www.bsc.es

Assessment of the Implementation of the EFL Time-Randomised Cache in the NGMP Architecture

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Cobham Gaisler AB

Francisco J. Cazorla (TC),
Carles Hernandez,
Jaume Abella,
Leonidas Kosmidis

Jan Andersson,
Nils-Johan Wessman

Marcel Verhoef (TO)

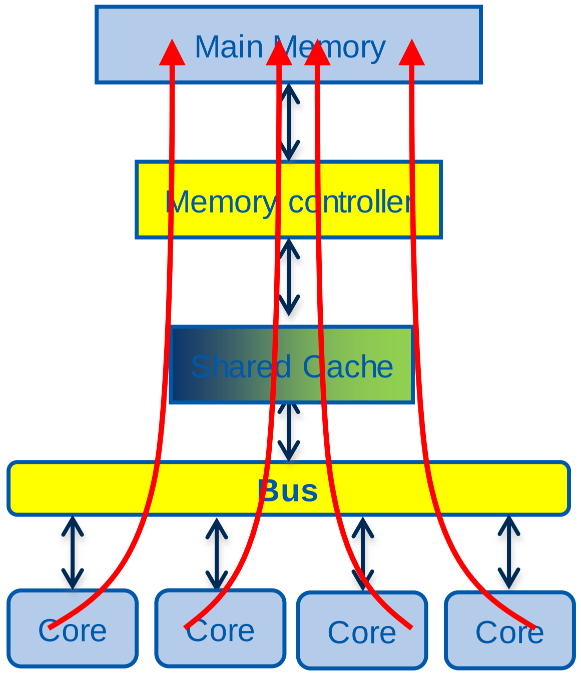
Software Systems Division & Data Systems Division Final Presentation Days
09/05/2017

EFL Time-Randomised Cache in the NGMP architecture

- Contract No: 4000116120/15/NL/FE/as
- Contractor: Barcelona Supercomputing Center (ES) and Cobham Gaisler (SE)
- TRP (65k Euro)
- TRL: 3 / 4
- Duration: 12 months (KO: Feb 2016, FR: May 2017)
- TO: M. Verhoef

Agenda

- Motivation
  - Pros and Cons of LLC partitioning
- Setting the Scene
  - Probabilistic Timing Analysis
- Idea
  - Probabilistic Control of cache interference
- Implementation
  - From a simulator to RTL
- Results
  - Performance
  - Quality of Service
- Conclusions



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Motivation

Motivation

- More and more functionality is provided by software components in every new generation of CRTES
  - Increase of the required computational power
  - High-performance hardware (HW) features (e.g. **caches**) used

- Shared last-level cache (LLC)
  - Improves average performance → deployed in high-performance processors (e.g. ARM Cortex A9, A15, Freescale P4080,…)
  - Its use in CRTES is not straight forward
    - Interferences in the LLC makes task WCET estimates history dependent
    - WCET estimate for a task depends on its corunners → affects time composability

The issue: Inter-task interferences

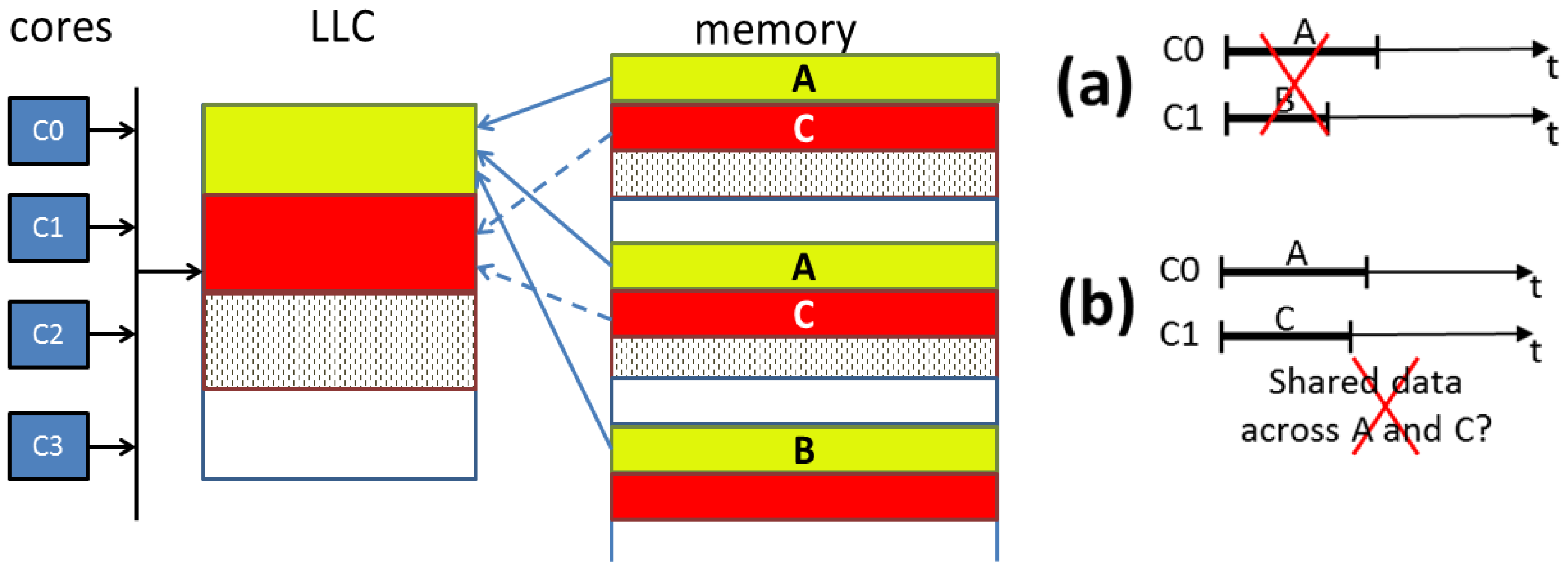
- Two Solutions to handle inter-task interferences

