

Shared Data Cache Conflicts Reduction for WCET Computation in Multi-Core Architectures.

Benjamin Lesage, Damien Hardy & Isabelle Puaut

5th November 2010

Context and Issues



- WCET computation:
 - Compute tight and safe bounds of tasks' WCET.
 - Must take hardware into account.

- Caches:
 - Help improving average case execution time.
 - Dynamic behaviour, access history dependent.

- Multi-core architectures:
 - Multiple tasks executing at the same time.
 - Share hardware resources, such as caches.

Context and Issues

Static cache analysis



- WCET contribution of memory references w.r.t. caches
 - Requires an estimate of cache contents at each point of a program.
- Sources of indeterminism in the context of cache analysis:
 - Path indeterminism:
 - Gather information for all possible incoming paths.
 - Access indeterminism:
 - Specific to data caches.
 - The precise target of a reference may not be available statically.
 - Different accesses may have the same target in memory.
 - The same access may have different target according to the context.
 - Cache hierarchies:
 - Estimate accessed cache levels upon an access.

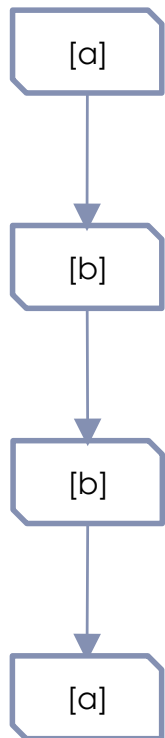


Context and Issues

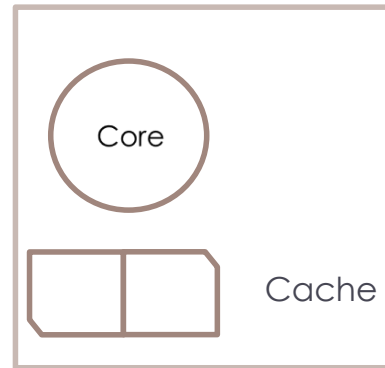
Multi-cores and shared caches - Example



Analysed task



Assumed architecture



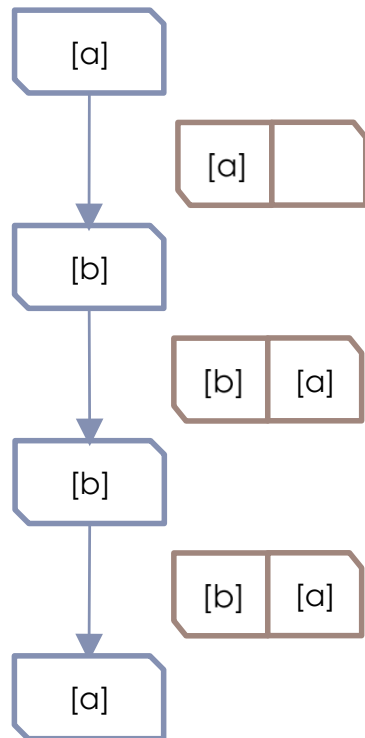
- Is the second access to memory block [a] a hit in the cache hierarchy ?

Context and Issues

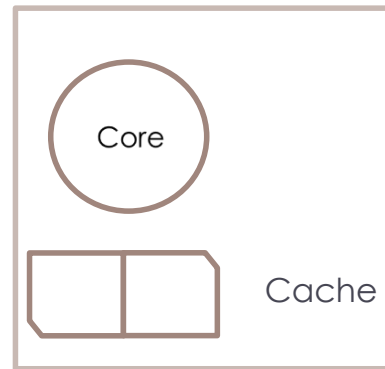
Multi-cores and shared caches - Example



Analysed task



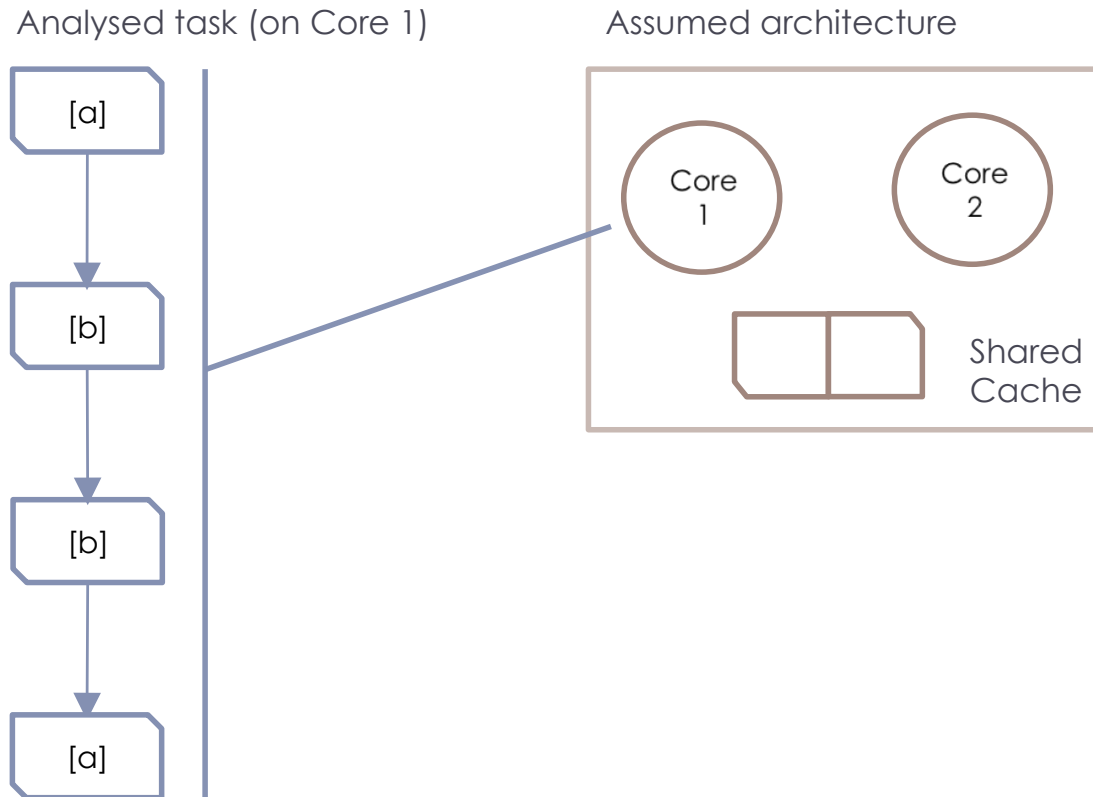
Assumed architecture



- Is the second access to memory block [a] a hit in the cache hierarchy ?
 - Yes.

