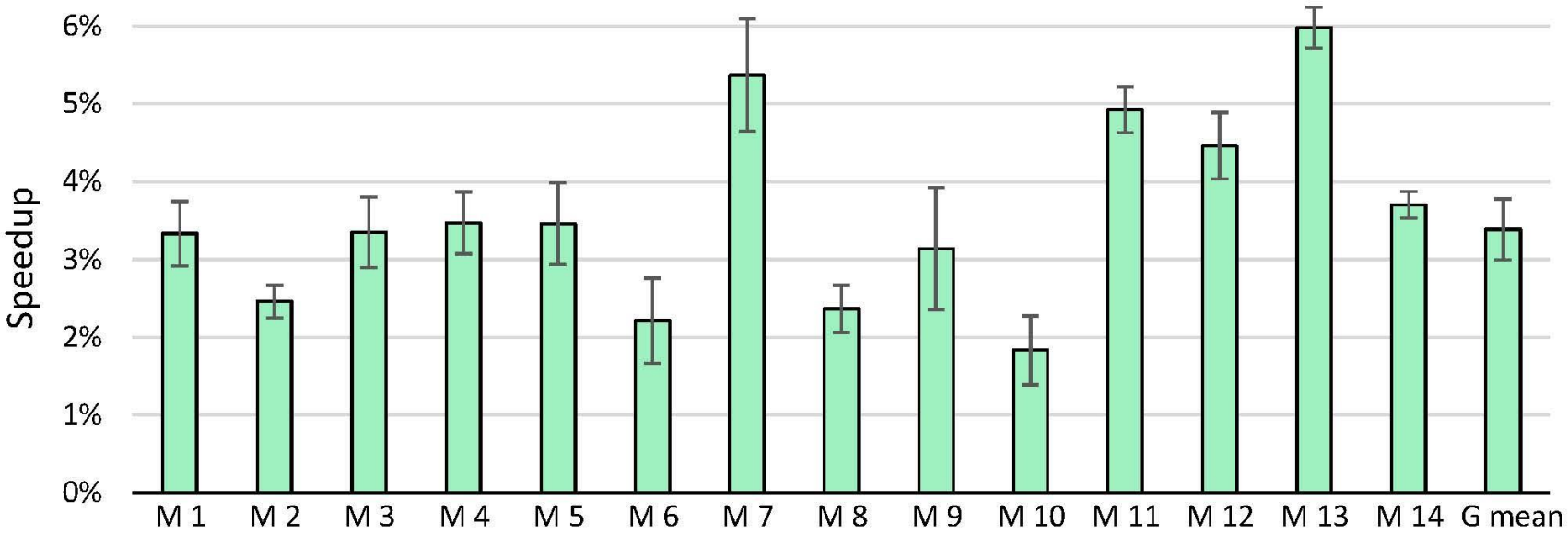


Fig 11. Speedup of the system throughput (STP) over the Linux scheduler

- The progress-aware scheduler improves Linux scheduler throughput
- Key reason to improve throughput:
  - L1-bandwidth aware process allocation policy

Conclusions

- Scheduling considering fairness is gaining importance to:
  - Keep process priorities, QoS, guarantee WCET, fair billing in cloud computing, etc.
  - Unfairness problems rise in heterogeneous systems
- Progress-Aware scheduler for SMT multicores
  - Balances the slowdowns suffered by the processes of a workload
  - Calculates the progress of the processes using estimates of their standalone performance
  - Prioritizes the processes with lower accumulated progress to maximize fairness
- Reduces Linux unfairness to a third, while slightly improving performance

Thank you for your attention. Any question???

Addressing Fairness in SMT Multicores with a Progress-Aware Scheduler

J. Feliu, J. Sahuquillo, S. Petit, and J. Duato

Universitat Politècnica de València

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

May 26th, 2015

Hyderabad, India

Progress-Aware Scheduling

- Scheduling goal: maximize fairness
  - All the processes achieve the same progress along the mix execution
- Interferences impact differently on the individual performance of the processes.
  - Equal execution time ≠ equal progress

IF the IPC estimate of any process P expires

IPC estimation-oriented mode

Creates a low-contention co-schedule for the process P

ELSE

Fairness-oriented mode

Schedules the processes to reduce unfairness

FI

Update the progress, expiration counter and classification for the executed processes

Progress-Aware Scheduling

IPC estimation-oriented

- Creates a low-contention co-schedule:
  - Remove intra-core interferences: P allocated alone on a core
  - Minimize inter-core interferences: selecting light-sharing co-runners
- The quanta used to estimate IPC should be low:
  - Not directly devoted to improve fairness
  - If possible, multiple estimates performed in a single quantum

IF P is a light-sharing processes

- Schedule light-sharing processes that have consumed half their period length between estimates
- Allocate them on a entire core to estimate their IPC.