- Combined multi-task WCET analysis
  - Tighter WCET estimates
  - Affects time composability: if any task is changed → redo the analysis for all tasks

- The current solution to enable LLC in multicore real-time processors is based on cache partitioning (SW and HW)
  - Prevents inter-task interferences ✓
  - Impacts data sharing among tasks and task scheduling ✗

Example: software cache partitioning

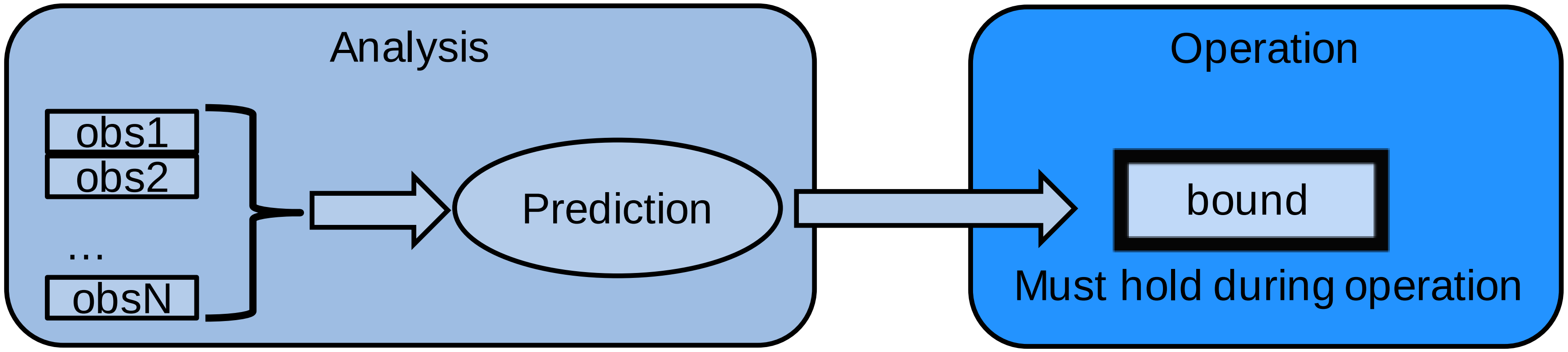
- Let assume tasks ***A*** and ***B*** are mapped on the same cache sets, and task ***C*** is mapped on a different sets
  - (a) The scheduler has to prevent tasks ***A*** and ***B*** to run simultaneously
  - (b) It is non-obvious how to manage read-write shared data among tasks ***A*** and ***C***



Setting the Scene

Measurement-Based Timing Analysis

- Analysis phase
  - Collect measurements to derive a WCET estimate that holds valid during system operation
- Operation phase
  - Actual use of the system (under assumption it stays within its performance profile)

