Understanding Cache Hierarchy Contention in CMPs to Improve Job Scheduling

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

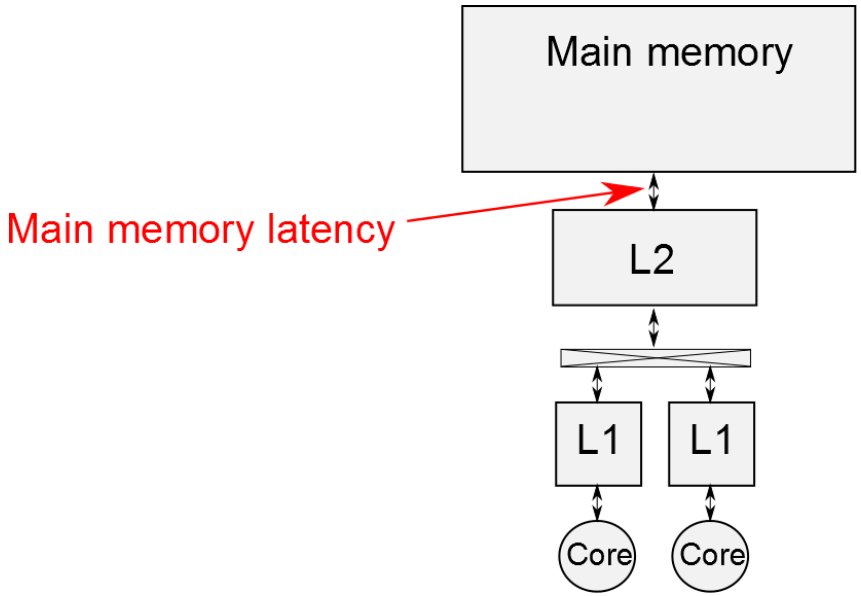
Universitat Politècnica de València
Spain

Outline

- Introduction
- Experimental platform
- Benchmark characterization and performance degradation analysis
- Cache-hierarchy bandwidth aware scheduler
- Methodology and evaluation
- Conclusions

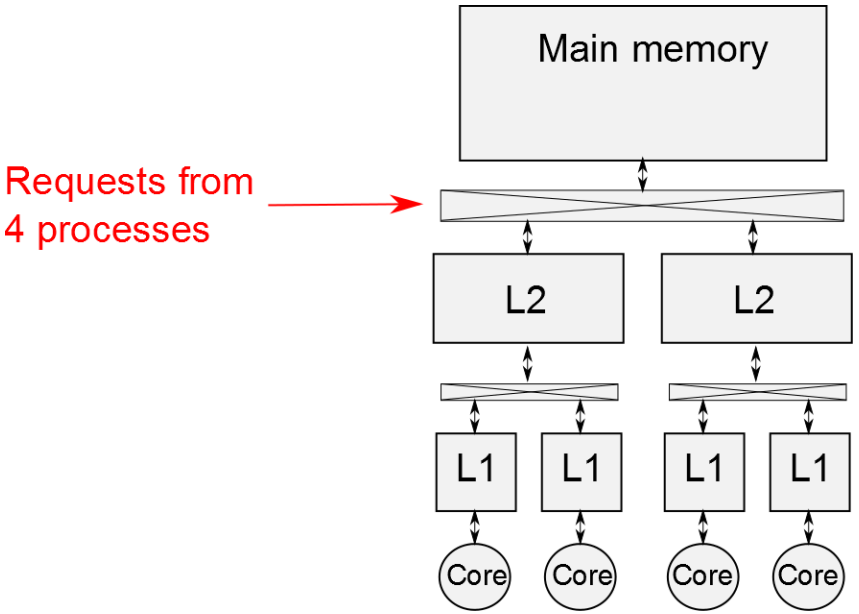
Introduction

- Multi-core processors have become the common implementation for high-performance microprocessors.
- CMPs main performance bottleneck lies in the main memory latency.



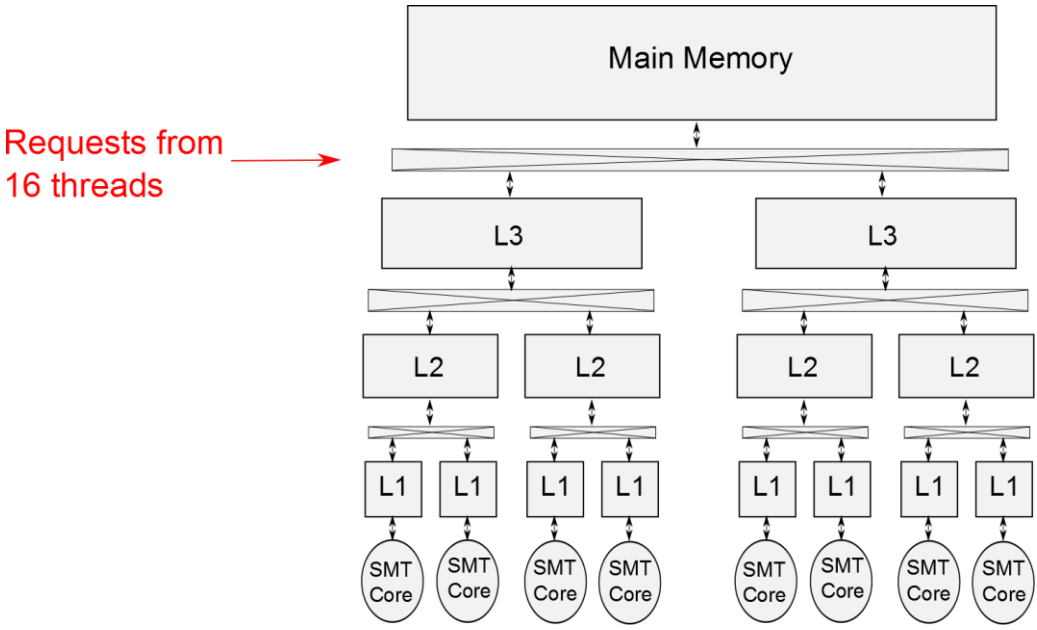
Introduction

- As the number of cores and multithreading capabilities increase, the available memory bandwidth is becoming a major concern.



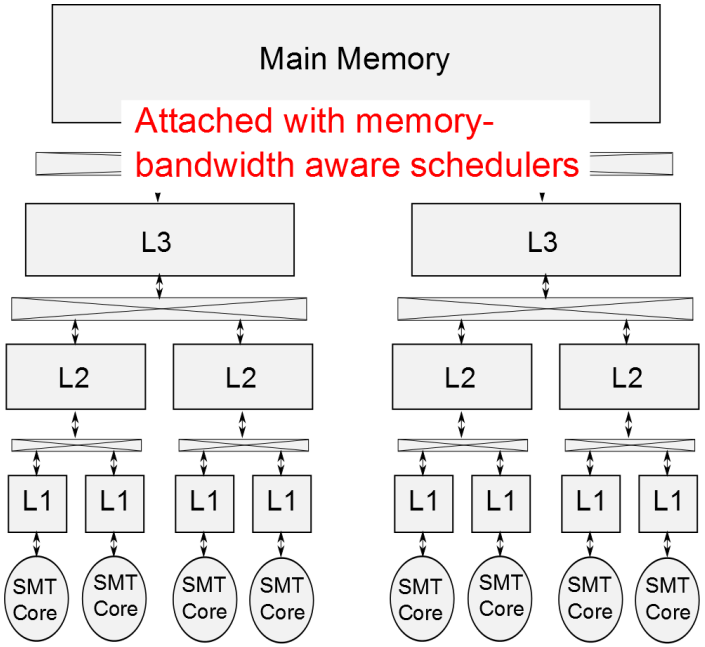
Introduction

- As the number of cores and multithreading capabilities increase, the available memory bandwidth is becoming a major concern.



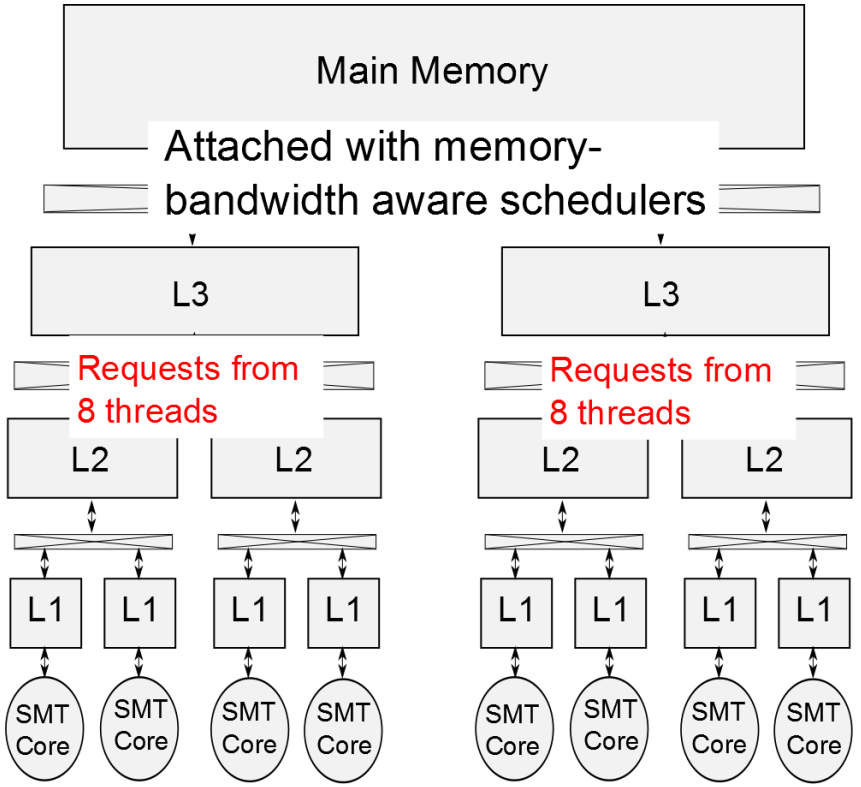
Introduction

- Memory bandwidth aware schedulers can help to reduce memory contention by avoiding the concurrent execution of memory-hungry applications.



Introduction

What about other contention points?



Introduction

Main contributions

- Two main contributions:
  - Characterize the sensitiveness of the SPEC CPU2006 benchmarks to each contention point in the memory hierarchy of a quad-core Intel Xeon which claims the necessity of the proposal.
  - Propose a scheduling approach for multi-core processors with shared caches to improve the performance.

Experimental platform

Specifications

Hardware specifications

CPU	Intel Xeon X3320
Frequency	2.5 GHz
Number of cores	4
Multithreading	No
L1 cache	Code L1: 4 x 32 KB Data L1: 4 x 32 KB
L2 cache	2 x 3 MB
Main memory	4 GB DDR2

Software specifications

Operating system	Fedora Core 10 Linux
Kernel	2.6.29 with perfmon patch
Software	pfmon, libpfm
Benchmarks	Spec CPU2006 with train input

Experimental platform

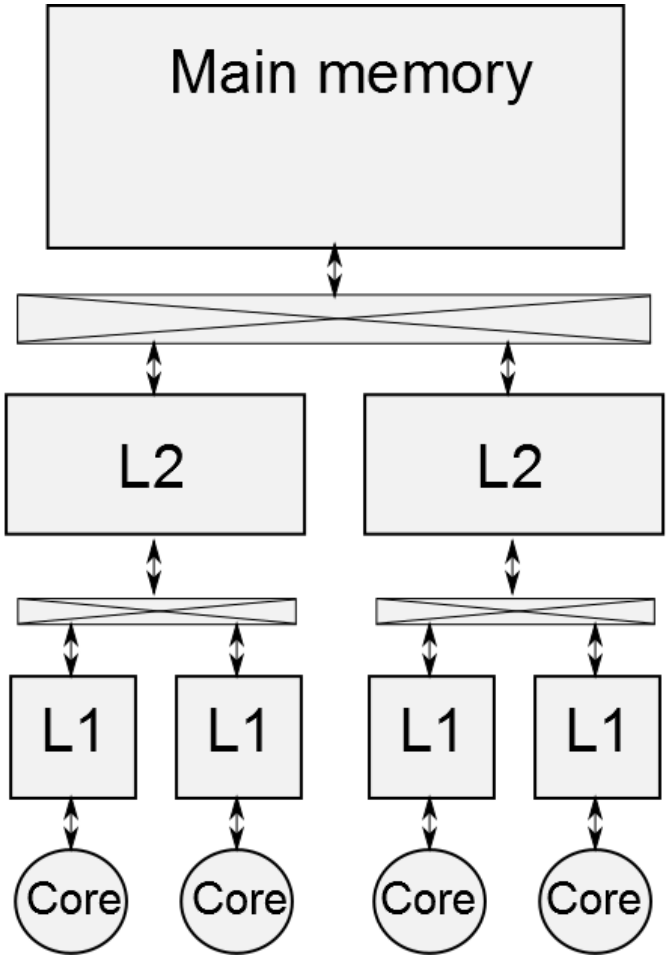
Performance counters

- A set of special-purpose registers built into modern processors.
- Store the counts of hardware-related activities within computer systems.
- Keep track of the events in a per process basis.

Monitored event	Information
UNHALTED_CORE_CYCLES	Cycles
INSTRUCTIONS_RETIRED	Instructions
L2_RQSTS:MESI	L1 misses
LAST_LEVEL_CACHE_MISSES	L2 misses