Context and Issues

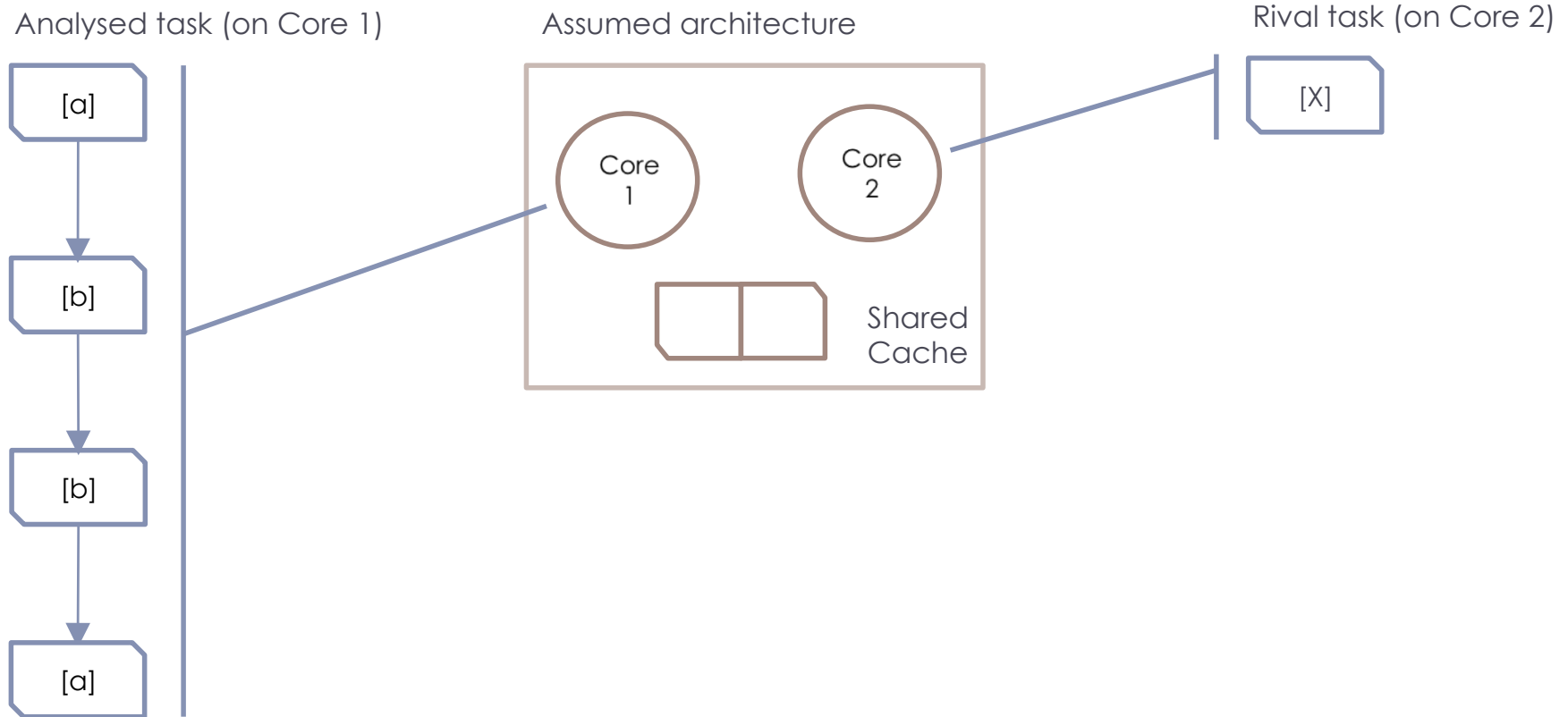
Multi-cores and shared caches - Example



- Is the second access to memory block [a] a hit in the cache hierarchy ?

Context and Issues

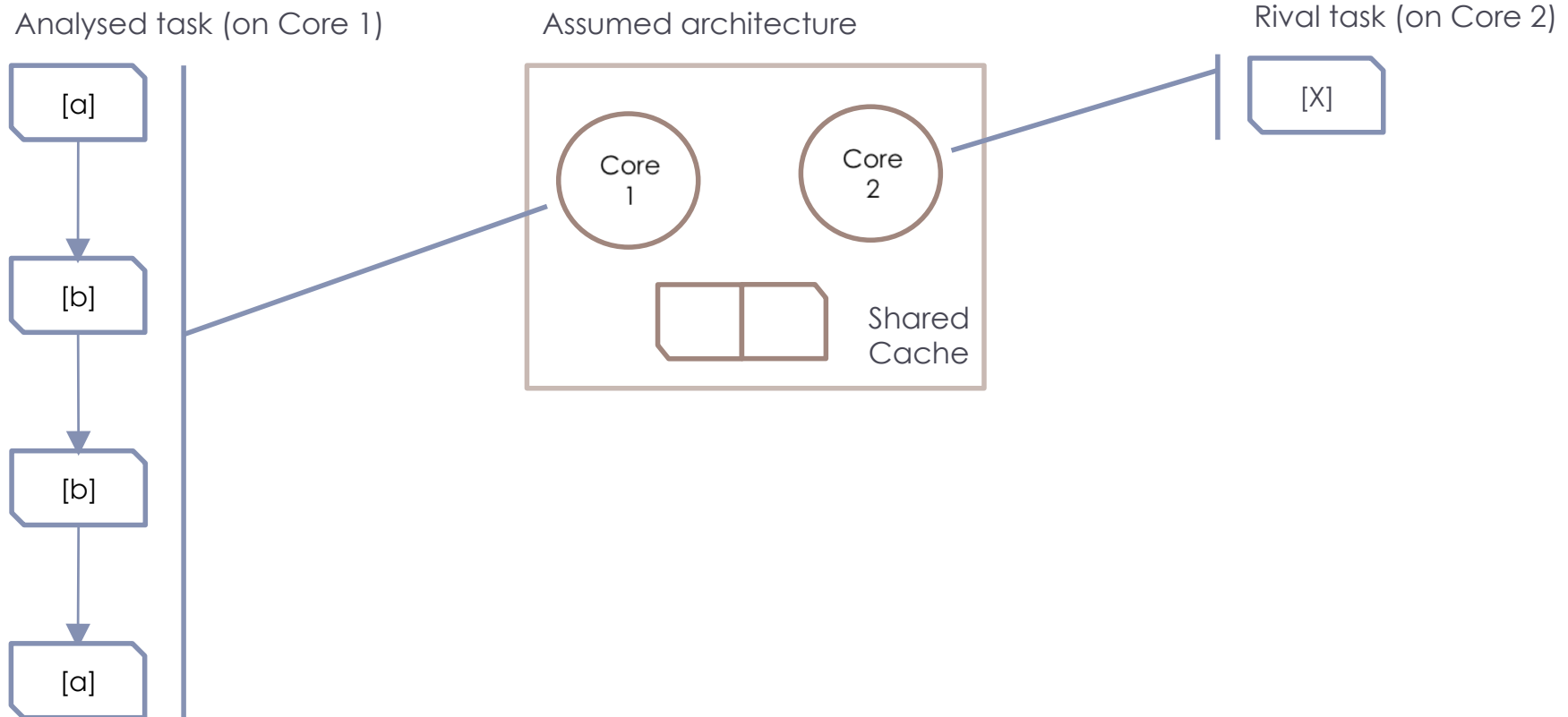
Multi-cores and shared caches - Example



- Is the second access to memory block [a] a hit in the cache hierarchy ?