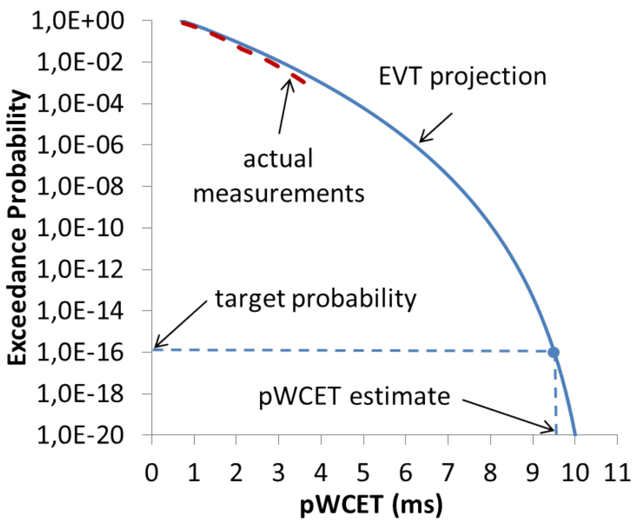


Motivation

- Sources of Jitter (SoJ) challenge WCET derivation
  - Any platform element in the platform that cause execution time of a program to vary is a SoJ
- Measurements have to properly capture the impact SoJ have on execution time to achieve a reliable bound
  - Low-level sources of jitter scape the user controllability
    - What is the placement objects in memory that leads to the WCET?
  - Systems are complex to understand, **the user can only follow what happens at a high level**

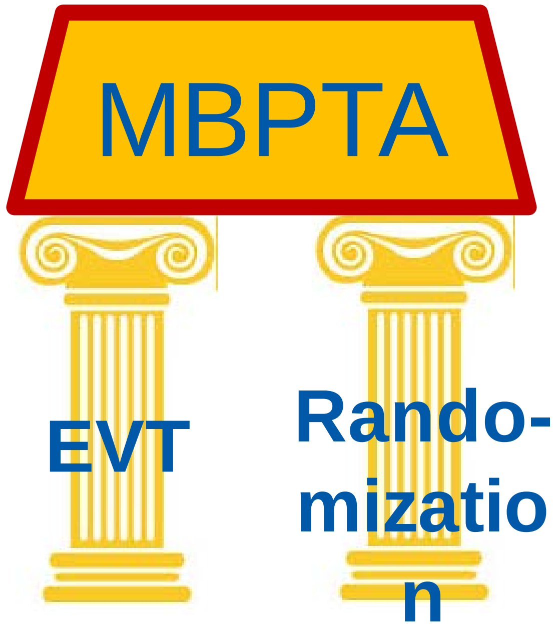
Measurement-Based Probabilistic Timing Analysis

- Control SoJ so their worst operation-time behaviour is captured at analysis time
  - Deterministic Upperbound → make them work on their worst latency
    - A simple run suffices to capture its behaviour
  - Probabilistic Upperbounding → randomize them
    - A probabilistic argument is required on the number of runs needed



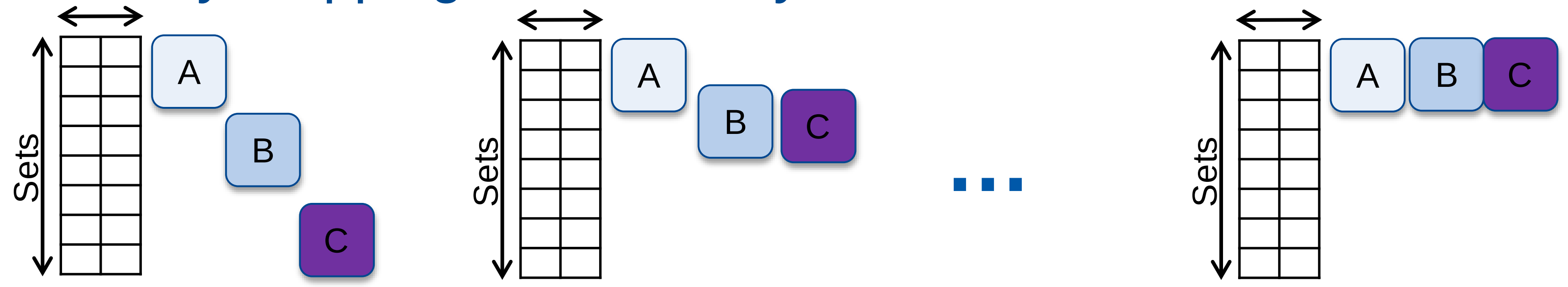
Measurement-based Probabilistic Timing Analysis