Experimental platform

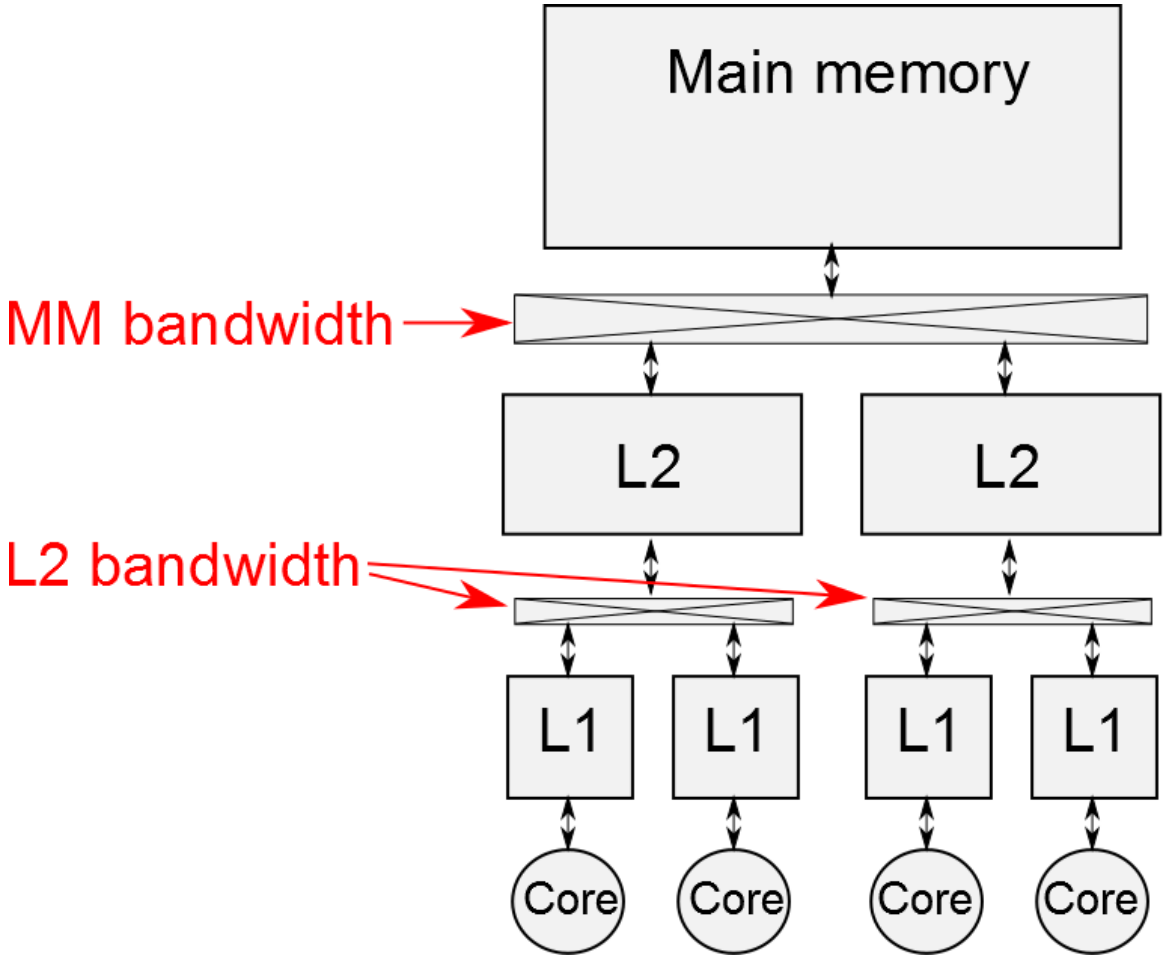
Intel Xeon X3320



Xeon X3320 memory hierarchy

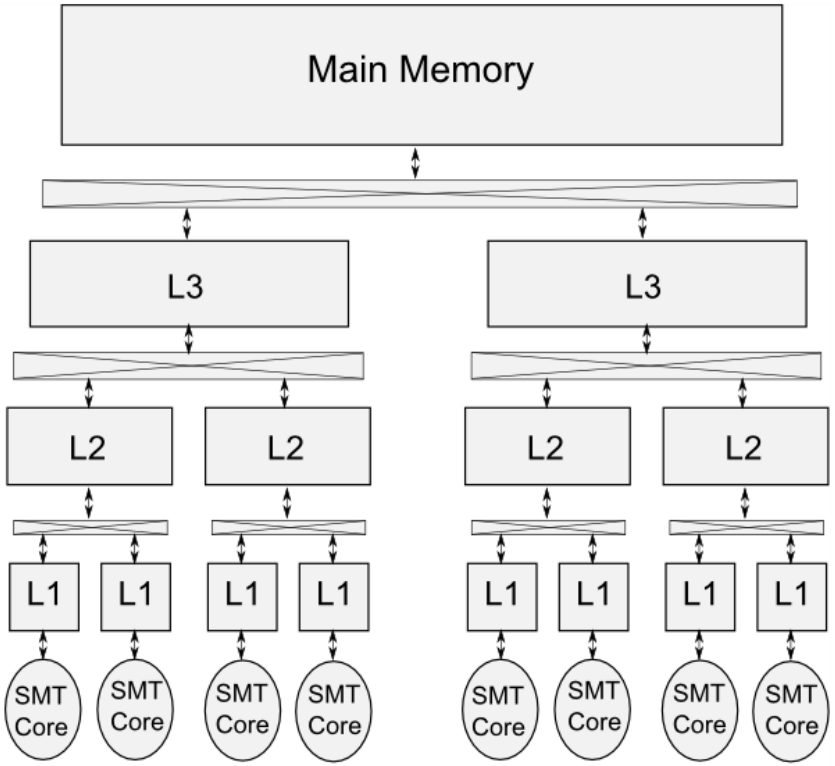
Experimental platform

Intel Xeon X3320



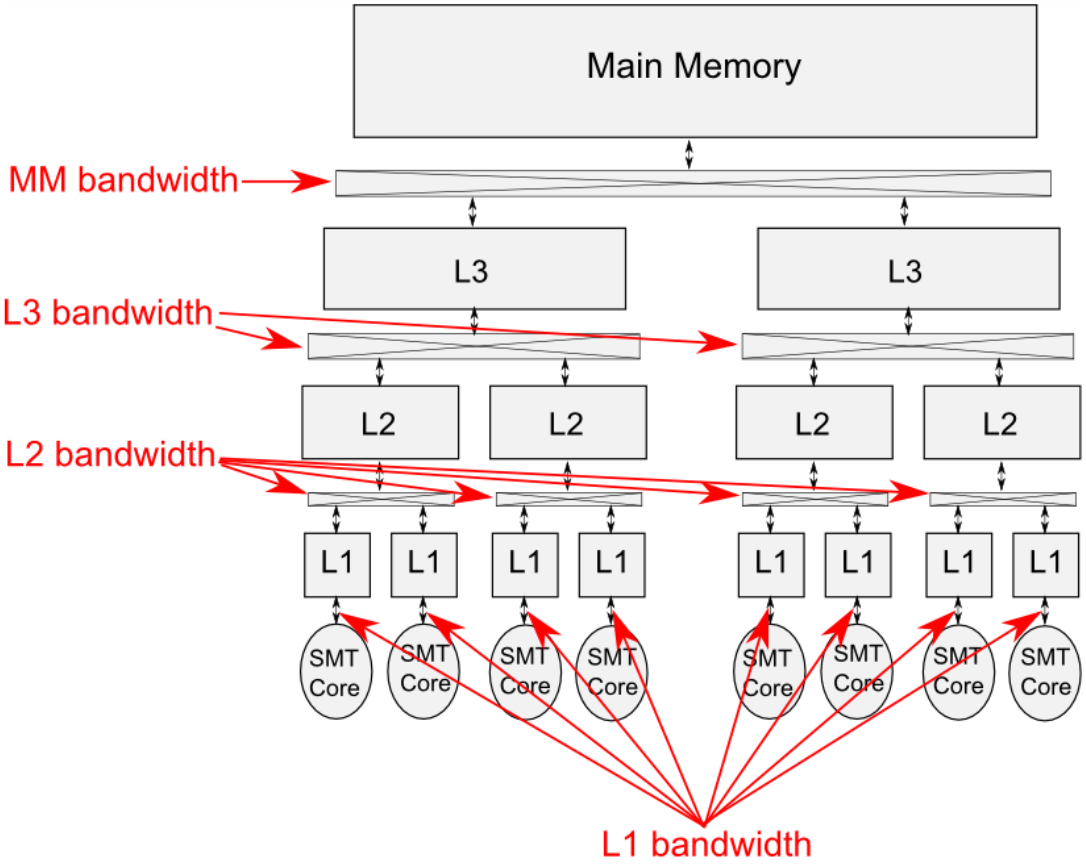
Contention points related to the memory subsystem in the Xeon X3320

Cache hierarchy in the IBM Power 5



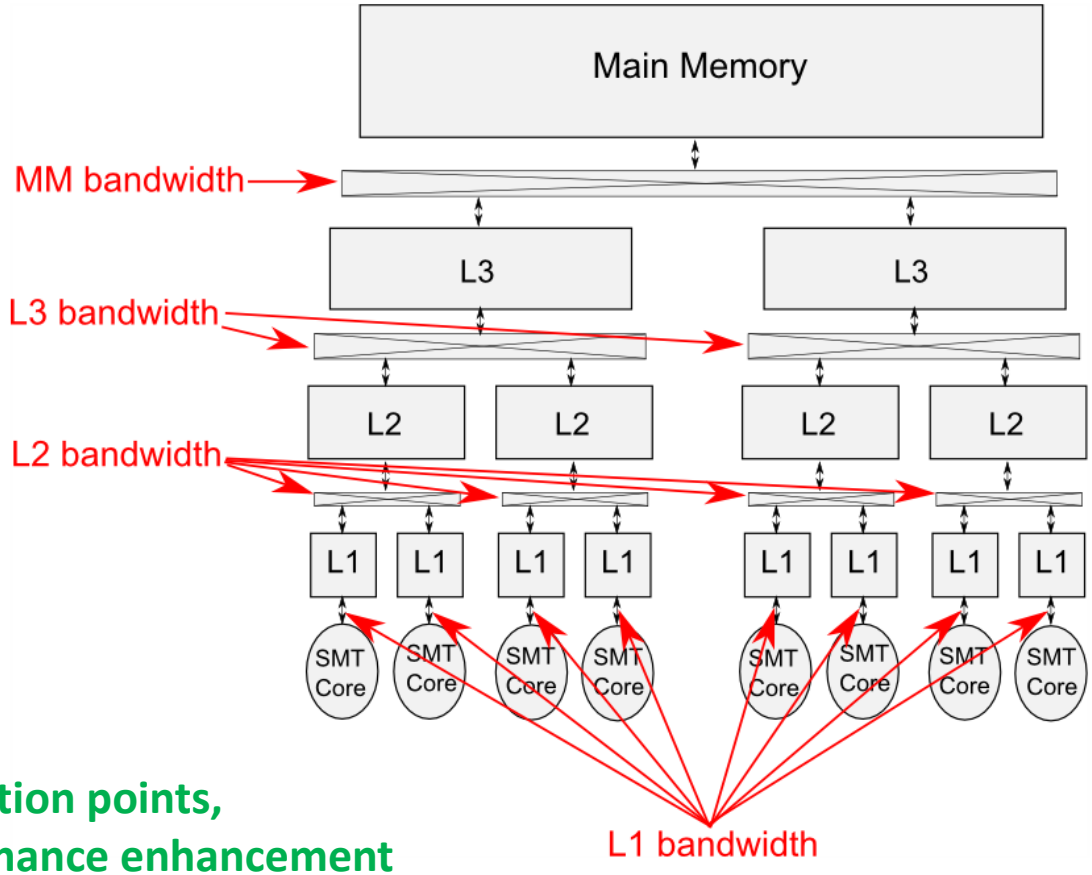
Power 5 memory hierarchy

Cache hierarchy in the IBM Power 5



Contention points related to the memory subsystem in the IBM Power 5

Cache hierarchy in the IBM Power 5



The more contention points, the more performance enhancement is expected.

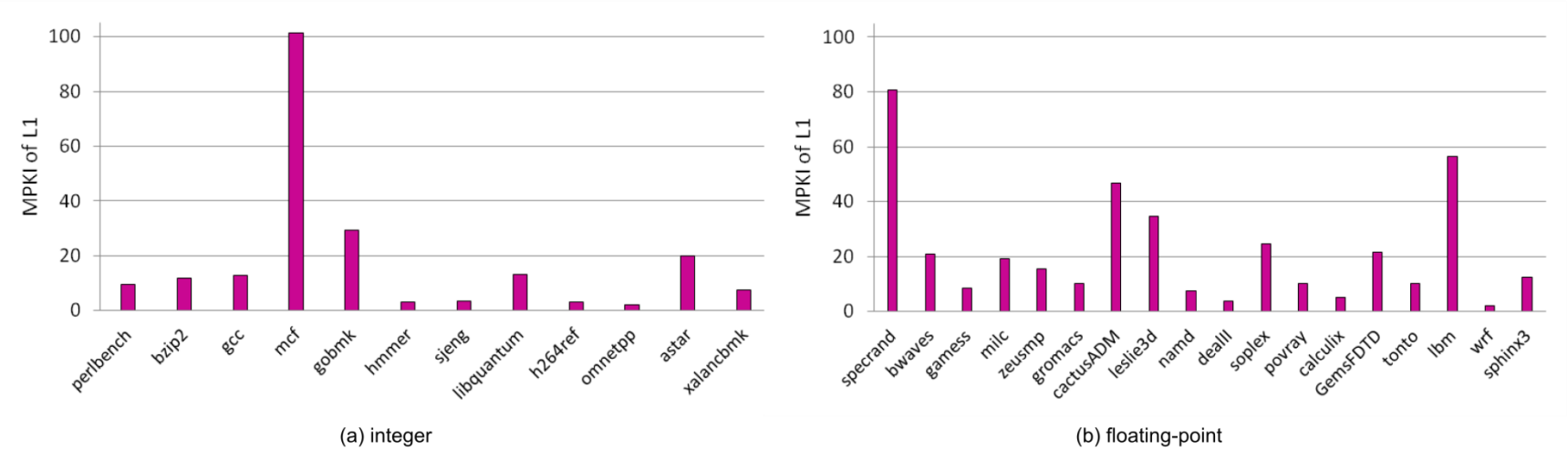
Contention points related to the memory subsystem in the IBM Power 5

Benchmark characterization and performance degradation analysis

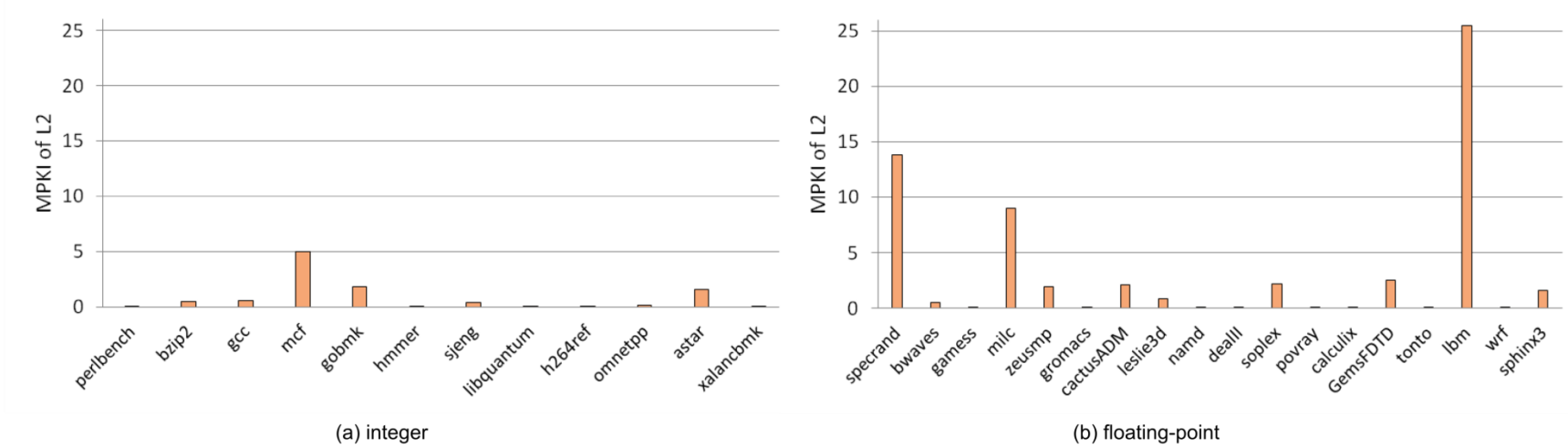
- Benchmark characterization
  - Classify the benchmarks as memory-bounded or L2-bounded.
  - Build “interesting” mixes.
- Estimation of the performance degradation due to main memory and L2 contention
  - Degradation over 60% due to main memory and around 13% due to L2 contention.
  - Motivate the work.