Context and Issues

Multi-cores and shared caches - Example



- Is the second access to memory block [a] a hit in the cache hierarchy ?
 - Not necessarily, depends on when the rival task accesses the shared cache.



This presentation

- WCET of task while considering its rivals on other cores.
 - Safely consider rival tasks impact on shared data caches.
 - Used by scheduling analyses.
- Reduce tasks' pressure on shared data caches using bypass.

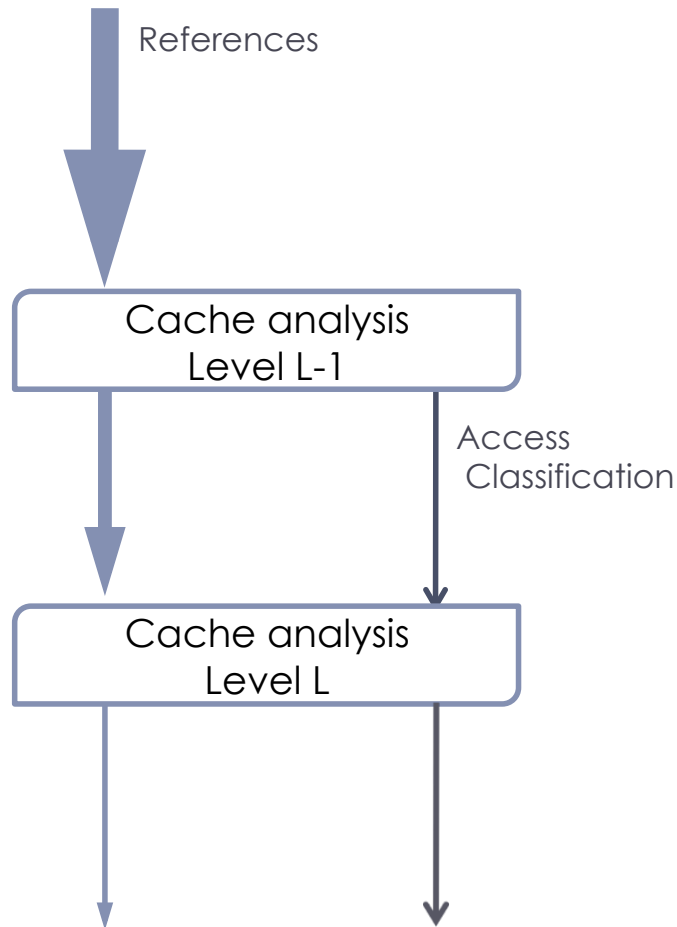
Outline

- Context and Issues
- Method – Data cache analysis
 - Overview
 - Private caches
 - Shared caches
- Bypass heuristics
- Results
- Conclusion and Future works



Method

Method overview – Multi-level Data cache analysis



- Hierarchies enforcing neither exclusion nor inclusion:
 - LRU replacement policy.
 - Set-associative caches.
 - Details in the paper.
- Static analysis of data caches, one by one
 - From the top to the bottom.
- Compute accesses occurrences on cache level L based on:
 - Occurrences on cache level L-1,
 - Access classifications on cache level L-1,
 - See [Hardy & Puaut – RTSS 2008].

Cache analysis



- Estimate safely cache contents.
 - All paths must be considered altogether.
- Based on abstract interpretation [Ferdinand & al. – 2000], 3 analyses:
 - Must: memory blocks always present in the cache.
 - May: memory blocks that may be present in the cache.
 - Persistence: memory blocks which once loaded will not be evicted from the cache.

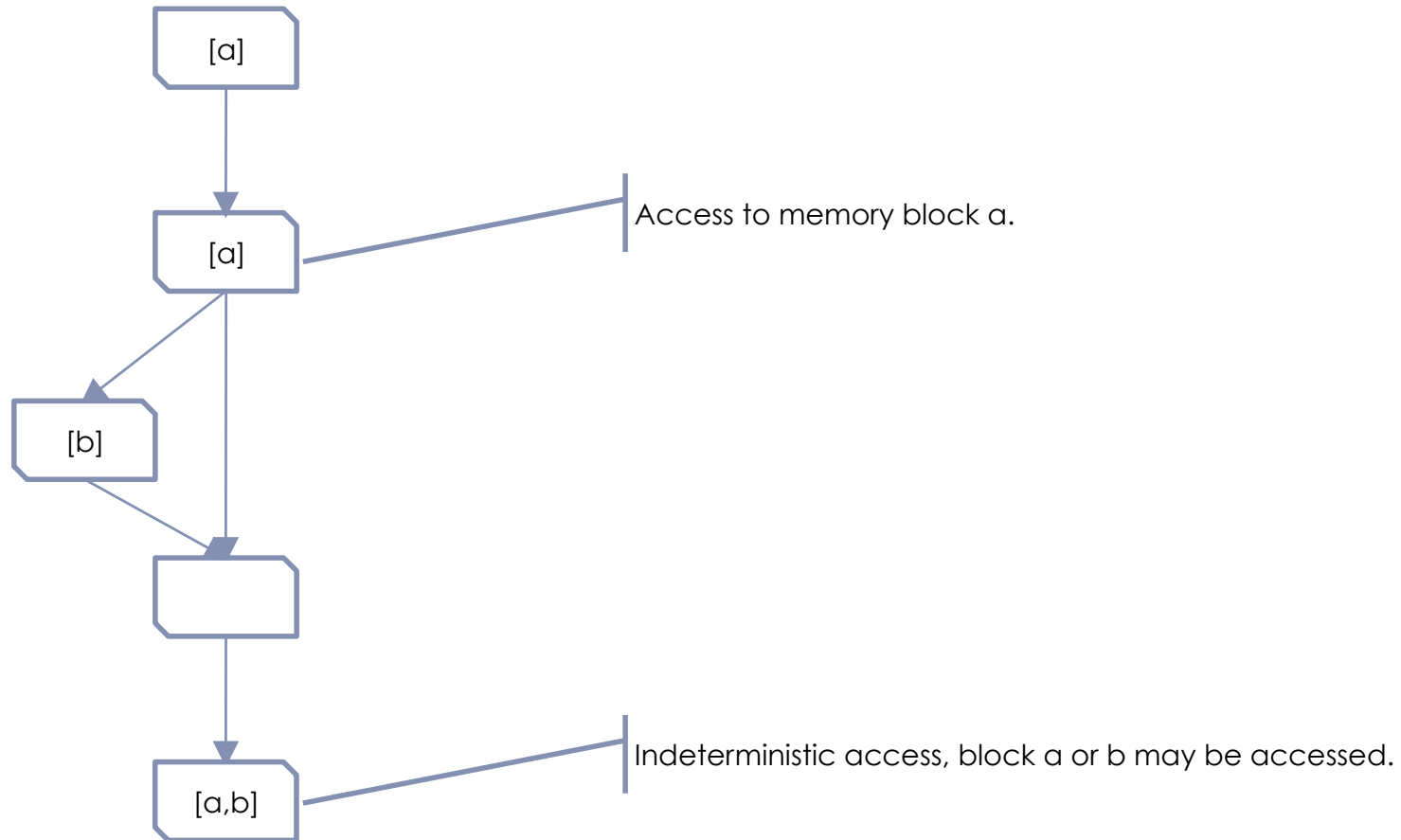
Cache Hit/Miss Classification based on abstract cache contents
WCET contribution of references w.r.t. caches