- Measurements capture the impact of SoJ
  - The user does not need to control low-level SoJ
    - Upperbounded
    - Randomized
- Probabilistic WCET (pWCET)
  - Quantification of the possibility of exceeding the estimated bound
- How many runs are required?
  - It depends on the platform
  - Sufficient to capture all SoJ
  - Extreme Value Theory (EVT)
    - To achieve arbitrarily low pWCET



Example: The Cache

- Memory mapping → cache layouts → execution time

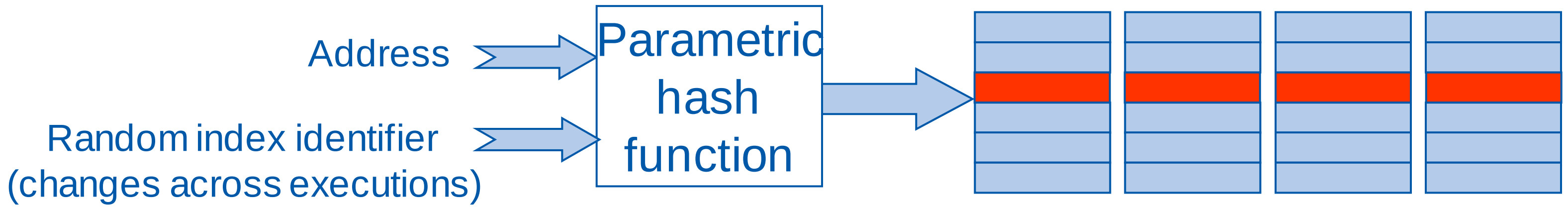


- Deterministic system
  - How does the user get confident that experiments capture bad (worst) mappings?
  - Memory mapping varies across runs, but not in a random manner
- Randomized systems
  - Make N runs
  - We can derive
    - the probability of the observed mappings @ operation
    - the probability of unobserved mappings

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Time-randomised caches

- **Random Replacement**
  - Evict on miss policy, randomly selects a victim line from the target set to be evicted
- **Random Placement across runs**, so executions times are probabilistic

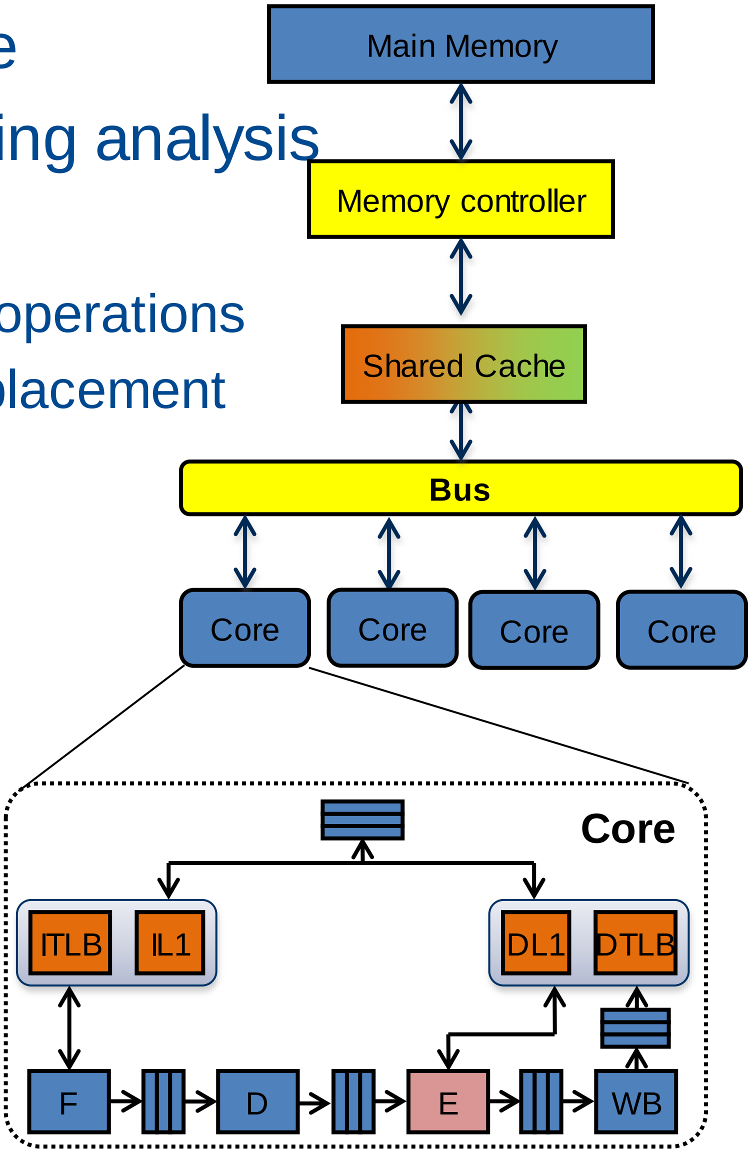


- Random placement and replacement caches break dependence of performance on actual addresses
  - Does not matter *where* (which cache set) data are located