Benchmark characterization

L1 MPKI & L2 MPKI



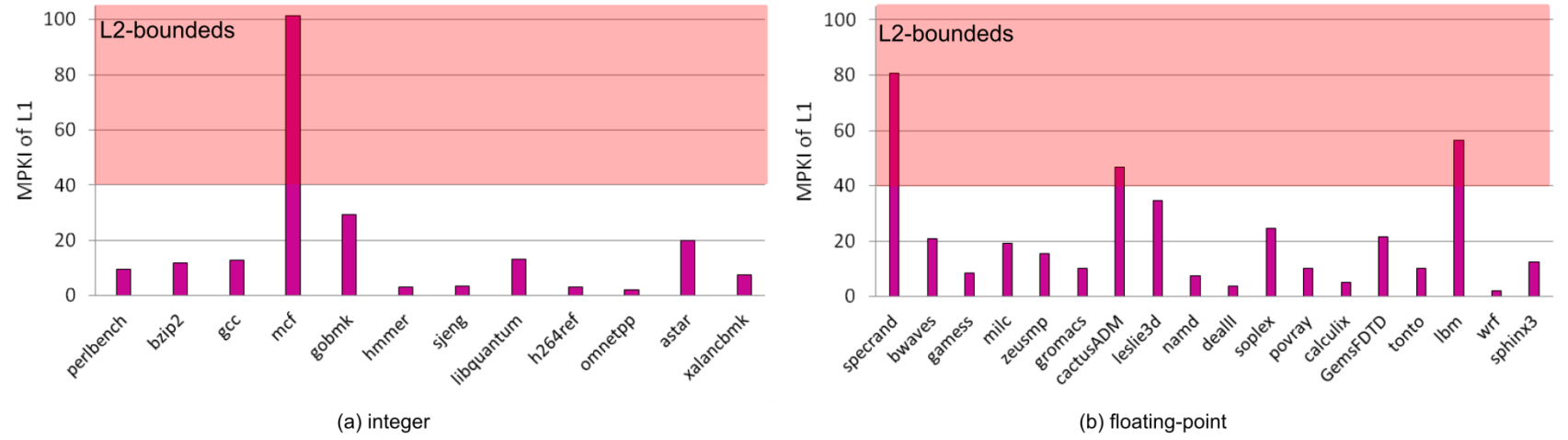
L1 MPKI for each SPEC CPU2006 benchmark



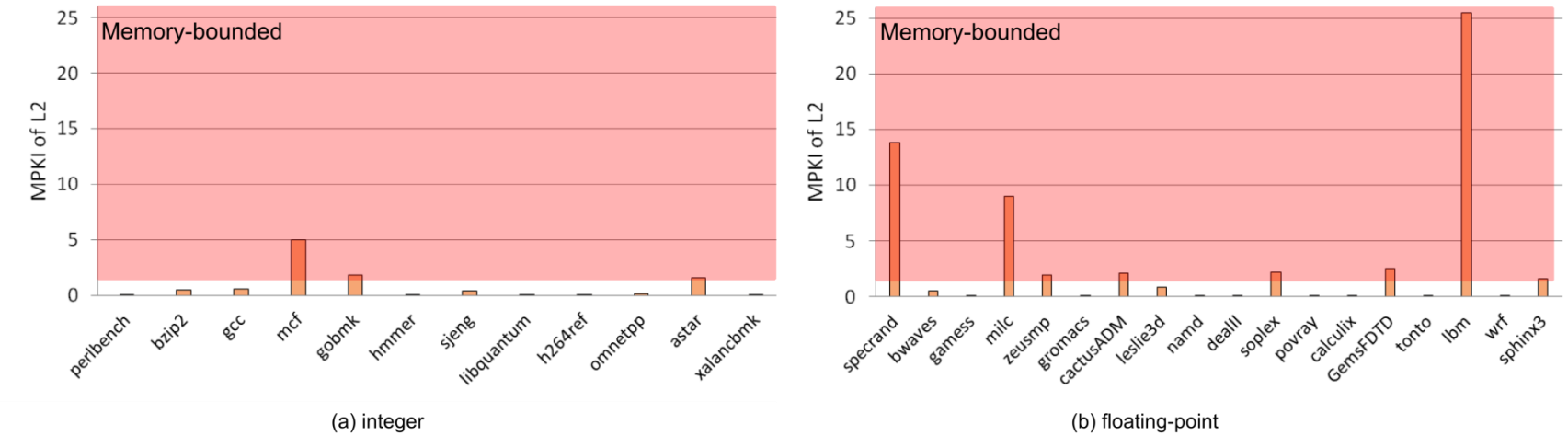
L2 MPKI for each SPEC CPU2006 benchmark

Benchmark characterization

L1 MPKI & L2 MPKI



L1 MPKI for each SPEC CPU2006 benchmark



L2 MPKI for each SPEC CPU2006 benchmark

Performance degradation analysis

Microbenchmark

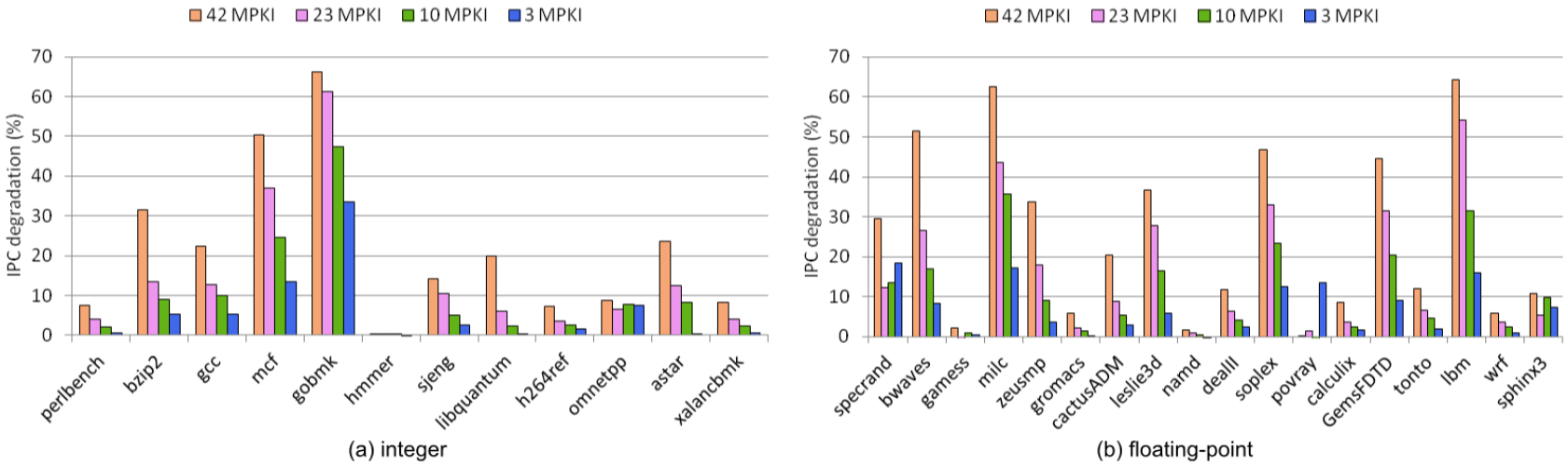
Listing 1. Microbenchmark code

```
int A[N][CACHE_LINE_SIZE];
int B[N][CACHE_LINE_SIZE];
while (1) {
    for (i=0; i< (#misses/2); i++)
    {
        A[i][0] = B[i][0];
    }
    for (i=0; i< (#nops; i++) {
        asm("nop");
    }
}
```

- Mimic the behavior of both memory-bounded and L2-bounded.
- Setting the CACHE_LINE_SIZE and N parameters according to the target cache

Performance degradation analysis

Degradation due to memory contention (I)

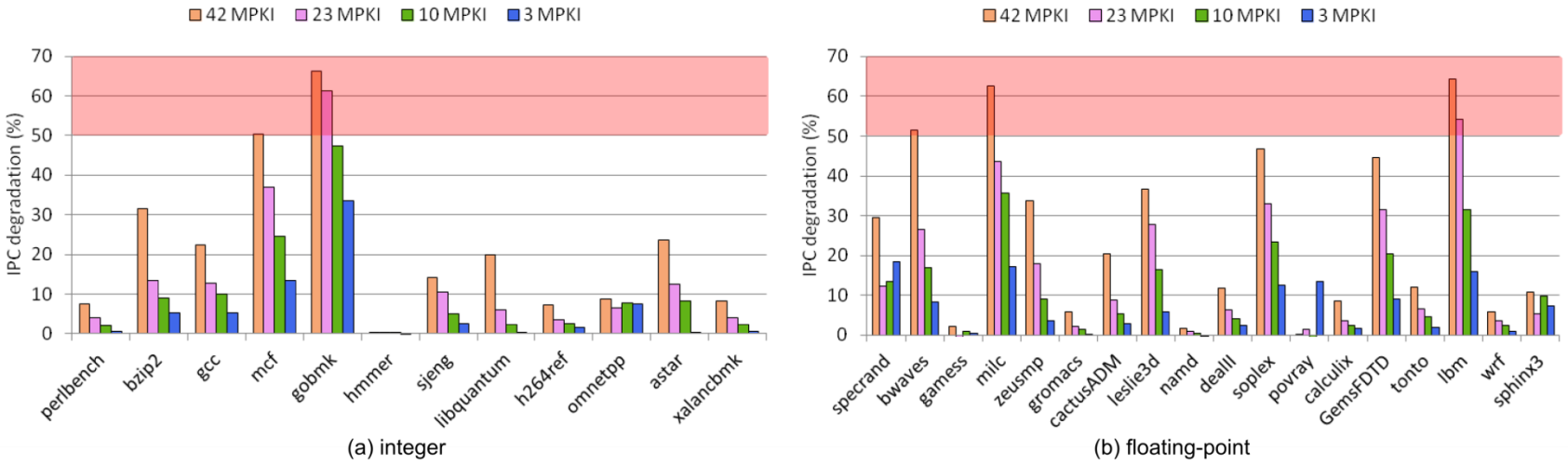


IPC degradation due to memory contention varying the MPKI of the co-runners

- Commonly, the lower the MPKI of the benchmark, the lower IPC degradation.

Performance degradation analysis

Degradation due to memory contention (I)

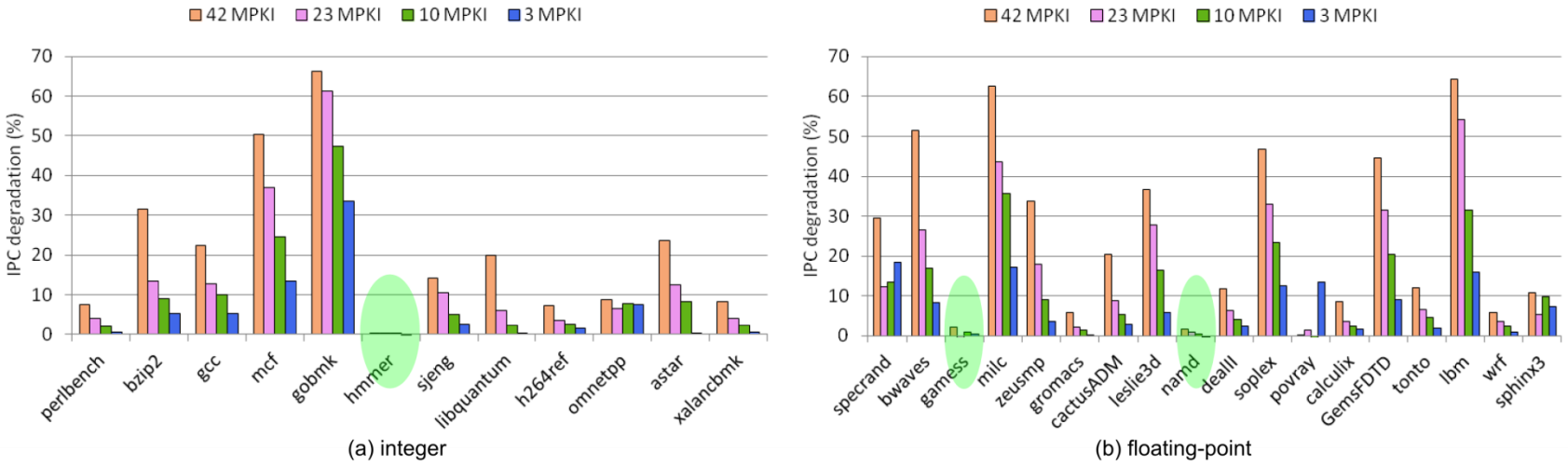


IPC degradation due to memory contention varying the MPKI of the co-runners

- Commonly, the lower the MPKI of the benchmark, the lower IPC degradation.
- Performance degradation is over 50% in some benchmarks and high MPKI of the co-runners.

Performance degradation analysis

Degradation due to memory contention (I)

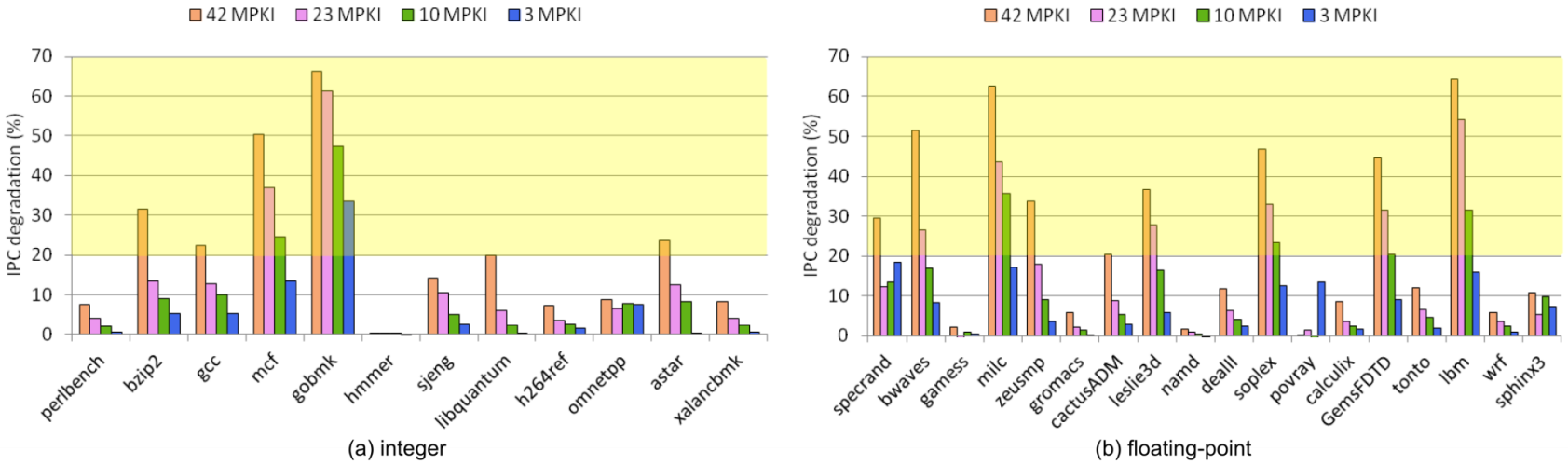


IPC degradation due to memory contention varying the MPKI of the co-runners

- Commonly, the lower the MPKI of the benchmark, the lower IPC degradation.
- Performance degradation is over 50% in some benchmarks and high MPKI of the co-runners.
- A few benchmarks are poorly affected by contention.