Cache analysis

Example – Must analysis




- Considered Data cache:

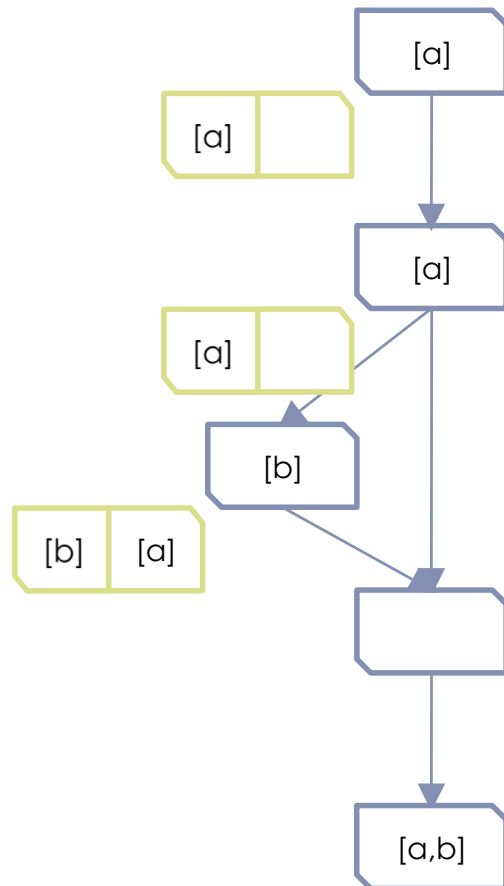


Cache analysis

Example – Must analysis

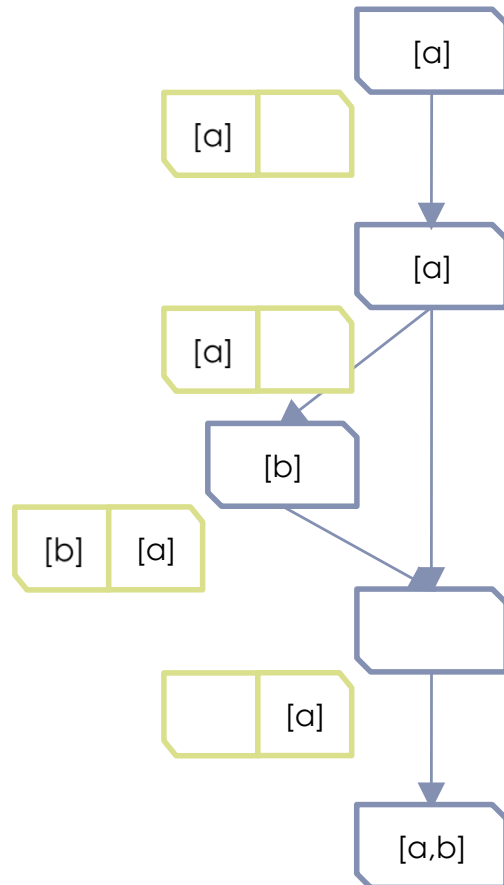



- Considered Data cache: 
- Update the cache upon an access.



Cache analysis

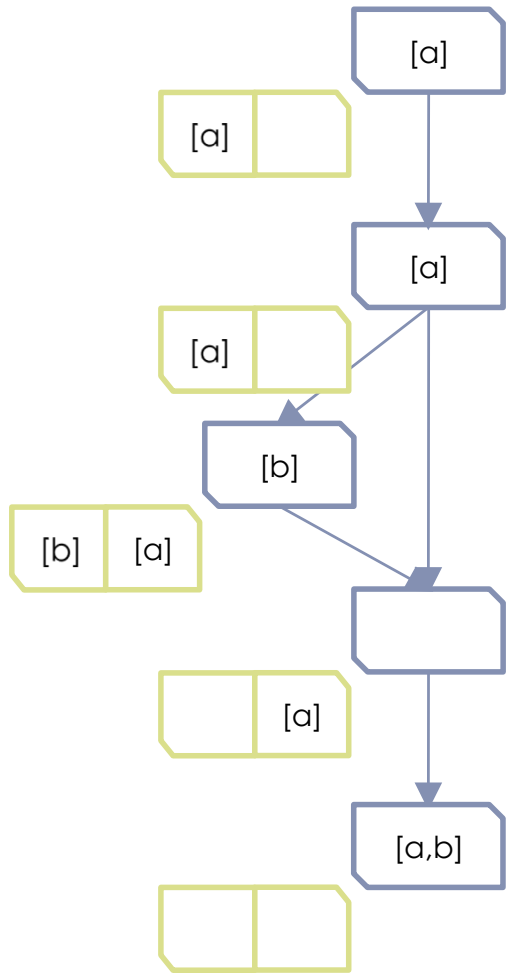
Example – Must analysis



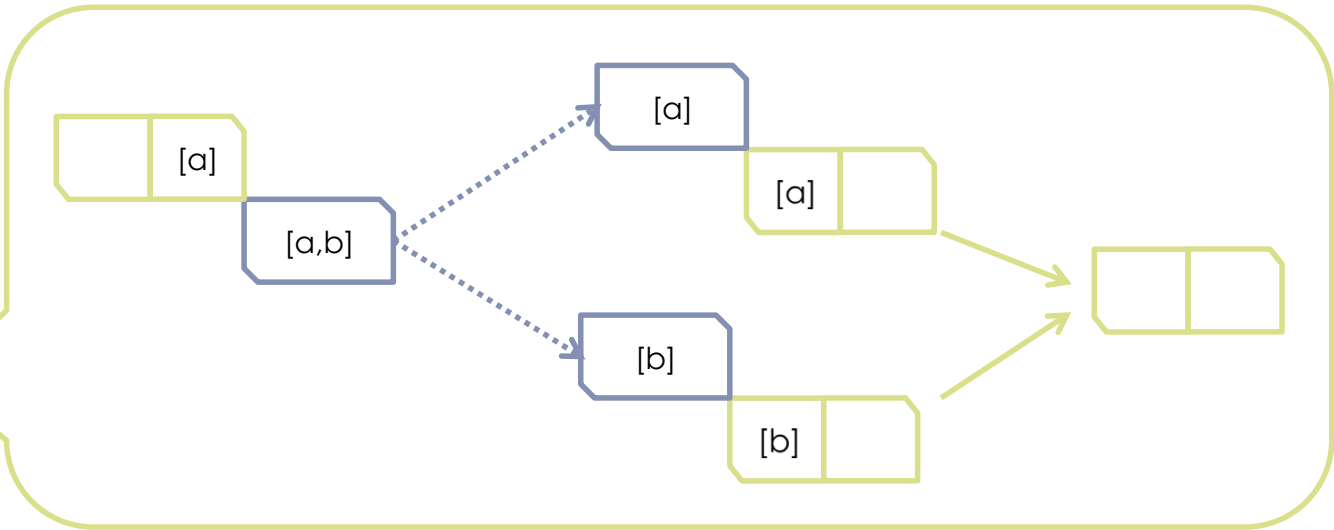
- Considered Data cache: 
- Update the cache upon an access.
- Join incoming states on branch convergence.
 - Must: Intersection + Maximal Age