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

A Time-randomized Processor for the Space Domain

- We made a processor resembling the NGMP compatible with probabilistic timing analysis
- Changes were required at:
  - Floating point unit → Fixed latency FPU operations
  - L1 Cache → Random placement and replacement
  - L2 Cache → Random placement and Replacement + partitioning
  - Random arbitration in shared resources

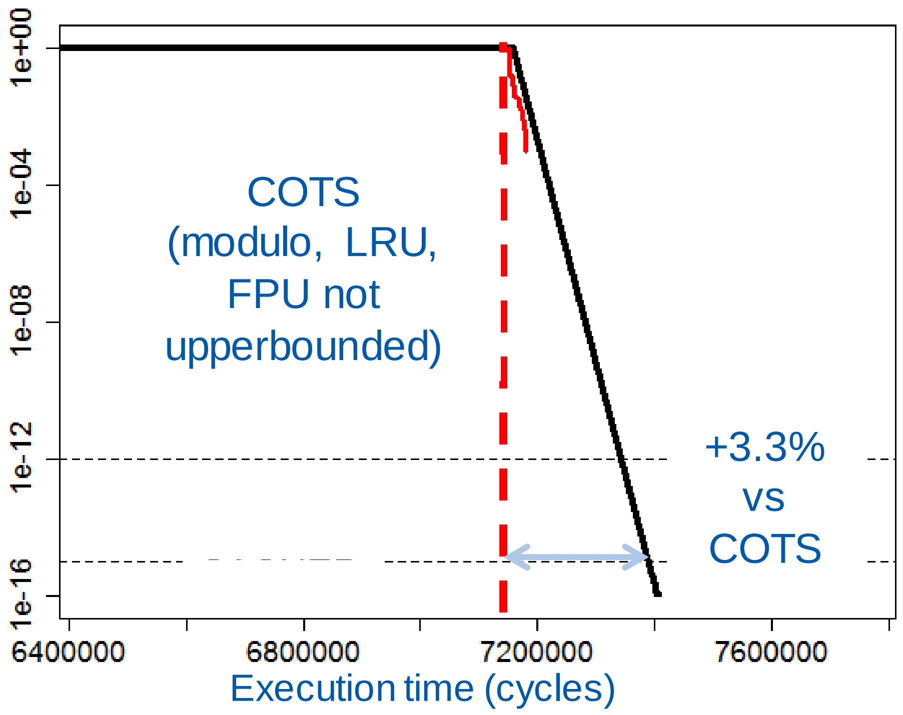


PROXIMA

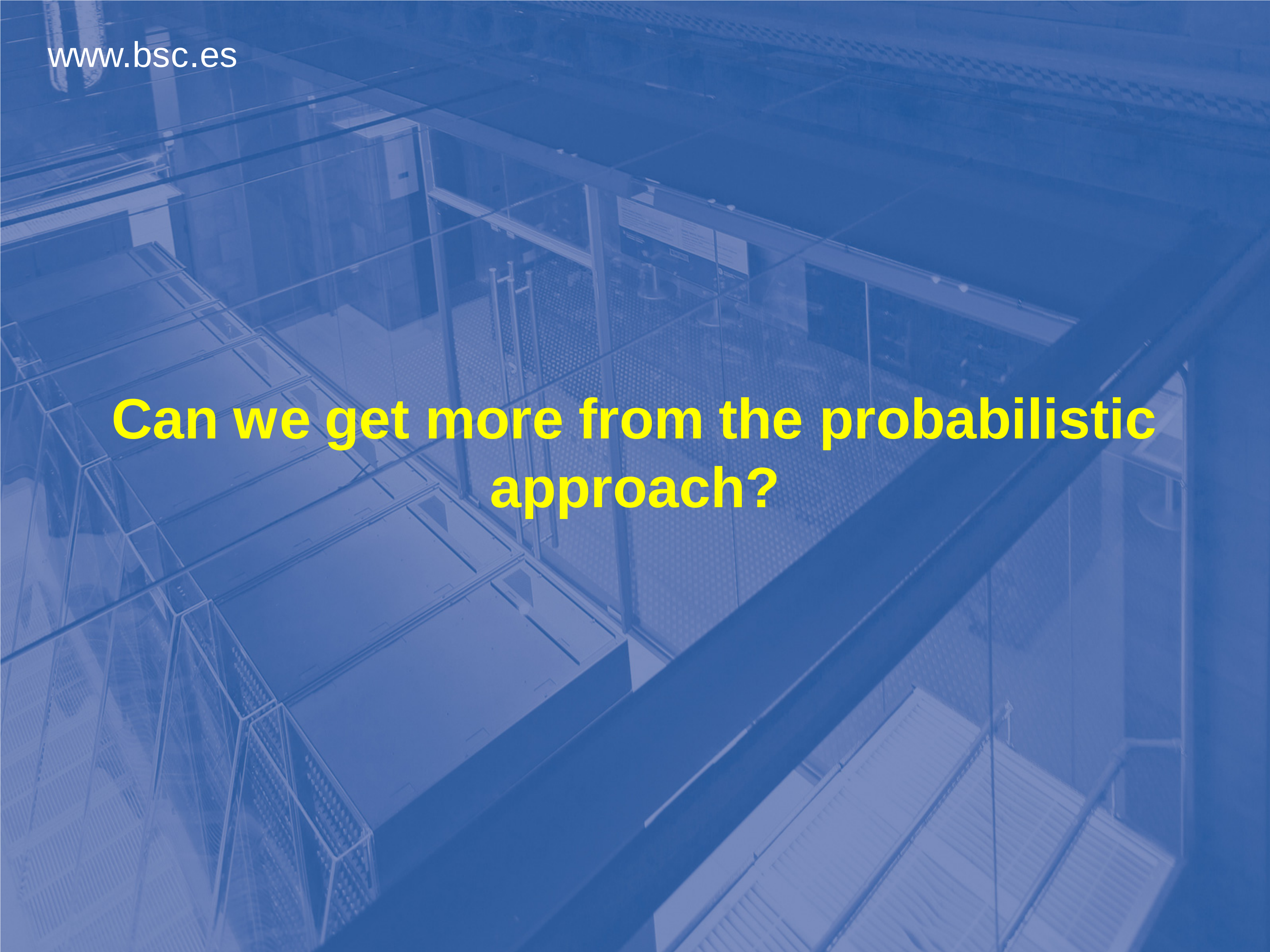
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

A Time-randomized Processor for the Space Domain

- Measurements capture the effect of the existing SoJ
  - Randomization and upperbounding
- Probabilistic WCET can be derived



Barcelona Supercomputing Center
Centro Nacional de Supercomputación



www.bsc.es

Can we get more from the probabilistic approach?

L2 Cache Inter-task Interference

- Given two tasks accessing a shared cache, features that shape their interference are the following:

- Time deterministic cache (Modulo and LRU)
  - Memory mapping of tasks → it determines the sets accessed
  - Access frequencies of each tasks → affects LRU state
  - Relative order of accesses → affects LRU state