Performance degradation analysis

Degradation due to memory contention (I)

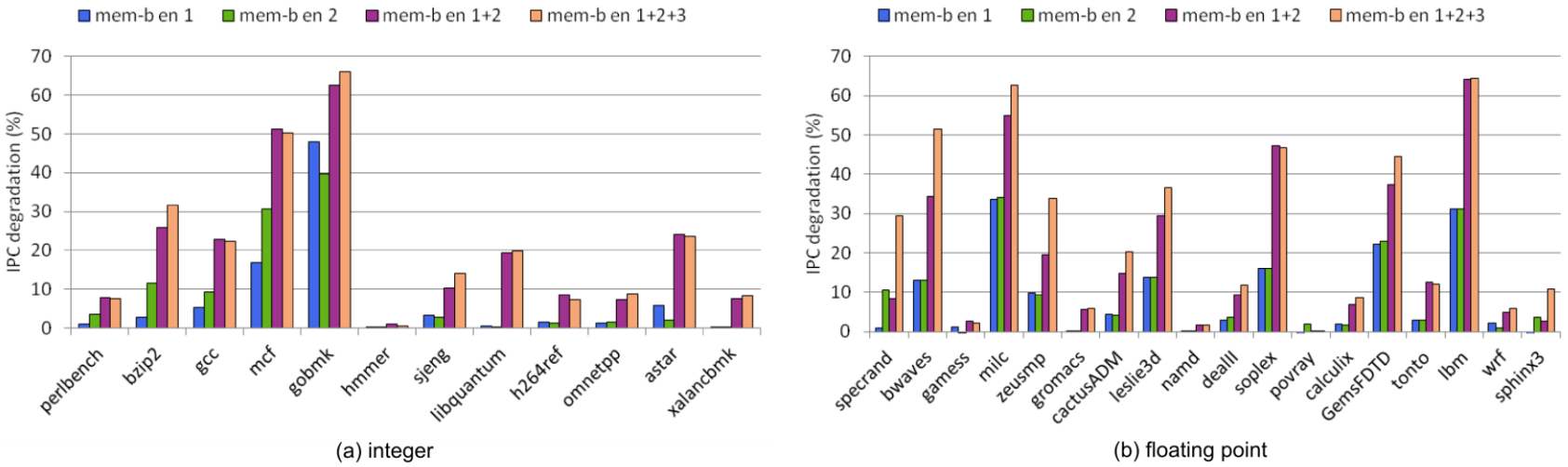


IPC degradation due to memory contention varying the MPKI of the co-runners

- Commonly, the lower the MPKI of the benchmark, the lower IPC degradation.
- Performance degradation over 50% in some benchmarks and high MPKI of the co-runners.
- A few benchmarks are poorly affected by contention.
- Performance degradation is over 20% in most benchmarks and different MPKI of the co-runners.

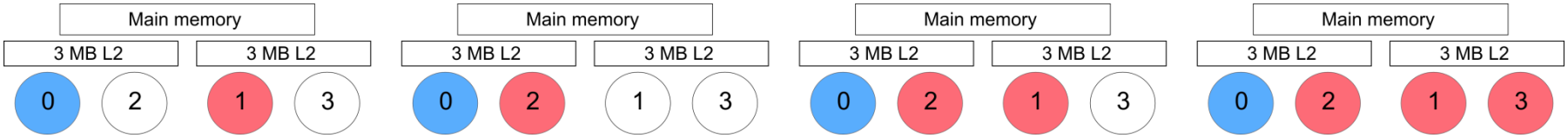
Performance degradation analysis

Degradation due to memory contention (II)



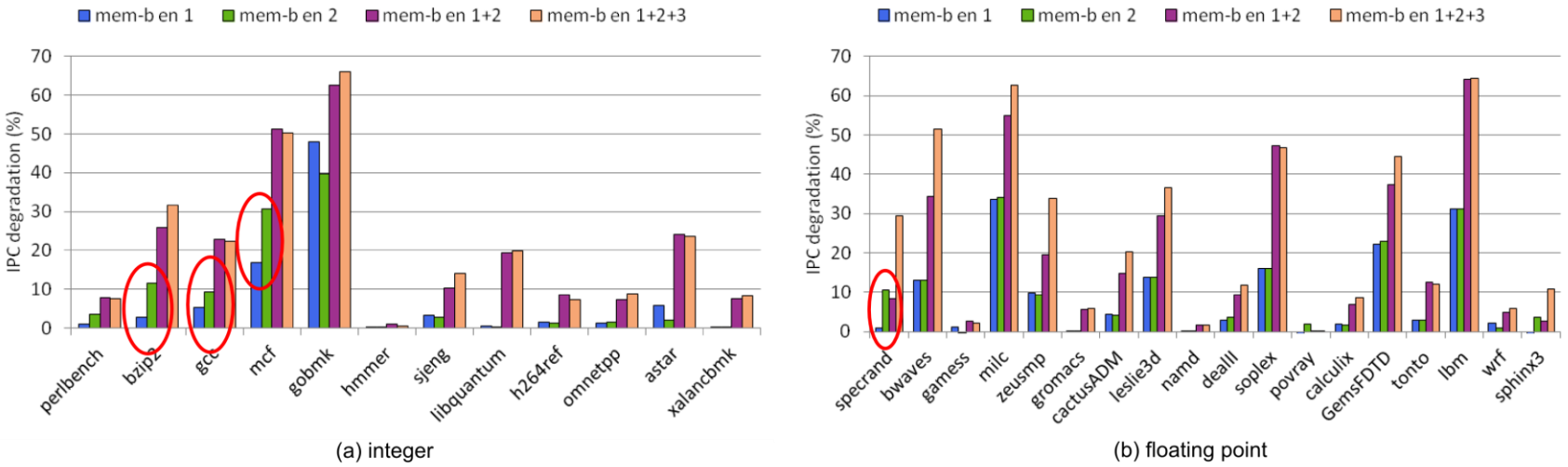
IPC degradation due to memory contention varying the number of co-runners

Four scenarios are analyzed:



Performance degradation analysis

Degradation due to memory contention (II)

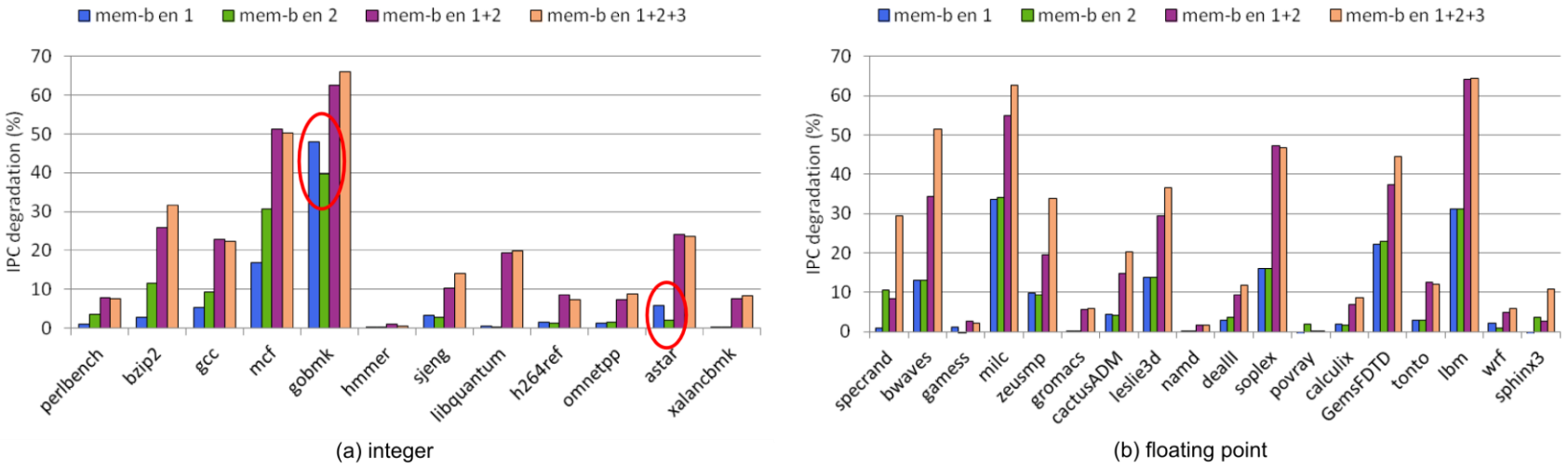


IPC degradation due to memory contention varying the number of co-runners

- Some benchmarks suffer higher IPC degradation when the co-runner runs in the other bi-core, since memory is more frequently accessed.

Performance degradation analysis

Degradation due to memory contention (II)

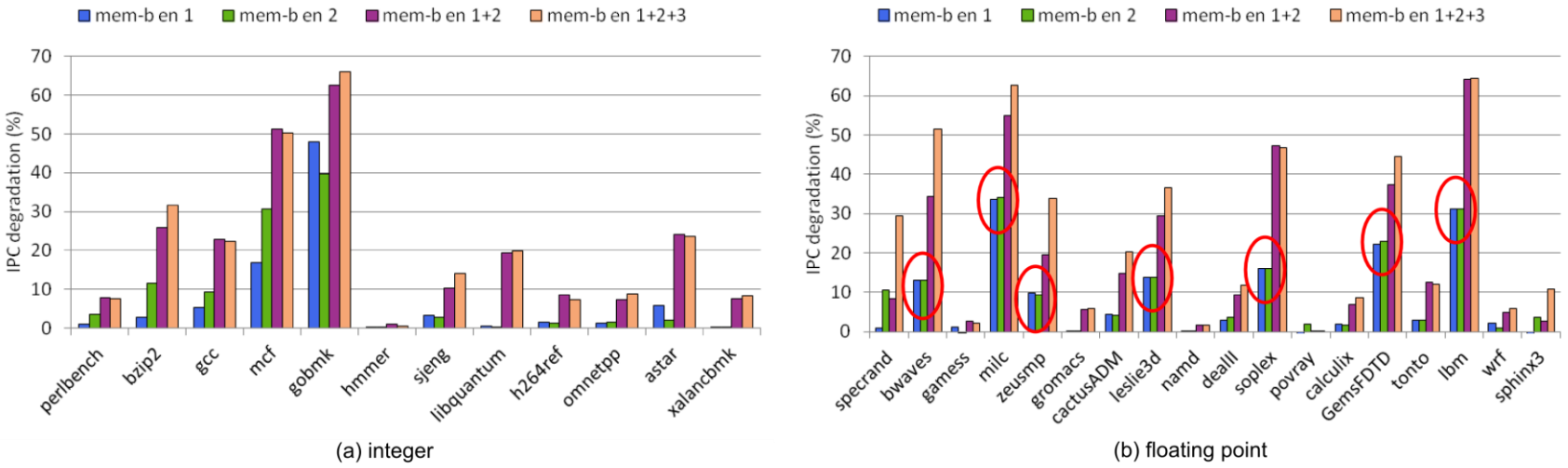


IPC degradation due to memory contention varying the number of co-runners

- Some benchmarks suffer higher IPC degradation when the co-runner runs in the other bi-core, since memory is more frequently accessed.
- Other benchmarks suffer higher IPC degradation when the co-runner runs in the same bi-core. This can be caused by L2 cache conflicts or L2 bandwidth.

Performance degradation analysis

Degradation due to memory contention (II)

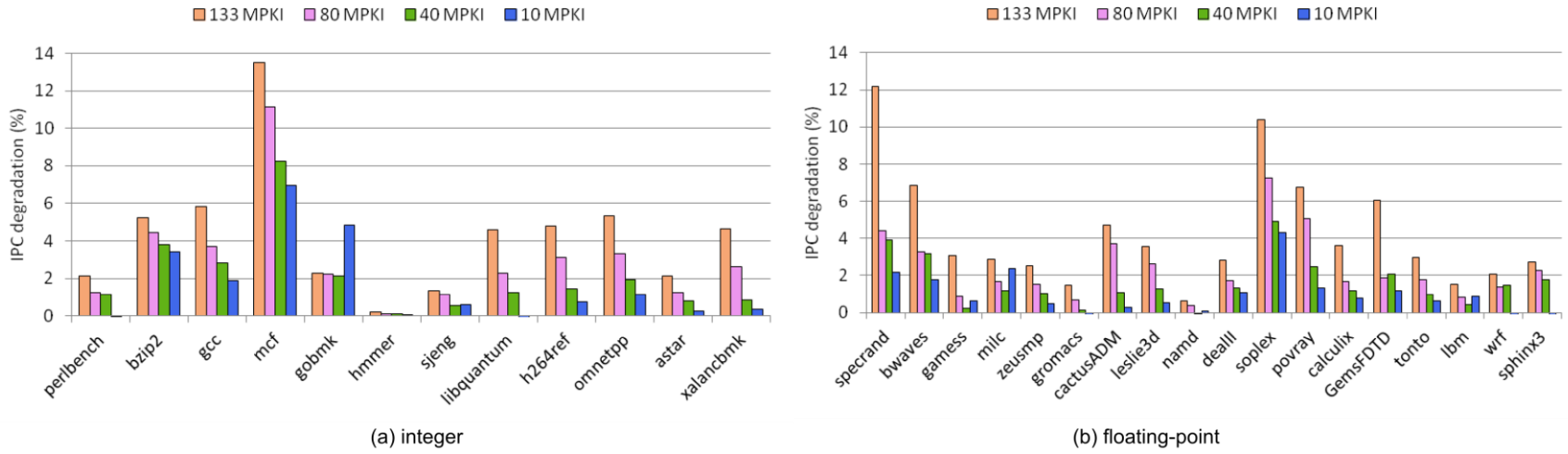


IPC degradation due to memory contention varying the number of co-runners

- Some benchmarks suffer higher IPC degradation when the co-runner runs in the other bi-core since memory is more frequently accessed.
- Other benchmarks suffer higher IPC degradation when the co-runner runs in the same bi-core. This can be caused by L2 cache conflicts or L2 bandwidth.
- In the common case, both degradations are similar.

Performance degradation analysis

Degradation due to L2 contention

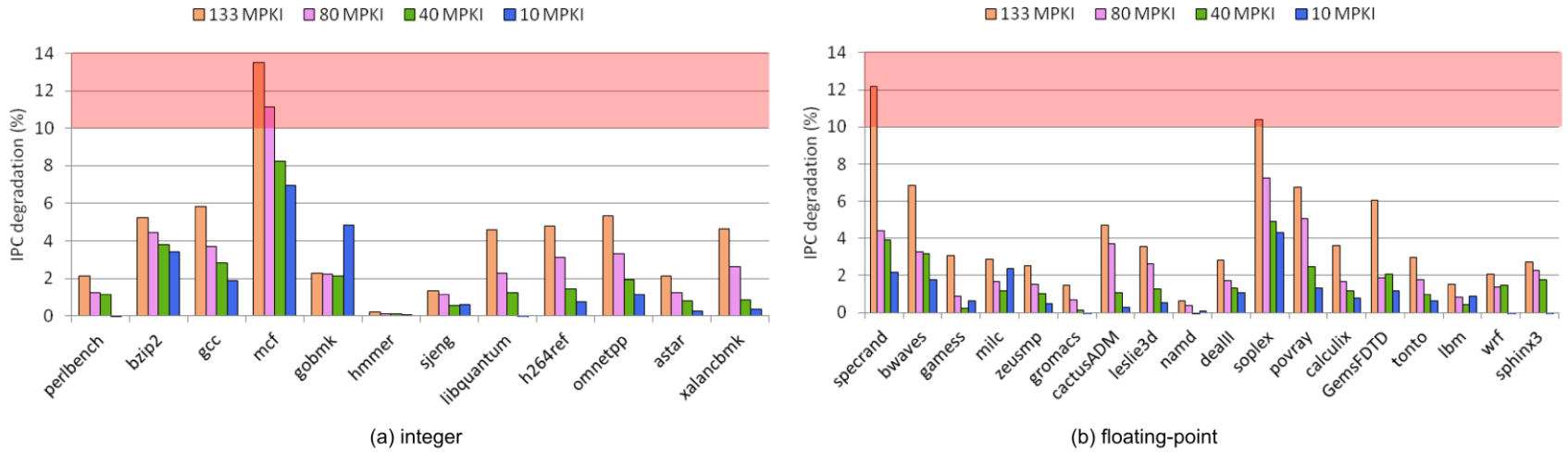


IPC degradation due to memory contention varying the MPKI of the co-runners

- Only the benchmark and one co-runner are involved.