Cache analysis

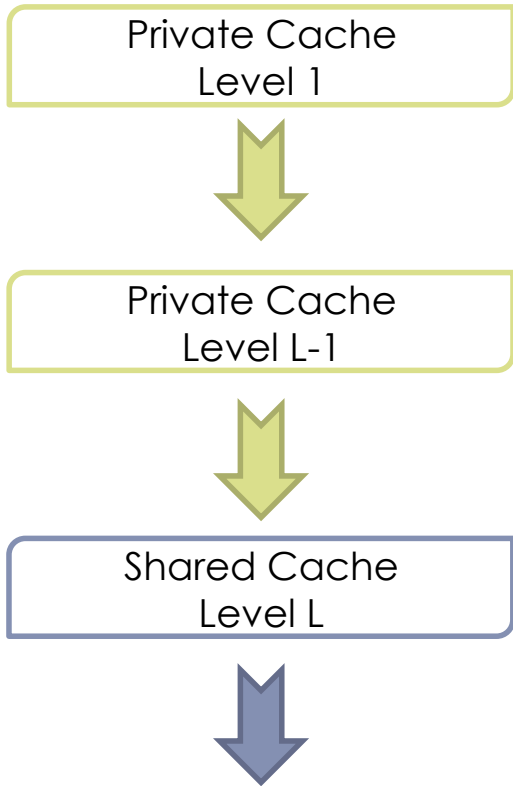
Example – Must analysis



- Considered Data cache:
- Update the cache upon an access.
- Join incoming states on branch convergence.
 - Must: Intersection + Maximal Age
- Indeterministic accesses:
 - Consider all possibilities, combine join and update.



Cache analysis



- Private caches hierarchy: analysed.

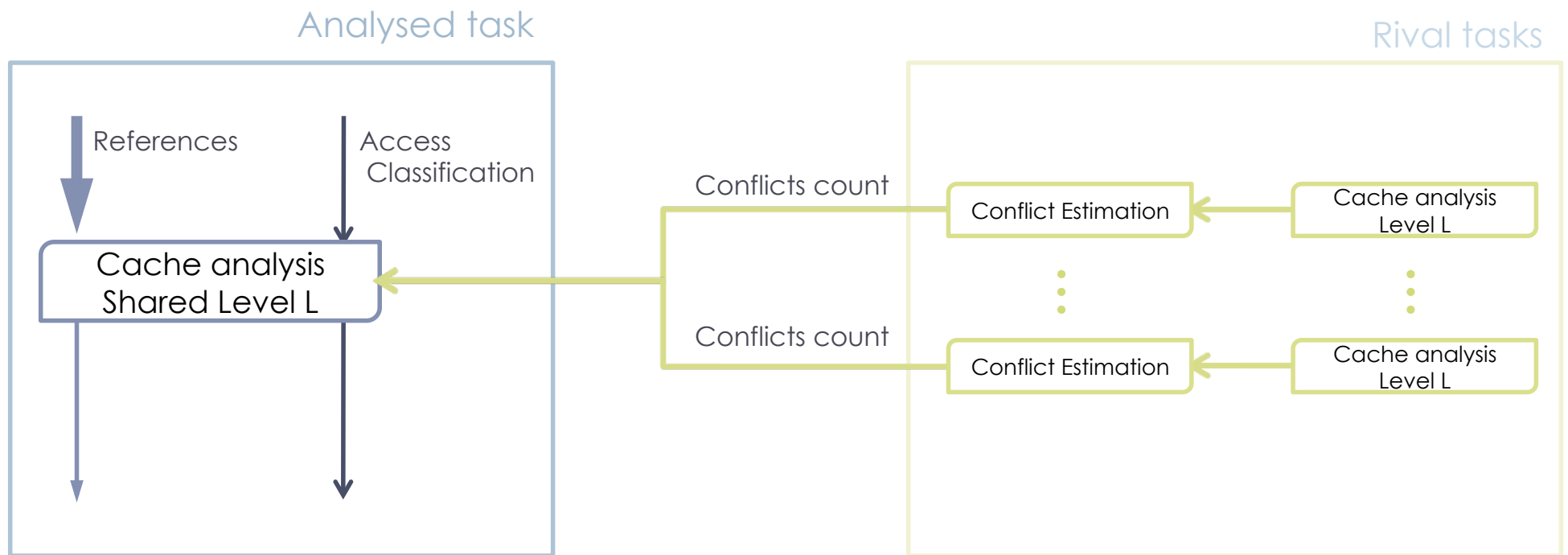
- Still need to deal with shared caches.

Shared cache analysis

Overview

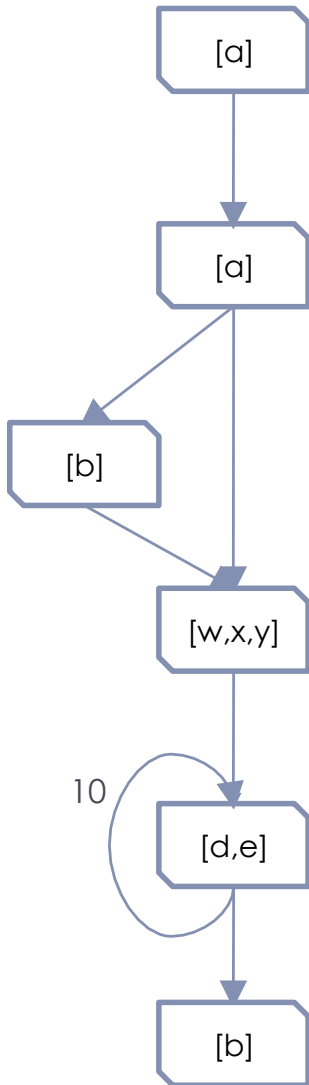


- Conflicts stemming from rival tasks are estimated.
- Conflicts estimation used during shared cache level analysis.
 - Produced Cache Hit/Miss Classifications takes possible conflicts into account.