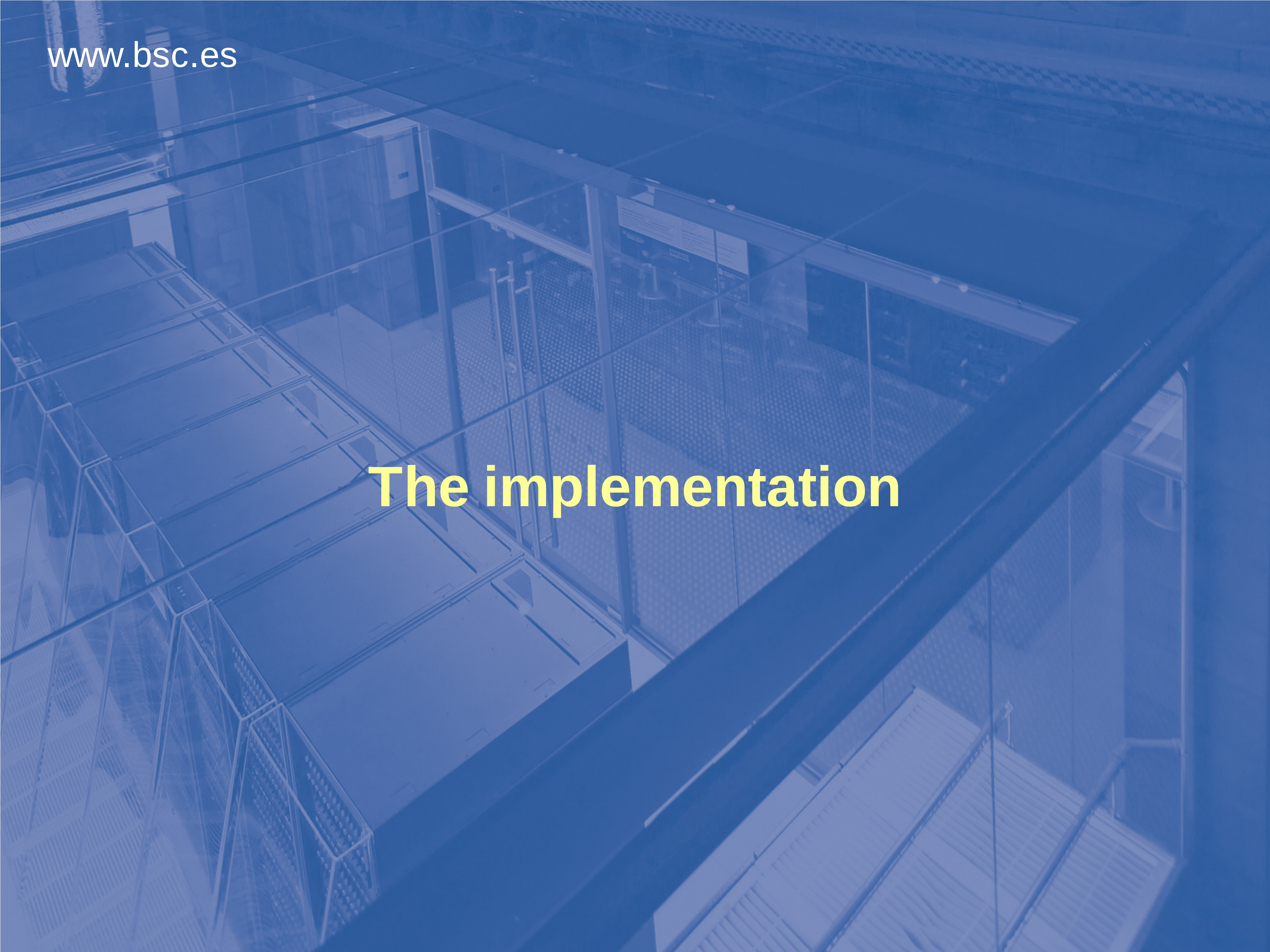
- Time Randomised cache:
  - ~~Memory mapping of tasks~~ ✓
  - **Miss** frequencies of tasks (hits do not affect cache state)
  - Relative order of accesses

Probabilistically upper-bounding interference

- Idea: In Time-Randomised caches **limiting how often a task can evict lines** from LLC is enough to derive trustworthy and tight WCET estimates

- Limiting LLC eviction frequency as a way to control intertask interferences in LLC

- **No need to physically partition the cache** ✓

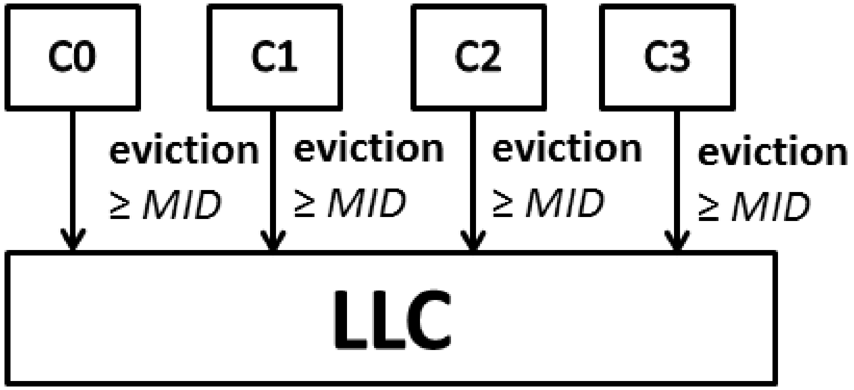


www.bsc.es

The implementation

Eviction Frequency Limitation (EFL)

- Each task running in the multicore is allowed to generate a new eviction in the LLC as long as at least MID cycles elapsed since its last eviction