Performance degradation analysis

Degradation due to L2 contention

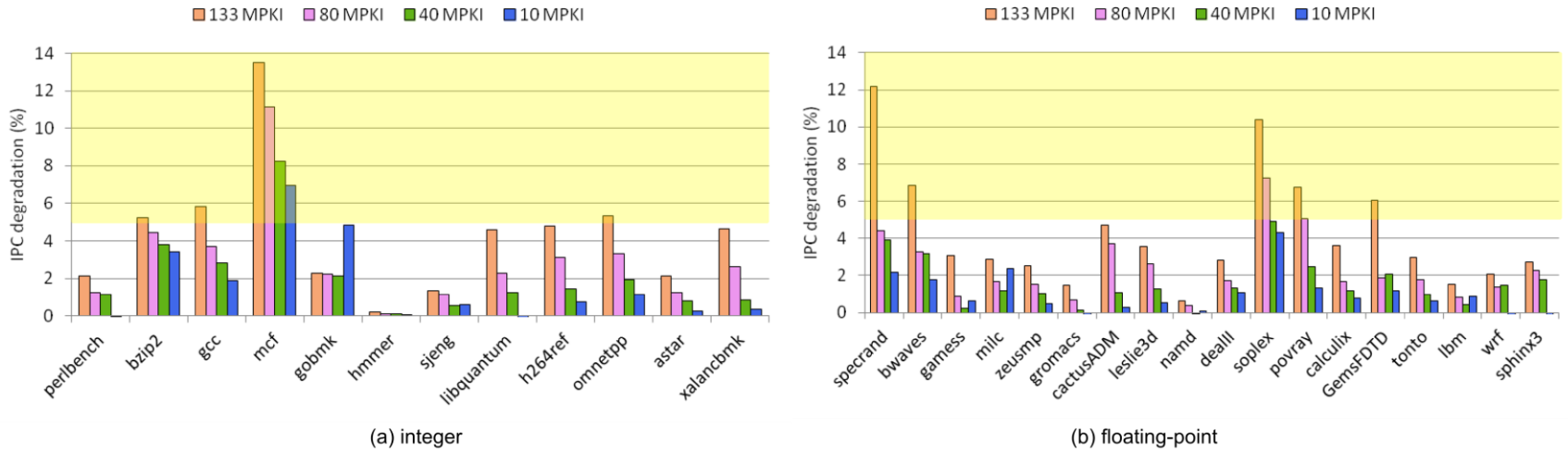


IPC degradation due to memory contention varying the MPKI of the co-runners

- Only the benchmark and one co-runner are involved.
- Three benchmarks present high IPC degradation with an L2-bounded co-runner over 10%.

Performance degradation analysis

Degradation due to L2 contention

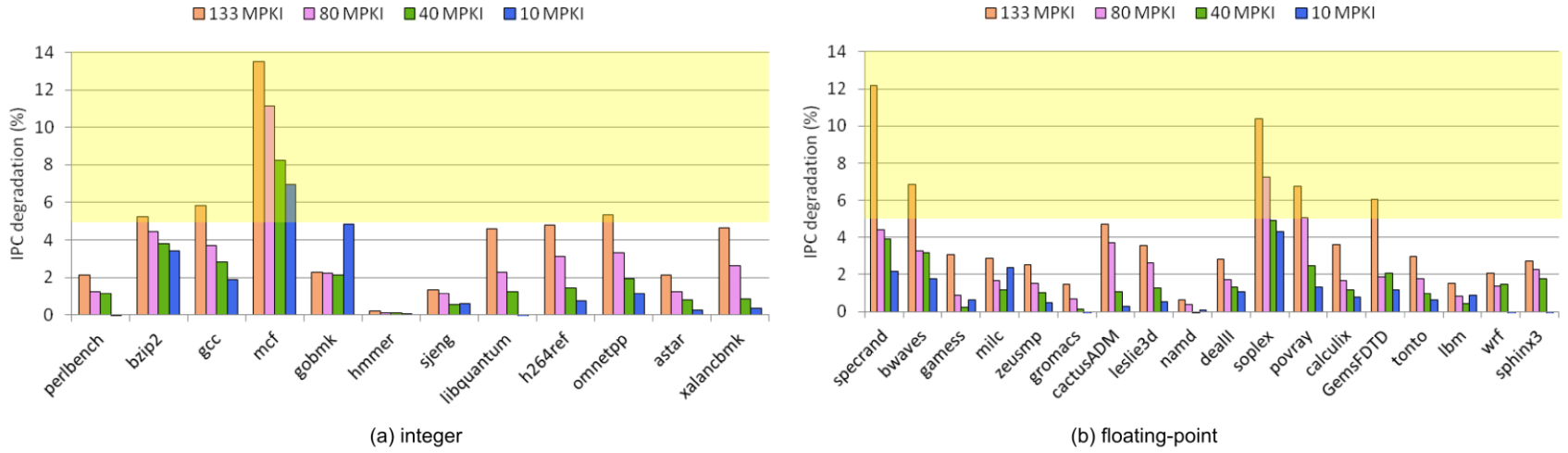


IPC degradation due to memory contention varying the MPKI of the co-runners

- Only the benchmark and one co-runner are involved.
- Three benchmarks present high IPC degradation with an L2-bounded co-runner over 10%.
- About half of the benchmarks present an IPC degradation close (or over) 5% due to L2 bandwidth.

Performance degradation analysis

Degradation due to L2 contention



IPC degradation due to memory contention varying the MPKI of the co-runners

Although this degradation is lower than the caused by main memory contention, since the trend is to increase the number of cores and shared caches we claim the necessity of a cache-hierarchy bandwidth aware scheduling and not only memory aware scheduling.

Cache-hierarchy memory aware scheduling

Algorithm 1 Cache-hierarchy memory aware scheduler

Block the executing processes and place them at the queue tail.
for each process P executed in the last quantum **do**
  for each cache level L **do**
    Update BTR for process P in cache level L
  end for
end for
while there are unfinished jobs **do**
  BW_{Remain} = average memory bandwidth
  Select the process at the queue head and update BW_{Remain}
  while selected processes $<$ cores **do**
    select the process that maximizes

$$FITNESS(p) = \frac{1}{\left| \frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required} \right|}$$

    and update BW_{Remain} and CPU_{Remain}
  end while
  for $i = max\_cache\_level$ downto level 2 **do**

$$AVG_{BTR}(L_i) = \frac{\sum BTR \ \ of \ \ L(i-1)}{\#Caches \ \ at \ \ Li}$$

    for each cache in level L_i **do**
      BW_{Remain} = $AVG_{BTR}(L_i)$
      while #selected processes for the cache $<$ # cores sharing the cache **do**
        Select the process that maximizes the FITNESS(p) function and update BW_{Remain} and CPU_{Remain}
      end while
    end for
  end for
  Unblock the processes, and allocate them in the chosen core.
  Sleep during the quantum.
end while

- Addresses the target bandwidth at each contention point.
- Schedules the processes in n steps (as many as cache levels).
- Top-down approach: from the MM to the L1 cache.
  - Final step allocates the processes to cores.

Cache-hierarchy memory aware scheduling

Algorithm 1 Cache-hierarchy memory aware scheduler

Block the executing processes and place them at the queue tail.
for each process P executed in the last quantum **do**
  for each cache level L **do**
    Update BTR for process P in cache level L
  end for
end for
while there are unfinished jobs **do**
  BW_{Remain} = average memory bandwidth
  Select the process at the queue head and update BW_{Remain}
  while selected processes $<$ cores **do**
    select the process that maximizes

$$FITNESS(p) = \frac{1}{\left| \frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required} \right|}$$

    and update BW_{Remain} and CPU_{Remain}
  end while
  for $i = max\_cache\_level$ downto level 2 **do**

$$AVG_{BTR}(L_i) = \frac{\sum BTR \ \ of \ \ L(i-1)}{\#Caches \ \ at \ \ Li}$$

    for each cache in level L_i **do**
      BW_{Remain} = $AVG_{BTR}(L_i)$
      while #selected processes for the cache $<$ # cores sharing the cache **do**
        Select the process that maximizes the FITNESS(p) function and update BW_{Remain} and CPU_{Remain}
      end while
    end for
  end for
  Unblock the processes, and allocate them in the chosen core.
  Sleep during the quantum.
end while

- When a quantum expires …
- Update the BTR values in each cache level for each executed process.
- Use these values as predicted BTR for the next quantum.

Cache-hierarchy memory aware scheduling

Algorithm 1 Cache-hierarchy memory aware scheduler

Block the executing processes and place them at the queue tail.
for each process P executed in the last quantum **do**
  for each cache level L **do**
    Update BTR for process P in cache level L
  end for
end for
while there are unfinished jobs **do**
  BW_{Remain} = average memory bandwidth
  Select the process at the queue head and update BW_{Remain}
  while selected processes $<$ cores **do**
    select the process that maximizes

$$FITNESS(p) = \frac{1}{\left| \frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required} \right|}$$

    and update BW_{Remain} and CPU_{Remain}
  end while
  for $i = max\_cache\_level$ downto level 2 **do**

$$AVG_{BTR}(L_i) = \frac{\sum BTR \ \ of \ \ L(i-1)}{\#Caches \ \ at \ \ Li}$$

    for each cache in level L_i **do**
      BW_{Remain} = $AVG_{BTR}(L_i)$
      while #selected processes for the cache $<$ # cores sharing the cache **do**
        Select the process that maximizes the FITNESS(p) function and update BW_{Remain} and CPU_{Remain}
      end while
    end for
  end for
  Unblock the processes, and allocate them in the chosen core.
  Sleep during the quantum.
end while

- BW_{Remain} is set to the total number of memory requests divided by the total execution time of the processes in stand-alone execution.
- Unfinished jobs are kept in a software queue structure.
- The process at the queue head is always selected to avoid process starvation.

Cache-hierarchy memory aware scheduling

Algorithm 1 Cache-hierarchy memory aware scheduler

Block the executing processes and place them at the queue tail.
for each process P executed in the last quantum **do**
  for each cache level L **do**
    Update BTR for process P in cache level L
  end for
end for
while there are unfinished jobs **do**
  BW_{Remain} = average memory bandwidth
  Select the process at the queue head and update BW_{Remain}
  while selected processes $<$ cores **do**
    select the process that maximizes
    $$FITNESS(p) = \frac{1}{\left| \frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required} \right|}$$
    and update BW_{Remain} and CPU_{Remain}
  end while
  for $i = max\_cache\_level$ downto level 2 **do**
    $$AVG_{BTR}(L_i) = \frac{\sum BTR \ \ of \ \ L(i-1)}{\#Caches \ \ at \ \ Li}$$
    for each cache in level L_i **do**
      BW_{Remain} = $AVG_{BTR}(L_i)$
      while #selected processes for the cache $<$ # cores sharing the cache **do**
        Select the process that maximizes the FITNESS(p) function and update BW_{Remain} and CPU_{Remain}
      end while
    end for
  end for
  Unblock the processes, and allocate them in the chosen core.
  Sleep during the quantum.
end while

- Then, the scheduler selects the remaining c minus 1 processes that maximize the Fitness function*.
  - That estimates the gap between the BTR_{Remain} and the predicted BTR of each process.
- BW_{Remain} and CPU_{Remain} (# of cores) are updated each time a process is selected.
- The result of this step is the list of processes to be executed considering taking into account the MM bandwidth constraint.

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

Cache-hierarchy memory aware scheduling

Algorithm 1 Cache-hierarchy memory aware scheduler

Block the executing processes and place them at the queue tail.
for each process P executed in the last quantum **do**
  for each cache level L **do**
    Update BTR for process P in cache level L
  end for
end for
while there are unfinished jobs **do**
  BW_{Remain} = average memory bandwidth
  Select the process at the queue head and update BW_{Remain}
  while selected processes $<$ cores **do**
    select the process that maximizes

$$FITNESS(p) = \frac{1}{\left| \frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required} \right|}$$

    and update BW_{Remain} and CPU_{Remain}
  end while
  for $i = max\_cache\_level$ downto level 2 **do**

$$AVG_{BTR}(L_i) = \frac{\sum BTR \ \ of \ \ L(i-1)}{\#Caches \ \ at \ \ Li}$$

    for each cache in level L_i **do**
      BW_{Remain} = $AVG_{BTR}(L_i)$
      while #selected processes for the cache $<$ # cores sharing the cache **do**
        Select the process that maximizes the FITNESS(p) function and update BW_{Remain} and CPU_{Remain}
      end while
    end for
  end for
  Unblock the processes, and allocate them in the chosen core.
  Sleep during the quantum.
end while

- For each level in the cache hierarchy with shared caches:
  - AVG_BTR is set to the average BTR of the selected processes divided by the number of cache structures.
  - BWremain is set to AVG_BTR for each cache and the processes are selected using the Fitness function, updating the BW remain and CPU remain.
  - The iteration in the last shared cache level allocates the processes to the concrete cores in its cache structure.

Example

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

Average memory bandwidth = 30

P Ident
BTR L2
BTR L1
Fitness

1. Select the first process in the queue:

P0	P1	P2	P3	P4	P5	P6	P7
12	20	5	0	9	15	25	8
80	90	25	40	65	90	45	70

Selected processes:

Example

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

Average memory bandwidth = 30

P Ident
BTR L2
BTR L1
Fitness

1. Select the first process in the queue:

P0	P1	P2	P3	P4	P5	P6	P7
12	20	5	0	9	15	25	8
80	90	25	40	65	90	45	70

Update: BW_{Remain} = 30 - 12 = 18 Cpu_{Remain} = 3

P1	P2	P3	P4	P5	P6	P7
20	5	0	9	15	25	8
90	25	40	65	90	45	70

Selected processes:

P0
12
80

Example

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

Average memory bandwidth = 30

P Ident
BTR L2
BTR L1
Fitness

$$FITNESS\ (p) = \frac{1}{\left|\frac{18}{3} - BW^{P}_{required}\right|}$$

1. Select the first process in the queue:

P0	P1	P2	P3	P4	P5	P6	P7
12	20	5	0	9	15	25	8
80	90	25	40	65	90	45	70

Update: BW_{Remain} = 30 - 12 = 18 Cpu_{Remain} = 3

2. Select the process that maximizes the fitness function:

P1	P2	P3	P4	P5	P6	P7
20	5	0	9	15	25	8
90	25	40	65	90	45	70
0.07	1	0.17	0.33	0.11	0.05	0.5

Selected processes:

P0
12
80

Example

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

Average memory bandwidth = 30

P Ident
BTR L2
BTR L1
Fitness

$$FITNESS\ (p) = \frac{1}{\left|\frac{18}{3} - BW^{P}_{required}\right|}$$

1. Select the first process in the queue:

P0	P1	P2	P3	P4	P5	P6	P7
12	20	5	0	9	15	25	8
80	90	25	40	65	90	45	70

Update: BW_{Remain} = 30 - 12 = 18 Cpu_{Remain} = 3

2. Select the process that maximizes the fitness function:

P1	P2	P3	P4	P5	P6	P7
20	5	0	9	15	25	8
90	25	40	65	90	45	70
0.07	1	0.17	0.33	0.11	0.05	0.5

Update: BW_{Remain} = 18 - 5 = 13 Cpu_{Remain} = 2

P1	P3	P4	P5	P6	P7
20	0	9	15	25	8
90	40	65	90	45	70

Selected processes:

P0	P2
12	5
80	25

Example

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

Average memory bandwidth = 30

P Ident
BTR L2
BTR L1
Fitness

$$FITNESS\ (p) = \frac{1}{\left|\frac{18}{3} - BW^{P}_{required}\right|}$$

$$FITNESS\ (p) = \frac{1}{\left|\frac{13}{2} - BW^{P}_{required}\right|}$$

1. Select the first process in the queue:

P0	P1	P2	P3	P4	P5	P6	P7
12	20	5	0	9	15	25	8
80	90	25	40	65	90	45	70

Update: BW_{Remain} = 30 - 12 = 18 Cpu_{Remain} = 3

2. Select the process that maximizes the fitness function:

P1	P2	P3	P4	P5	P6	P7
20	5	0	9	15	25	8
90	25	40	65	90	45	70
0.07	1	0.17	0.33	0.11	0.05	0.5

Update: BW_{Remain} = 18 - 5 = 13 Cpu_{Remain} = 2

3. Select the process that maximizes the fitness function:

P1	P3	P4	P5	P6	P7
20	0	9	15	25	8
90	40	65	90	45	70
0.07	0.15	0.4	0.12	0.05	0.66

Selected processes:

P0	P2
12	5
80	25

Example

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

Average memory bandwidth = 30

P Ident
BTR L2
BTR L1
Fitness

$$FITNESS\ (p) = \frac{1}{\left|\frac{18}{3} - BW^{P}_{required}\right|}$$

$$FITNESS\ (p) = \frac{1}{\left|\frac{13}{2} - BW^{P}_{required}\right|}$$

1. Select the first process in the queue:

P0	P1	P2	P3	P4	P5	P6	P7
12	20	5	0	9	15	25	8
80	90	25	40	65	90	45	70

Update: BW_{Remain} = 30 - 12 = 18 Cpu_{Remain} = 3

2. Select the process that maximizes the fitness function:

P1	P2	P3	P4	P5	P6	P7
20	5	0	9	15	25	8
90	25	40	65	90	45	70
0.07	1	0.17	0.33	0.11	0.05	0.5

Update: BW_{Remain} = 18 - 5 = 13 Cpu_{Remain} = 2

3. Select the process that maximizes the fitness function:

P1	P3	P4	P5	P6	P7
20	0	9	15	25	8
90	40	65	90	45	70
0.07	0.15	0.4	0.12	0.05	0.66

Update: BW_{Remain} = 13 - 8 = 5 Cpu_{Remain} = 1

P1	P3	P4	P5	P6
20	0	9	15	25
90	40	65	90	45

Selected processes:

P0	P2	P7
12	5	8
80	25	70

Example

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

Average memory bandwidth = 30

P Ident
BTR L2
BTR L1
Fitness

1. Select the first process in the queue:

P0	P1	P2	P3	P4	P5	P6	P7
12	20	5	0	9	15	25	8
80	90	25	40	65	90	45	70

Update: BW_{Remain} = 30 - 12 = 18 Cpu_{Remain} = 3

$$FITNESS\ (p) = \frac{1}{\left|\frac{18}{3} - BW^{P}_{required}\right|}$$

2. Select the process that maximizes the fitness function:

P1	P2	P3	P4	P5	P6	P7
20	5	0	9	15	25	8
90	25	40	65	90	45	70
0.07	1	0.17	0.33	0.11	0.05	0.5

Update: BW_{Remain} = 18 - 5 = 13 Cpu_{Remain} = 2

$$FITNESS\ (p) = \frac{1}{\left|\frac{13}{2} - BW^{P}_{required}\right|}$$

3. Select the process that maximizes the fitness function:

P1	P3	P4	P5	P6	P7
20	0	9	15	25	8
90	40	65	90	45	70
0.07	0.15	0.4	0.12	0.05	0.66

Update: BW_{Remain} = 13 - 8 = 5 Cpu_{Remain} = 1

$$FITNESS\ (p) = \frac{1}{\left|\frac{5}{1} - BW^{P}_{required}\right|}$$

4. Select the process that maximizes the fitness function:

P1	P3	P4	P5	P6
20	0	9	15	25
90	40	65	90	45
0.06	0.5	0.25	0.1	0.05

Selected processes:

P0	P2	P7
12	5	8
80	25	70

Example

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

Average memory bandwidth = 30

P Ident
BTR L2
BTR L1
Fitness

1. Select the first process in the queue:

P0	P1	P2	P3	P4	P5	P6	P7
12	20	5	0	9	15	25	8
80	90	25	40	65	90	45	70

Update: BW_{Remain} = 30 - 12 = 18 Cpu_{Remain} = 3

$$FITNESS\ (p) = \frac{1}{\left|\frac{18}{3} - BW^{P}_{required}\right|}$$

2. Select the process that maximizes the fitness function:

P1	P2	P3	P4	P5	P6	P7
20	5	0	9	15	25	8
90	25	40	65	90	45	70
0.07	1	0.17	0.33	0.11	0.05	0.5

Update: BW_{Remain} = 18 - 5 = 13 Cpu_{Remain} = 2

$$FITNESS\ (p) = \frac{1}{\left|\frac{13}{2} - BW^{P}_{required}\right|}$$

3. Select the process that maximizes the fitness function:

P1	P3	P4	P5	P6	P7
20	0	9	15	25	8
90	40	65	90	45	70
0.07	0.15	0.4	0.12	0.05	0.66

Update: BW_{Remain} = 13 - 8 = 5 Cpu_{Remain} = 1

$$FITNESS\ (p) = \frac{1}{\left|\frac{5}{1} - BW^{P}_{required}\right|}$$

4. Select the process that maximizes the fitness function:

P1	P3	P4	P5	P6
20	0	9	15	25
90	40	65	90	45
0.06	0.5	0.25	0.1	0.05

Selected processes:

P0	P2	P7	P3
12	5	8	0
80	25	70	40

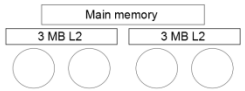
Example

$$AVG_{BTR}(L_i) = \frac{\sum BTR\ of\ L(i-1)}{\#\ caches\ at\ L_i} = \frac{\sum 80 + 25 + 70 + 40}{2} = 107.5$$

$$FITNESS\ (p) = \frac{1}{\left| \frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required} \right|}$$

P Ident
BTR L2
BTR L1
Fitness

P0	P2	P7	P3
12	5	8	0
80	25	70	40



Example

$$AVG_{BTR}(L_i) = \frac{\sum BTR\ of\ L(i-1)}{\#\ caches\ at\ L_i} = \frac{\sum 80 + 25 + 70 + 40}{2} = 107.5$$

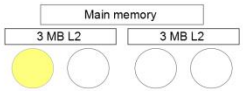
$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

1. Set $BW_{Remain} = AVG_{BTR}$

Select the process that maximizes the fitness function:

P Ident	P0	P2	P7	P3
BTR L2	12	5	8	0
BTR L1	80	25	70	40
Fitness	0.04	0.03	0.06	0.07

$$FITNESS\ (p) = \frac{1}{\left|\frac{107.5}{2} - BW^{P}_{required}\right|}$$



Example

$$AVG_{BTR}(L_i) = \frac{\sum BTR\ of\ L(i-1)}{\#\ caches\ at\ L_i} = \frac{\sum 80 + 25 + 70 + 40}{2} = 107.5$$

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

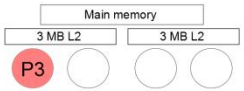
P Ident
BTR L2
BTR L1
Fitness

$$FITNESS\ (p) = \frac{1}{\left|\frac{107.5}{2} - BW^{P}_{required}\right|}$$

1. Set $BW_{Remain} = AVG_{BTR}$

Select the process that maximizes the fitness function:

P0	P2	P7	P3
12	5	8	0
80	25	70	40
0.04	0.03	0.06	0.07



Update: $BW_{Remain} = 107.5 - 40 = 67.5$ $Cpu_{Remain} = 1$

P0	P2	P7
12	5	8
80	25	70

Example

$$AVG_{BTR}(L_i) = \frac{\sum BTR\ of\ L(i-1)}{\#\ caches\ at\ L_i} = \frac{\sum 80 + 25 + 70 + 40}{2} = 107.5$$

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

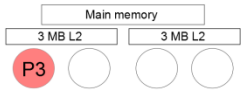
P Ident
BTR L2
BTR L1
Fitness

$$FITNESS\ (p) = \frac{1}{\left|\frac{107.5}{2} - BW^{P}_{required}\right|}$$

1. Set BW_{Remain} = AVG_{BTR}

Select the process that maximizes the fitness function:

P0	P2	P7	P3
12	5	8	0
80	25	70	40
0.04	0.03	0.06	0.07



Update: BW_{Remain} = 107.5 - 40 = 67.5 Cpu_{Remain} = 1

2. Select the process that maximizes the fitness function:

$$FITNESS\ (p) = \frac{1}{\left|\frac{67.5}{1} - BW^{P}_{required}\right|}$$

P0	P2	P7
12	5	8
80	25	70
0.08	0.02	0.4

Example

$$AVG_{BTR}(L_i) = \frac{\sum BTR\ of\ L(i-1)}{\#\ caches\ at\ L_i} = \frac{\sum 80 + 25 + 70 + 40}{2} = 107.5$$

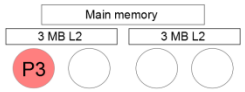
$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

1. Set $BW_{Remain} = AVG_{BTR}$

Select the process that maximizes the fitness function:

P Ident	P0	P2	P7	P3
BTR L2	12	5	8	0
BTR L1	80	25	70	40
Fitness	0.04	0.03	0.06	0.07

$$FITNESS\ (p) = \frac{1}{\left|\frac{107.5}{2} - BW^{P}_{required}\right|}$$



Update: $BW_{Remain} = 107.5 - 40 = 67.5$ $Cpu_{Remain} = 1$

2. Select the process that maximizes the fitness function:

P0	P2	P7
12	5	8
80	25	70
0.08	0.02	0.4

$$FITNESS\ (p) = \frac{1}{\left|\frac{67.5}{1} - BW^{P}_{required}\right|}$$

P0	P2
12	5
80	25

Example

$$AVG_{BTR}(L_i) = \frac{\sum BTR\ of\ L(i-1)}{\#\ caches\ at\ L_i} = \frac{\sum 80 + 25 + 70 + 40}{2} = 107.5$$

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

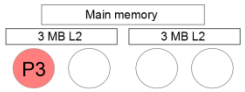
P Ident
BTR L2
BTR L1
Fitness

$$FITNESS\ (p) = \frac{1}{\left|\frac{107.5}{2} - BW^{P}_{required}\right|}$$

1. Set BW_{Remain} = AVG_{BTR}

Select the process that maximizes the fitness function:

P0	P2	P7	P3
12	5	8	0
80	25	70	40
0.04	0.03	0.06	0.07



Update: BW_{Remain} = 107.5 - 40 = 67.5 Cpu_{Remain} = 1

2. Select the process that maximizes the fitness function:

$$FITNESS\ (p) = \frac{1}{\left|\frac{67.5}{1} - BW^{P}_{required}\right|}$$

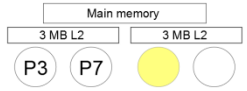
P0	P2	P7
12	5	8
80	25	70
0.08	0.02	0.4

3. Set BW_{Remain} = AVG_{BTR}

Select the process that maximizes the fitness function:

$$FITNESS\ (p) = \frac{1}{\left|\frac{107.5}{2} - BW^{P}_{required}\right|}$$

P0	P2
12	5
80	25
0.04	0.01



Example

$$AVG_{BTR}(L_i) = \frac{\sum BTR\ of\ L(i-1)}{\#\ caches\ at\ L_i} = \frac{\sum 80+25+70+40}{2} = 107.5$$

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

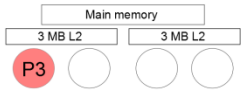
P Ident
BTR L2
BTR L1
Fitness

$$FITNESS\ (p) = \frac{1}{\left|\frac{107.5}{2} - BW^{P}_{required}\right|}$$

1. Set BW_{Remain} = AVG_{BTR}

Select the process that maximizes the fitness function:

P0	P2	P7	P3
12	5	8	0
80	25	70	40
0.04	0.03	0.06	0.07



Update: BW_{Remain} = 107.5 - 40 = 67.5 Cpu_{Remain} = 1

2. Select the process that maximizes the fitness function:

$$FITNESS\ (p) = \frac{1}{\left|\frac{67.5}{1} - BW^{P}_{required}\right|}$$

P0	P2	P7
12	5	8
80	25	70
0.08	0.02	0.4

3. Set BW_{Remain} = AVG_{BTR}

Select the process that maximizes the fitness function:

$$FITNESS\ (p) = \frac{1}{\left|\frac{107.5}{2} - BW^{P}_{required}\right|}$$

P0	P2
12	5
80	25
0.04	0.01

Update: BW_{Remain} = 107.5 - 80 = 27.5 Cpu_{Remain} = 1

P2
5
25

Example

$$AVG_{BTR}(L_i) = \frac{\sum BTR\ of\ L(i-1)}{\#\ caches\ at\ L_i} = \frac{\sum 80 + 25 + 70 + 40}{2} = 107.5$$

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

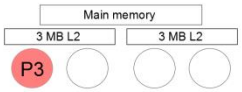
P Ident
BTR L2
BTR L1
Fitness

$$FITNESS\ (p) = \frac{1}{\left|\frac{107.5}{2} - BW^{P}_{required}\right|}$$

1. Set BW_{Remain} = AVG_{BTR}

Select the process that maximizes the fitness function:

P0	P2	P7	P3
12	5	8	0
80	25	70	40
0.04	0.03	0.06	0.07



Update: BW_{Remain} = 107.5 - 40 = 67.5 Cpu_{Remain} = 1

2. Select the process that maximizes the fitness function:

$$FITNESS\ (p) = \frac{1}{\left|\frac{67.5}{1} - BW^{P}_{required}\right|}$$

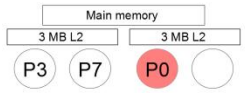
P0	P2	P7
12	5	8
80	25	70
0.08	0.02	0.4

3. Set BW_{Remain} = AVG_{BTR}

Select the process that maximizes the fitness function:

$$FITNESS\ (p) = \frac{1}{\left|\frac{107.5}{2} - BW^{P}_{required}\right|}$$

P0	P2
12	5
80	25
0.04	0.01

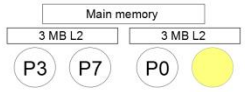


Update: BW_{Remain} = 107.5 - 80 = 27.5 Cpu_{Remain} = 1

4. Select the process that maximizes the fitness function:

$$FITNESS\ (p) = \frac{1}{\left|\frac{27.5}{1} - BW^{P}_{required}\right|}$$

P2
5
25
0.4



Example

$$AVG_{BTR}(L_i) = \frac{\sum BTR\ of\ L(i-1)}{\#\ caches\ at\ L_i} = \frac{\sum 80 + 25 + 70 + 40}{2} = 107.5$$

$$FITNESS\ (p) = \frac{1}{\left|\frac{BW_{Remain}}{CPU_{Remain}} - BW^{P}_{required}\right|}$$

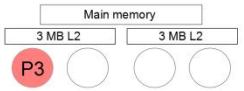
P Ident
BTR L2
BTR L1
Fitness

$$FITNESS\ (p) = \frac{1}{\left|\frac{107.5}{2} - BW^{P}_{required}\right|}$$

1. Set $BW_{Remain} = AVG_{BTR}$

Select the process that maximizes the fitness function:

P0	P2	P7	P3
12	5	8	0
80	25	70	40
0.04	0.03	0.06	0.07



Update: $BW_{Remain} = 107.5 - 40 = 67.5$ $Cpu_{Remain} = 1$

2. Select the process that maximizes the fitness function:

$$FITNESS\ (p) = \frac{1}{\left|\frac{67.5}{1} - BW^{P}_{required}\right|}$$

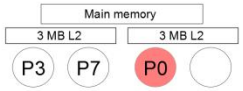
P0	P2	P7
12	5	8
80	25	70
0.08	0.02	0.4

3. Set $BW_{Remain} = AVG_{BTR}$

Select the process that maximizes the fitness function:

$$FITNESS\ (p) = \frac{1}{\left|\frac{107.5}{2} - BW^{P}_{required}\right|}$$

P0	P2
12	5
80	25
0.04	0.01

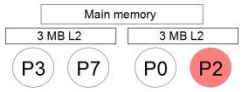


Update: $BW_{Remain} = 107.5 - 80 = 27.5$ $Cpu_{Remain} = 1$

4. Select the process that maximizes the fitness function:

$$FITNESS\ (p) = \frac{1}{\left|\frac{27.5}{1} - BW^{P}_{required}\right|}$$

P2
5
25
0.4



Evaluation methodology

- Evaluation is performed in the experimental platform.
- Implement the proposal in a user level scheduler (in a real machine)
  - At the end of each quantum, the scheduler uses:
    - PTRACE_ATTACH to block the execution of the processes.
    - PTRACE_DETACH to unblock the execution of the processes.
    - Sched_setaffinity to allocate processes in cores.
  - To evaluate the performance, a set of 10 mixes with eight benchmarks was designed.