Shared cache analysis

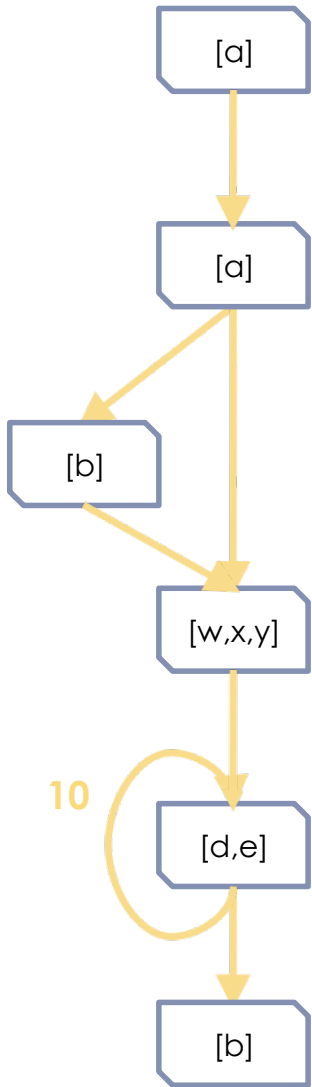
Conflict estimation



- Any task, any time may alter shared caches
 - Compute all interleavings: too costly in practice.

Shared cache analysis

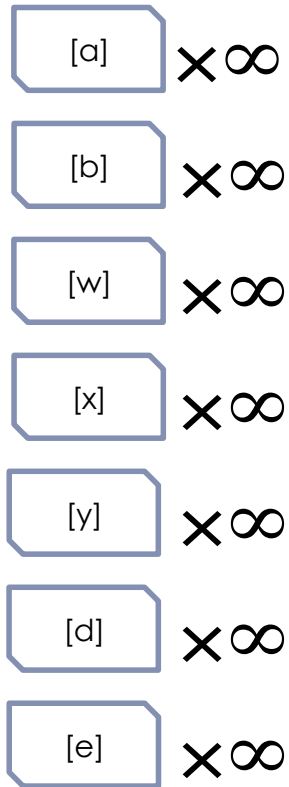
Conflict estimation



- Any task, any time may alter shared caches
 - Compute all interleavings: too costly in practice.
- To reduce this cost, we abstract from a task:
 - Access ordering
 - Occurrence count

Shared cache analysis

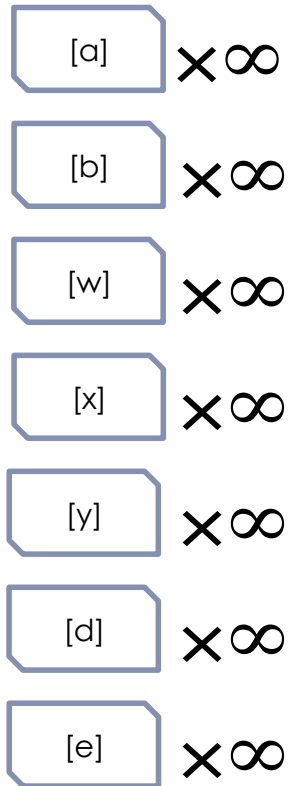
Conflict estimation



- Any task, any time may alter shared caches
 - Compute all interleavings: too costly in practice.
- To reduce this cost, we abstract from a task:
 - Access ordering
 - Occurrence count

Shared cache analysis

Conflict estimation



- Any task, any time may alter shared caches
 - Compute all interleavings: too costly in practice.
- To reduce this cost, we abstract from a task:
 - Access ordering
 - Occurrence count
 - Each block stored by a task on a shared cache level may trigger a conflict.
- For each cache set
 - Find memory blocks used by the task.
 - Count the total number of blocks.

Cache Block Conflict Number (CCN)
Conflicts generated by rival tasks for each cache set

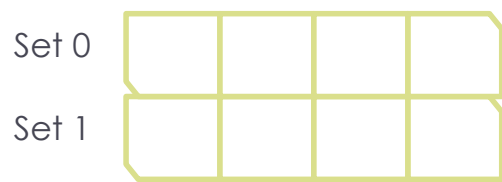
Shared cache analysis

Conflict Integration

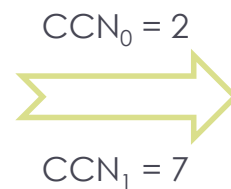


- A cache analysis is performed using modified cache states:
 - CCN cache blocks may have been allocated to rival tasks in the shared cache.
 - Only $(\text{Cache L Associativity} - \text{CCN})$ ways are for sure available.

Example:



Base cache configuration



Available cache space during analysis



Cache line occupied by a conflicting block

Shared cache analysis



- Conflict Estimation is pessimistic:
 - Many blocks may clutter up the cache.
 - Every block a task may store on a shared cache level is included.
- Little cache space detected as available during shared cache analyses.

Need to control the impact of tasks on shared caches.

Bypass

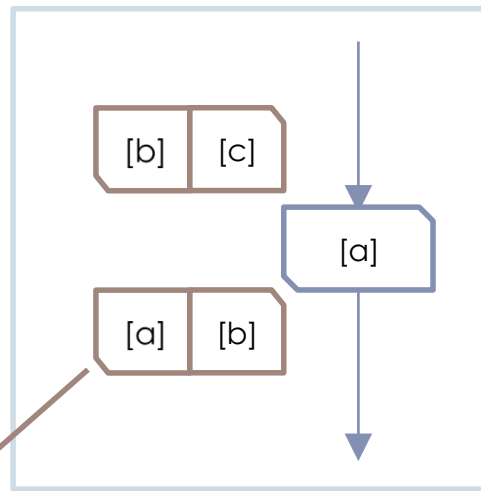
Mechanism overview



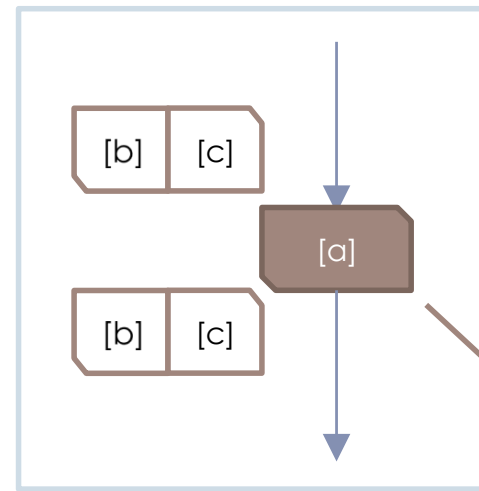
If an instruction bypasses a cache level, it does not alter this cache level:

Example:

Without bypass



Using bypass



Considered cache

Access bypassing the cache

Bypass



- Bypass is used for cache usage control.
 - Hardware support.
 - Software-taken decision, offline.
- If instruction i bypasses cache level L :
 - Instruction i does not alter cache level L in any way.
 - Instruction i does not contribute to conflicts on cache level L .

Find a way to select instructions bypassing shared cache levels.

Bypass

Heuristics – Reuse Information based



- Some data is not detected as reused before its eviction from the cache:
 - Inserting such data in the cache may not be useful.
- **Reuse Bypass** (RB) aims at preventing such insertions.
 - Uses cache classifications from a prior cache analysis.

Bypass

Heuristics – Accessed structures based



- Indeterministic accesses jeopardize the precision of data cache analyses:
 - No block can safely be inserted in the cache during analyses.
 - All accessed blocks have to be present for a hit classification.
- **All Indeterministic Bypass (AIB)** targets indeterministic accesses.
 - Based on the results of the address analysis.