FI

- The remaining processes are selected prioritizing the light-sharing processes with lower accumulated progress
- Processes with no allocation restrictions are allocated on cores balancing the L1-bandwidth among the cores[1]

[1] J. Feliu, J. Sahuquillo, S. Petit, and J. Duato, "L1-Bandwidth Aware Thread Allocation in Multicore SMT Processors", PACT'13

Progress-Aware Scheduling

Fairness-oriented mode

- *Rule of thumb* to improve the fairness: schedule the processes with lower accumulated progress
- Processes with high performance degradation may have difficulties to keep the progress pace of other processes
  - Their execution is favored allocating them on a entire core

- Compute the avg progress of the processes of the mix

FOR all the processes whose accumulated progress is 5% below the average progress

- Schedule and allocate them on an entire core to boost their progress

DONE

- The remaining processes are selected prioritizing the processes with lower accumulated progress
- Processes with no allocation restrictions are allocated on cores balancing the L1-bandwidth among the cores

Experimental evaluation

Deviation of IPC estimates

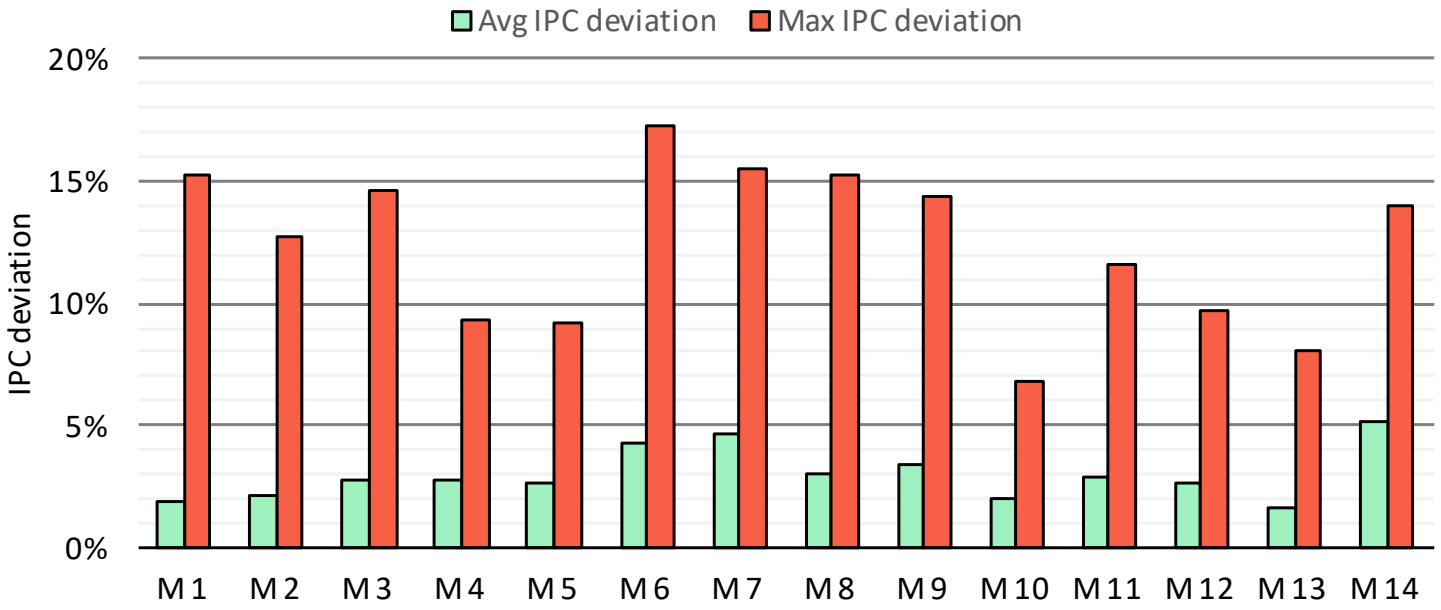


Fig 9. Average and maximum IPC deviation of the processes of the mixes.

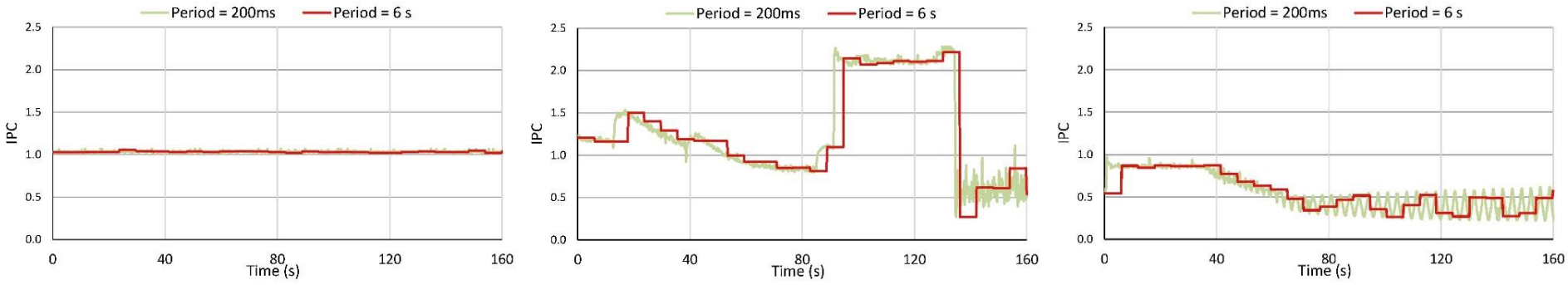
- The figure plots average and maximum IPC deviation over real IPC among the processes of each mix.
  - Related with progress estimations, and thus fairness.

- The average IPC deviation falls below 5% for all the mixes
- Maximum IPC deviation ranges between 7% to 18%.

Estimating progress

Period length between IPC estimates

- Tradeoff between accuracy and overhead
  - Long interval, estimations assumed valid more quanta
  - Short interval, more quanta devoted to IPC estimates
- Average deviation < 2%, maximum deviation < 6%, with period length of 6 seconds.
- The plots illustrate the small deviation, comparing IPC updated at 200ms and 6 seconds periods.



(a) hmmer (b) xalancbmk (c) cactusADM

Fig 4. Comparison between IPC measured each 200ms and each 6s.