Evaluation methodology

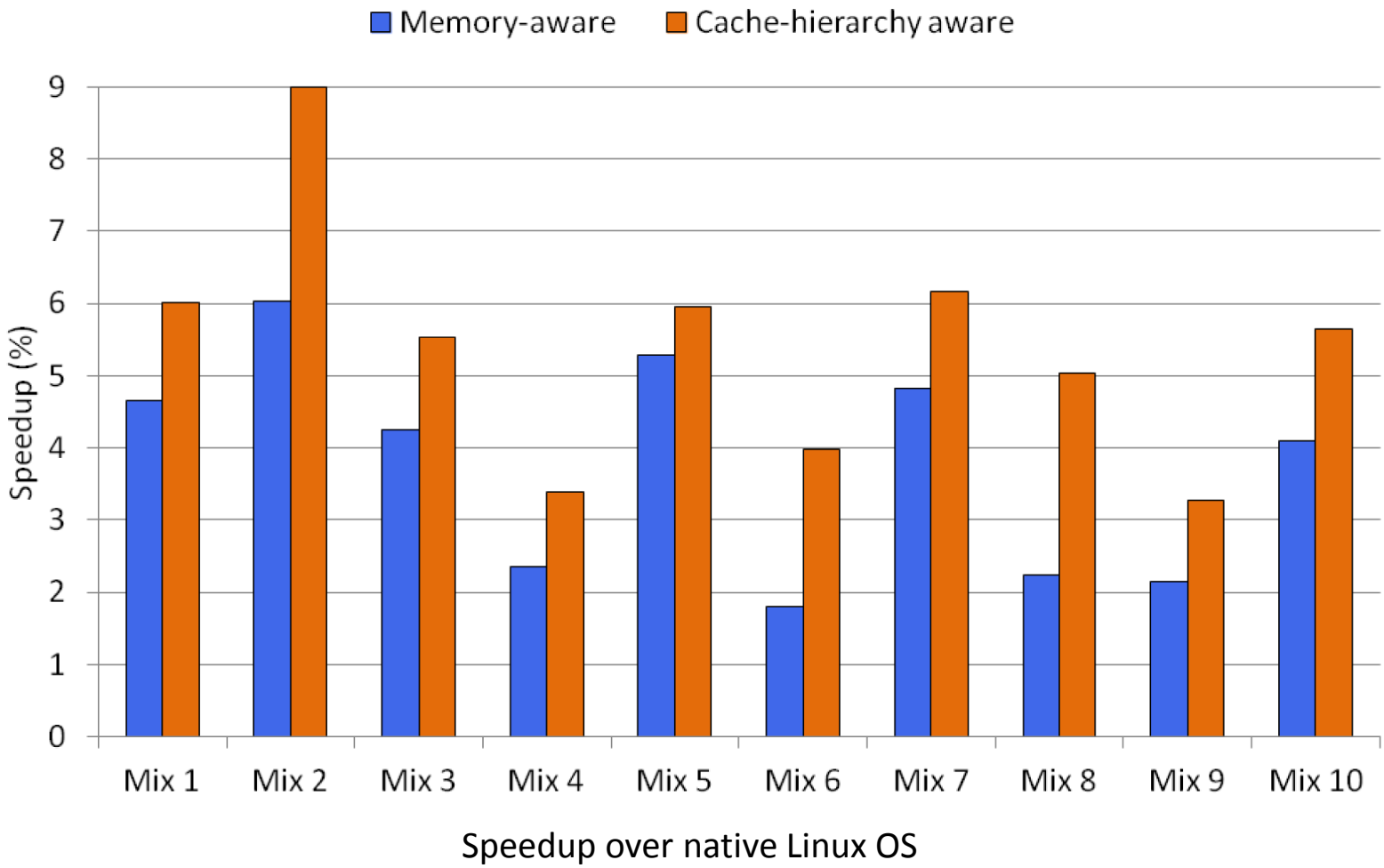
- The performance of the proposal is evaluated against:
  - Memory-aware scheduler *.
  - Linux OS scheduler.

- The schedulers differ in the selection process:
  - Memory-aware scheduler selects proper processes but do not allocate them to cores.
  - Cache-hierarchy scheduler selects the processes and allocates them to cores.