Bypass

Heuristics – Accessed structures based



- Indeterministic accesses jeopardize the precision of data cache analyses.
 - Large structures lead to an important shared cache usage from a task.
- **Reduced Cache Usage Bypass** (X-RCUB) allows for control on the maximum cache space occupied by a task.
 - Accesses with the largest accessed range are bypassed first.

Results

Experimental setup & Benchmarks



- 2 levels cache hierarchy:
 - Private L1: 4-way 1 KB data cache, 32B line size, 1 cycle access latency.
 - Shared L2: 8-way 4KB data cache, 32B line size, 10 cycles access latency.
- Memory: 100 cycles access latency.
 - 150 cycles store latency.
 - Perfect instruction cache.
- Metrics:
 - $DMCU_{L2}$: number of memory blocks stored by a task in the L2 cache.
 - PHR_{L2} : L2 cache hit ratio along the predicted worst-case execution path.

Results

Experimental setup & Benchmarks



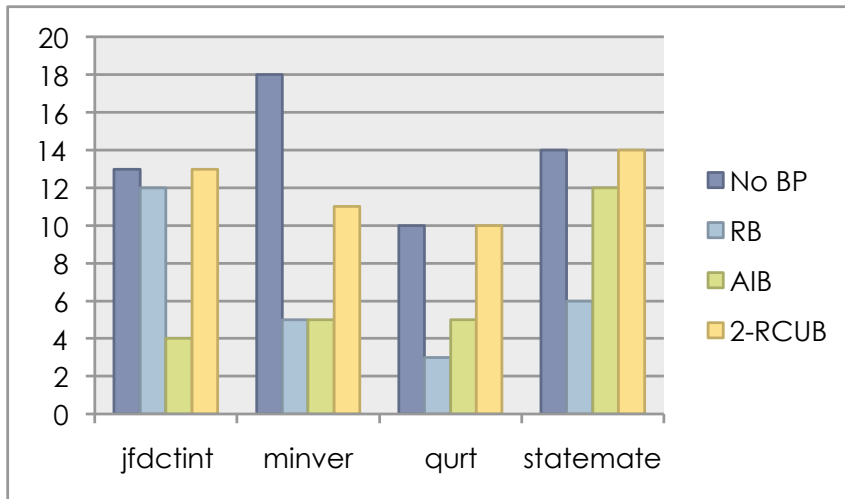
- Analysed task sets:
 - One task per core.
 - Conflicts stems from sharing.
 - Same bypass heuristic used for all benchmarks at once.
- Light task set:
 - Tasks nearly fit altogether in the cache.
- Heavy task set:
 - Each task generates more conflicts than the shared cache may hold.

Results

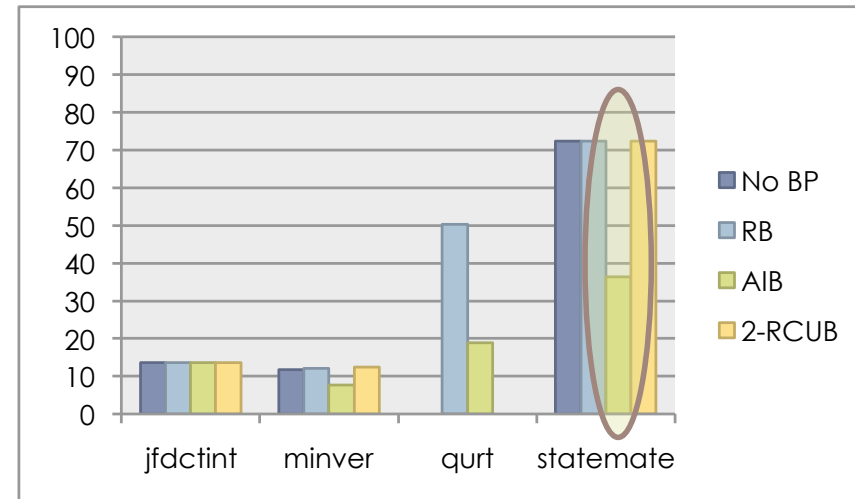
Light task set



DMCU_{L2} in cache blocks



PHR_{L2} (%)



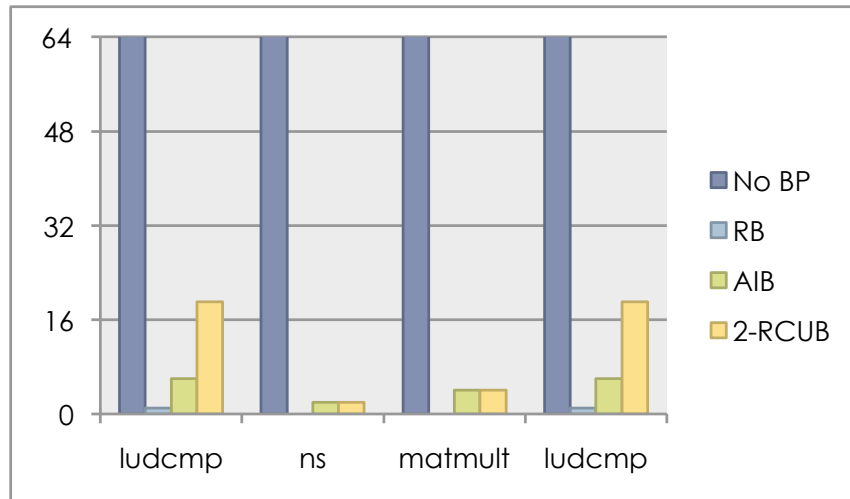
- Small pressure on the cache, plainly considering conflicts seems reasonable.
- Bypass reduces this pressure, leading to even better results.
 - But may have negative impact on tasks PHR_{L2}.

Results

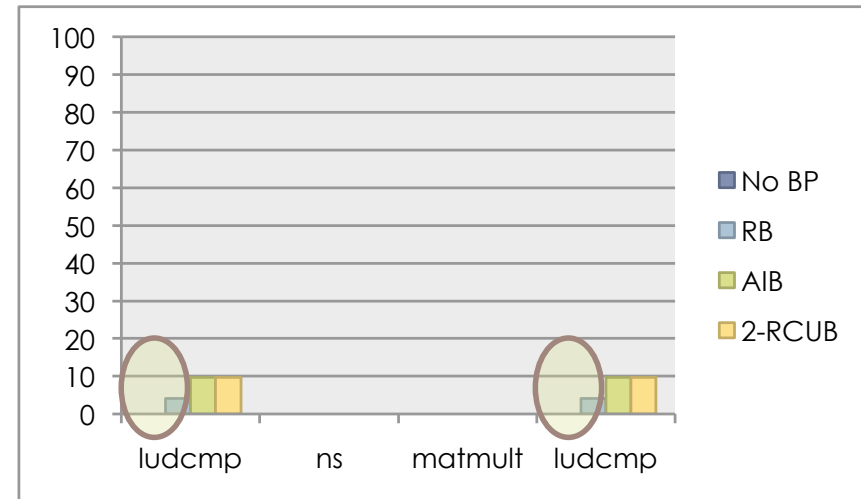
Heavy task set



DMCU_{L2} in cache blocks



PHR_{L2} (%)