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

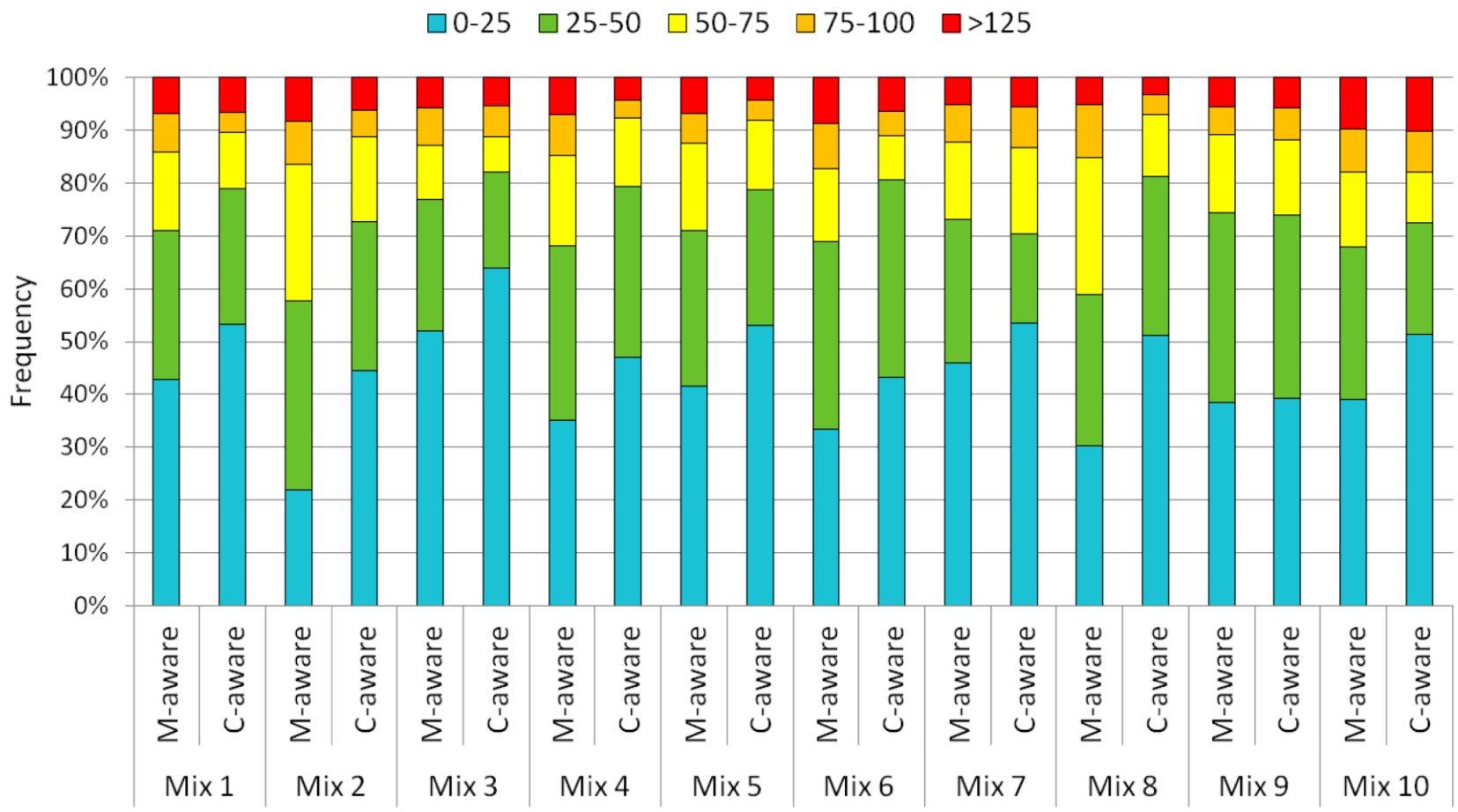
Scheduler performance

Speedup



Scheduler performance

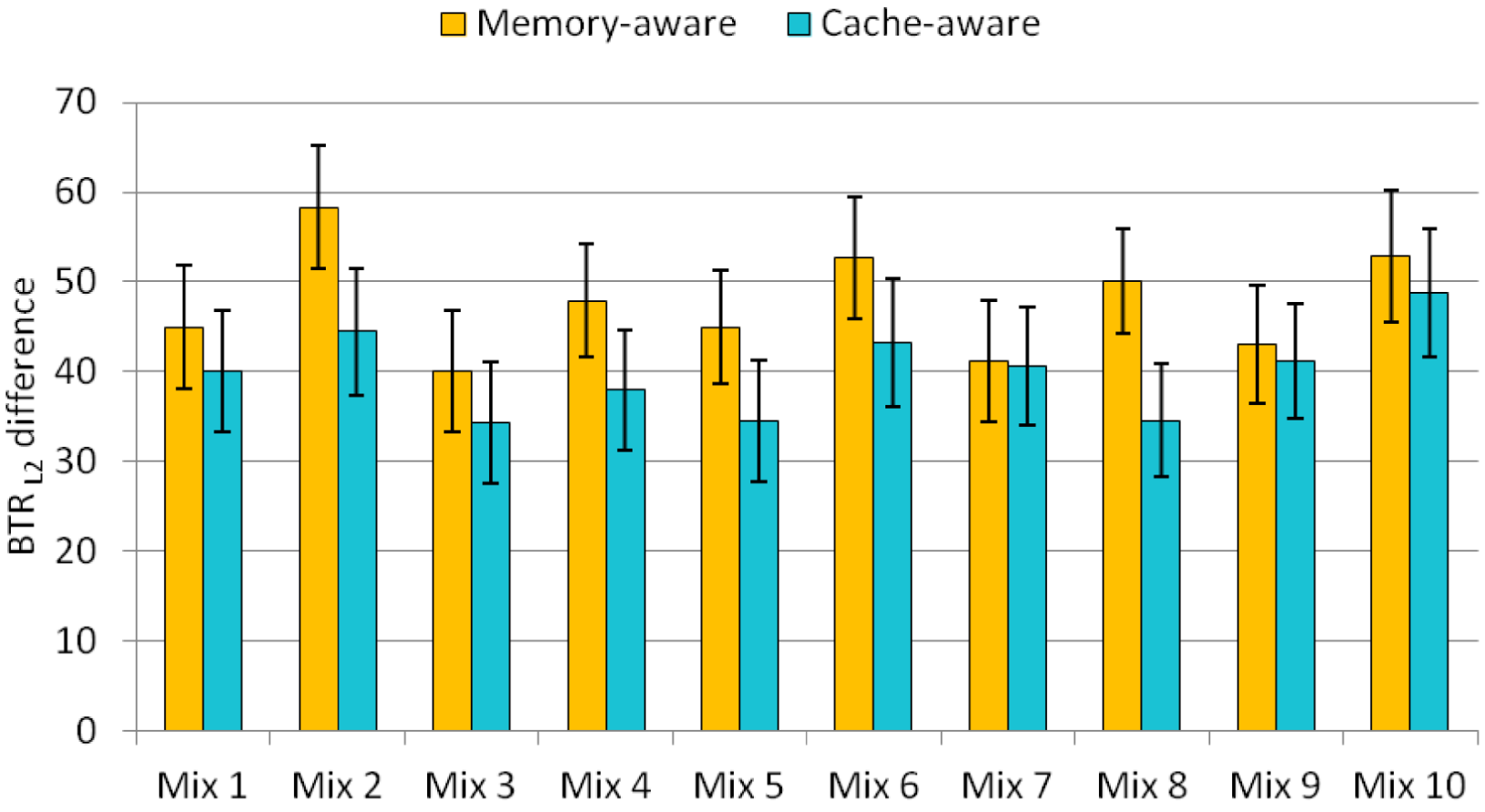
BTR balancing: histogram



BTR differences between the L2 shared caches

Scheduler performance

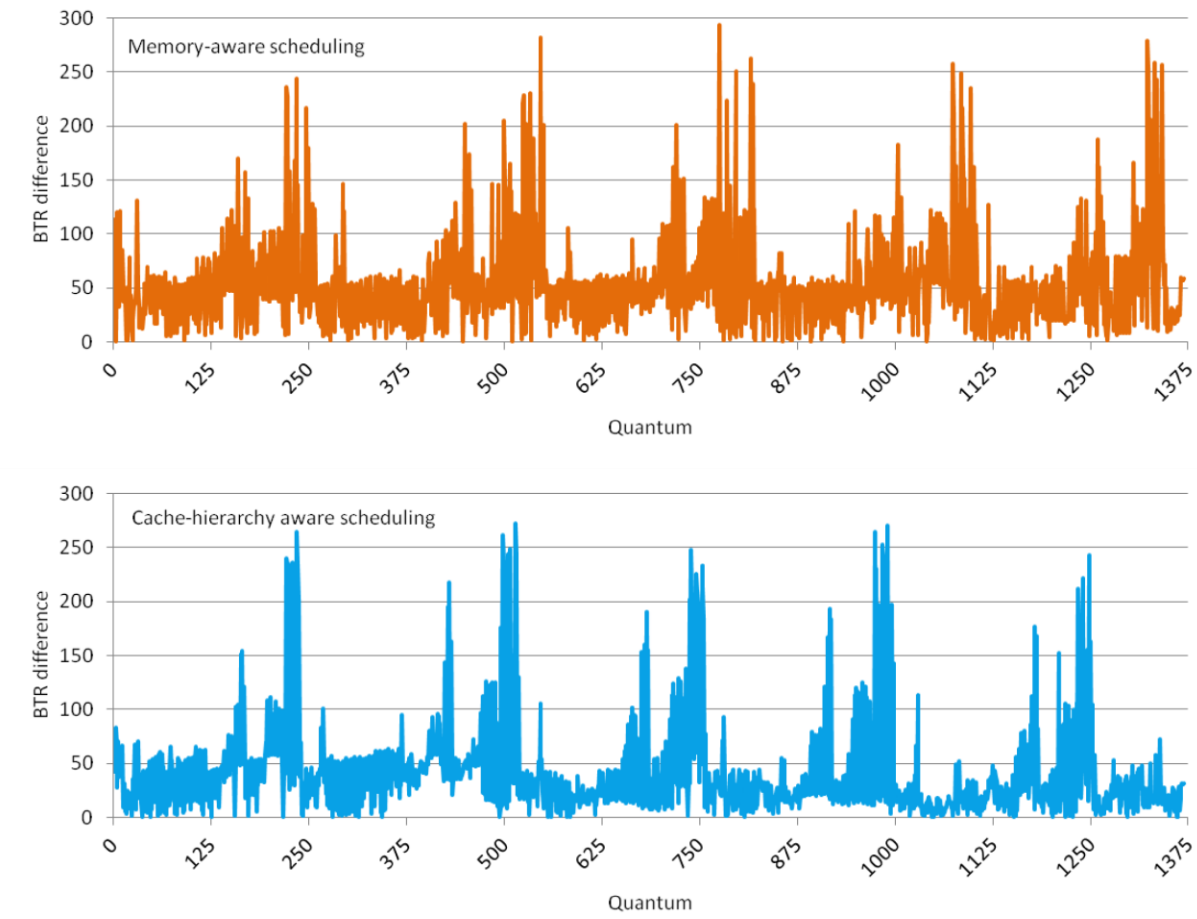
BTR balancing: average



Average and variance of the difference between the BTRs of the L2 caches

Scheduler performance

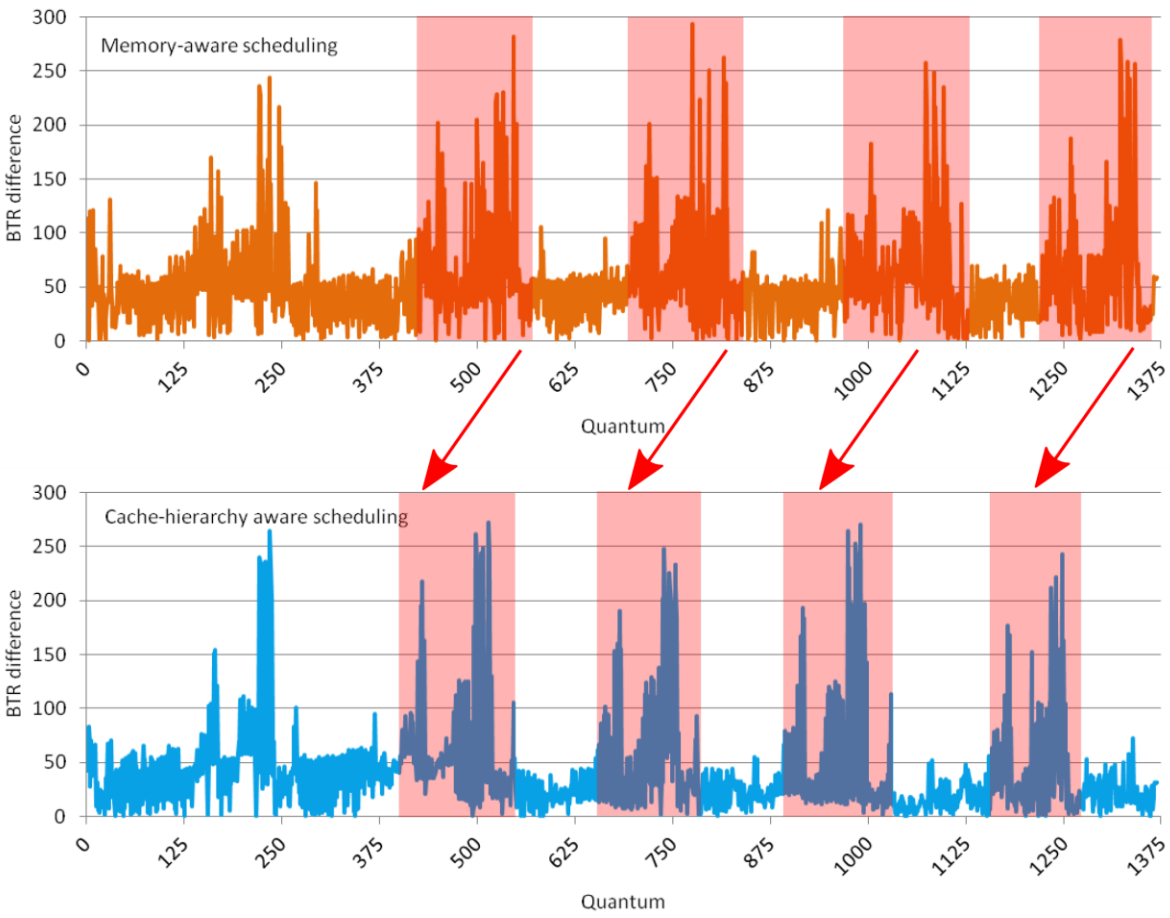
BTR L2 difference evolution



BTR L2 difference evolution time in mix 2

Scheduler performance

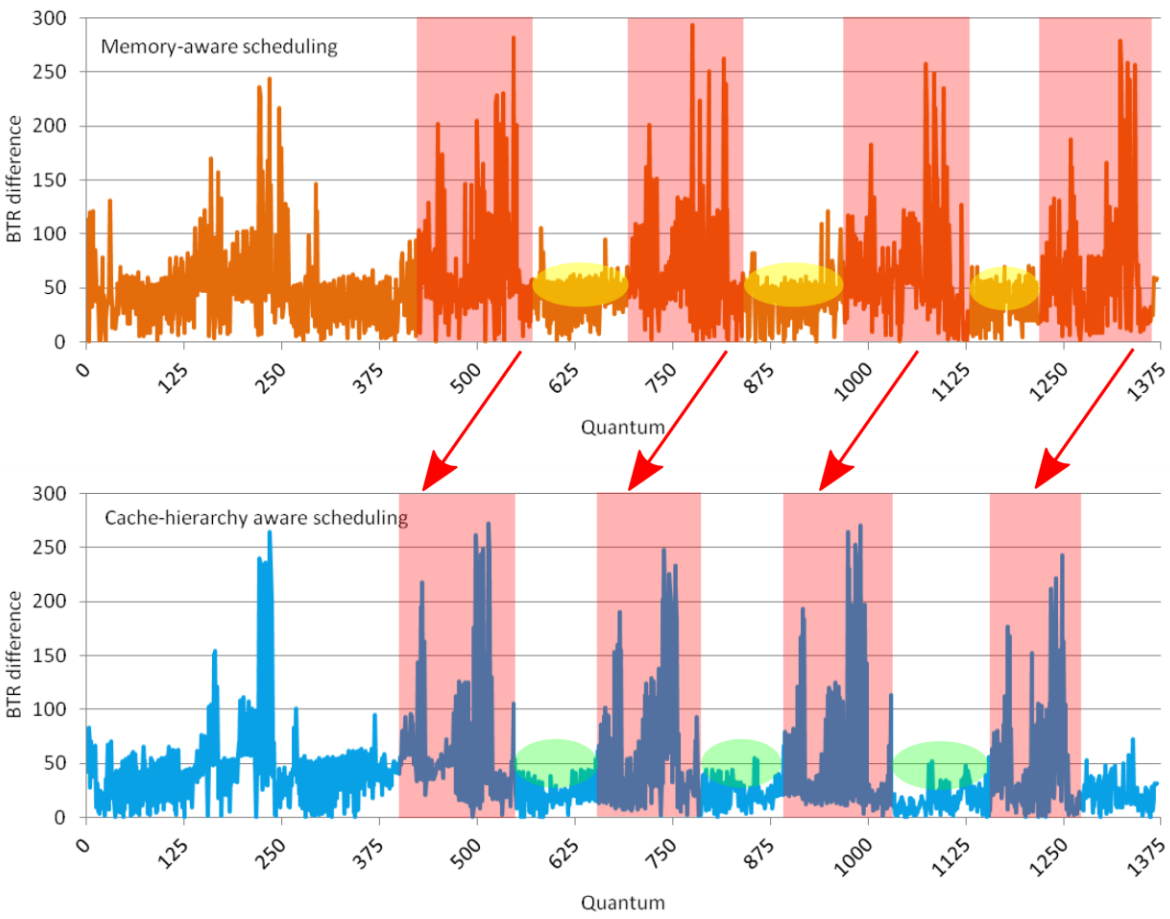
BTR L2 difference evolution



BTR L2 difference evolution time in mix 2

Scheduler performance

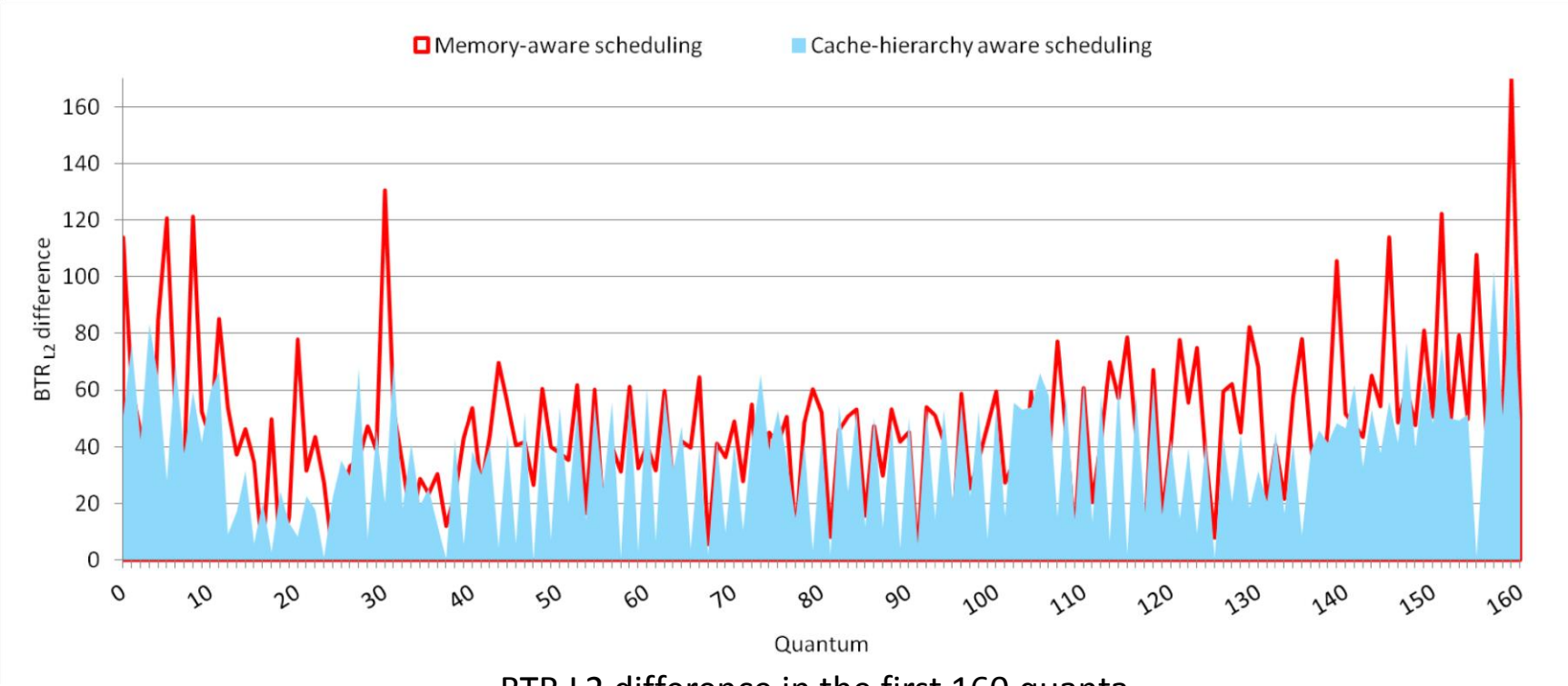
BTR L2 difference evolution



BTR L2 difference evolution time in mix 2

Scheduler performance

BTR balancing on mix 2



BTR L2 difference in the first 160 quanta

Conclusions

- Performance can drop due to bandwidth contention located at different levels of the memory hierarchy.
- The current processor industry trend increases the number of contentions points.
- Memory aware bandwidth jobs only attack main memory contention point.
- Cache-hierarchy bandwidth aware policy:
  - Attacks all the contention points of the cache hierarchy.
  - Increases the performance of the evaluated mixes 30% respect to the memory bandwidth aware scheduling.

- Thank you very much for your attention

- Questions?

Understanding Cache Hierarchy Contention in CMPs to Improve Job Scheduling

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

Universitat Politècnica de València
Spain

Evaluation methodology

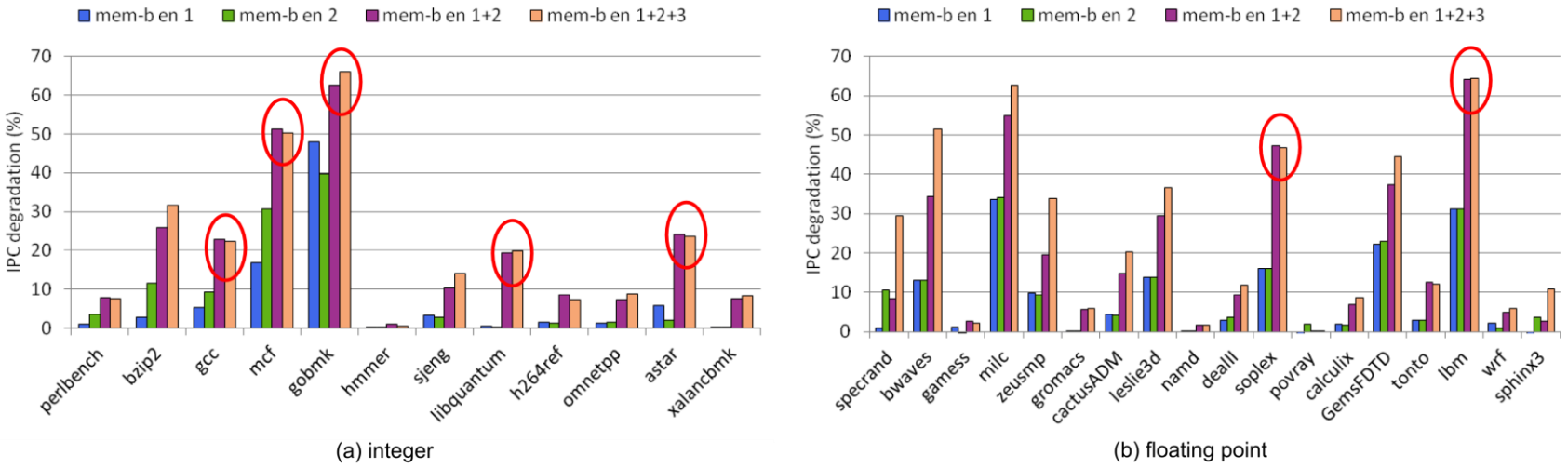
- To deal with the different execution time of the benchmarks, a benchmark execution is set to the number of instructions required to achieve a execution time of 120 seconds in stand alone execution.
- Otherwise, a long job first policy would provide the best performance in most mixes.
- The number of complete executions and instructions of the last execution is measured and recorded offline.
- If the execution time of the benchmarks is larger, the scheduler kills it when the target instructions are executed. If it is lower, the scheduler re-execute the benchmarks several times.

Evaluation methodology

- To evaluate the performance, a set of 10 mixes with eight benchmarks was designed.
- Mixes present an ideal bandwidth (IABW) falling in between 20 and 40 trans/usec.
  - Lower IABWs detract the necessity of a memory-aware scheduler since contention is low.
  - Higher IABWs cannot take advantage of memory-aware scheduling since all the scheduling possibilities reach high contention.

Performance degradation analysis

Degradation due to memory contention (II)



IPC degradation due to memory contention varying the number of co-runners

- Some benchmarks suffer higher IPC degradation when the co-runner runs in the other bi-core since memory is more frequently accessed.
- Other benchmarks suffer higher IPC degradation when the co-runner runs in the same bi-core. This can be caused by L2 cache conflicts or L2 bandwidth.
- In the common case, both degradations are similar.
- The IPC degradation difference is lower from 1 to 2 co-runners than from 2 to 3 co-runners, since 2 co-runners are close to saturate the bandwidth.