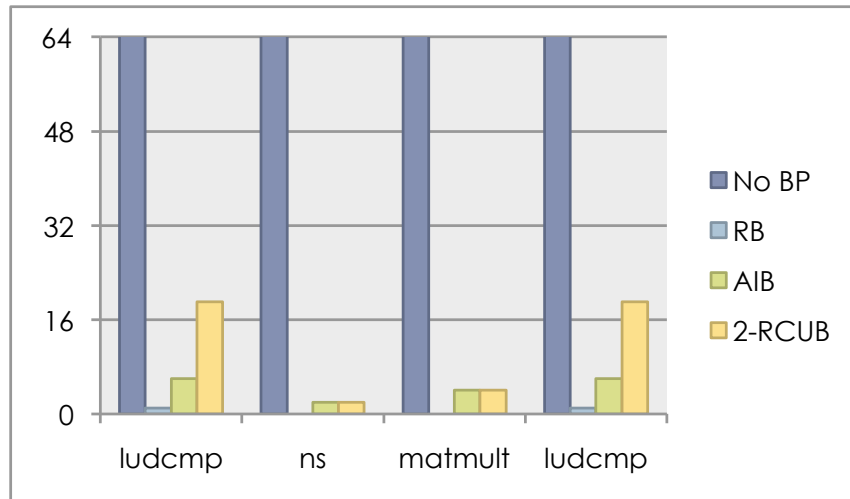
- Too many conflicts, no reuse captured on the shared cache without bypass.
 - Note that only ludcmp benefited from the L2 cache, according to analyses.

Results

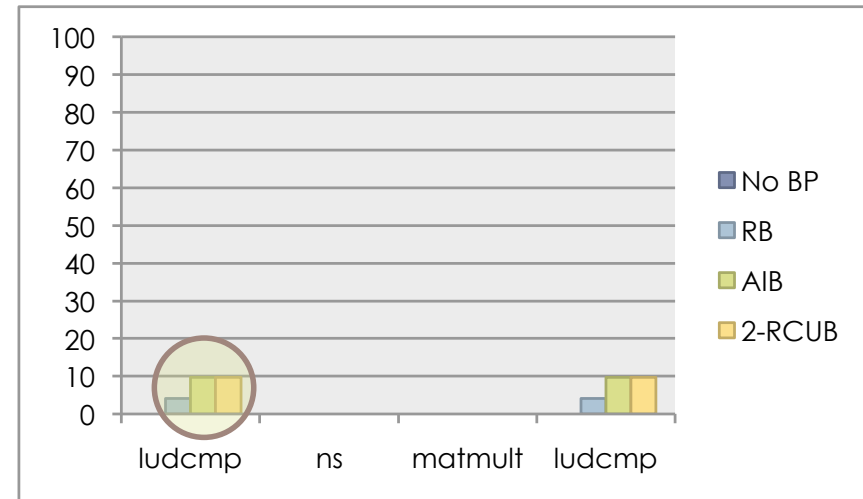
Heavy task set



DMCU_{L2} in cache blocks



PHR_{L2} (%)



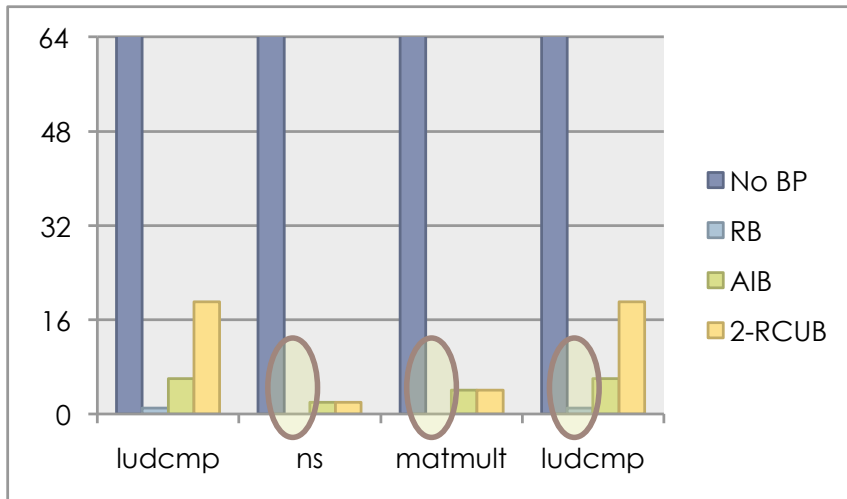
- AIB and 2-RCUB allows for the best predicted hit ratios:
 - They impact both inter-task and, in the case of ludcmp, intra-task conflicts.

Results

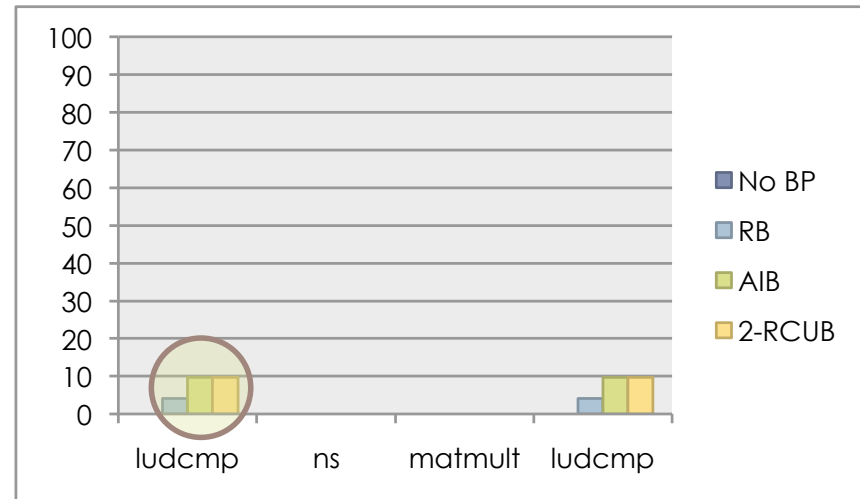
Heavy task set



DMCU_{L2} in cache blocks



PHR_{L2} (%)



- RB has the strongest impact on conflicts reduction.
 - But does not provide the best predicted hit ratios.
 - RB is a greedy heuristics:
 - The cache classifications it relies on are not computed again once a bypass decision has been taken.

Conclusion & Future works



- Plain conflicts estimation and consideration is not scalable.
 - The more information about rival tasks kept, the more costly the analysis.
 - A mechanism is required to reduce their number or occurrences.
- Bypass is an interesting solution to control a task's cache usage.
 - Clever selection of bypassed instructions depends on the goal.
- Combining conflicts precluding, reduction and estimation methods should help.
 - Aim at reducing hot spots in the cache.
 - Combinations of locking, partitioning, bypass, synchronisation, etc.
 - [Suhendra & al. – 2008 DAC]
- Including tasks' preemptions and migrations is also to be done.



Thank you for your attention.

Questions ?