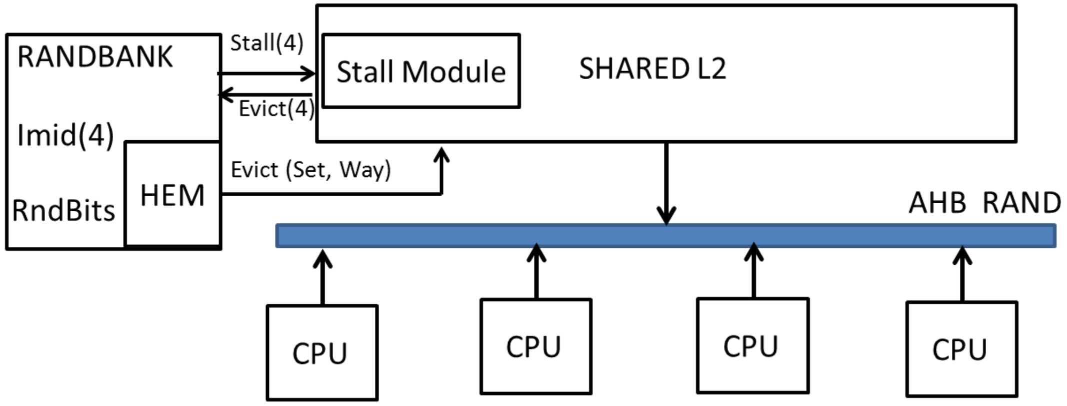


- Controlling the maximum eviction frequency is preserved we can bound (and measure) the maximum interference due to sharing the L2
- EFL requires including some extra features in the time-randomized processor we developed within PROXIMA

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EFL Required modifications

- Stall Module
  - A mechanism to ensure a given core is not evicting data until at least MID cycles have elapsed
  - Requires some counters to control eviction frequency is preserved for all cores
  - Requires relatively simple modifications in the L2
    - A new port is provided to allow signalling a give core is not entitled to perform a new eviction
      - An access that hits in the L2 can is processed normally
    - Sends an SPLIT command to the master (core) that cannot resume the execution until its counter reaches zero

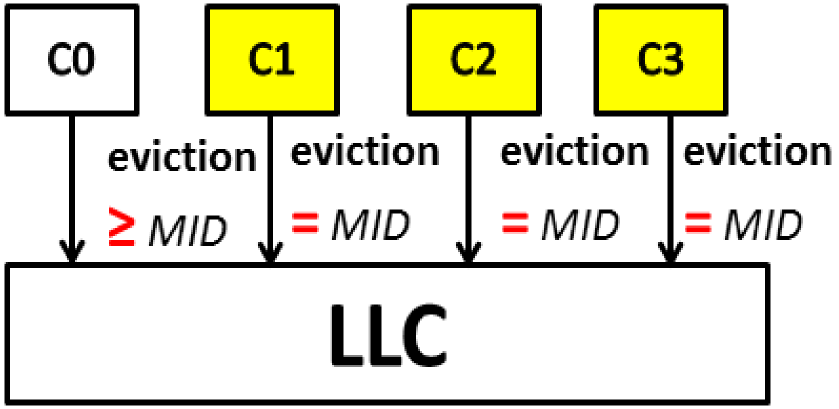


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Obtaining Time-composable WCET with EFL

- EFL allows to derive composable WCET estimates running tasks in isolation → do not assume any miss rate

- Analysis mode
  - Hardware
    - An Eviction module signals at the maximum rate (every MID cycles) a new eviction
    - When the L2 receive the evict signal evict randomly a cache line (the set and way are random and provided by random bits available in the randomized processor)
  - Software
    - Microbenchmarks causing the worst potential interference are run with the task under analysis



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

www.bsc.es

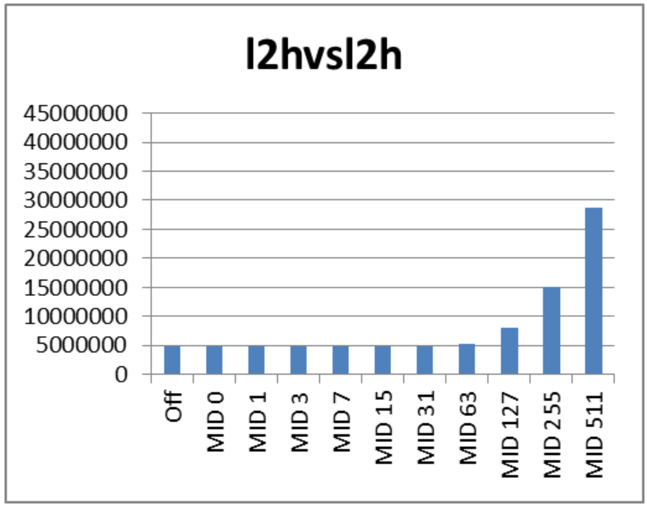
Results

Experimental setup

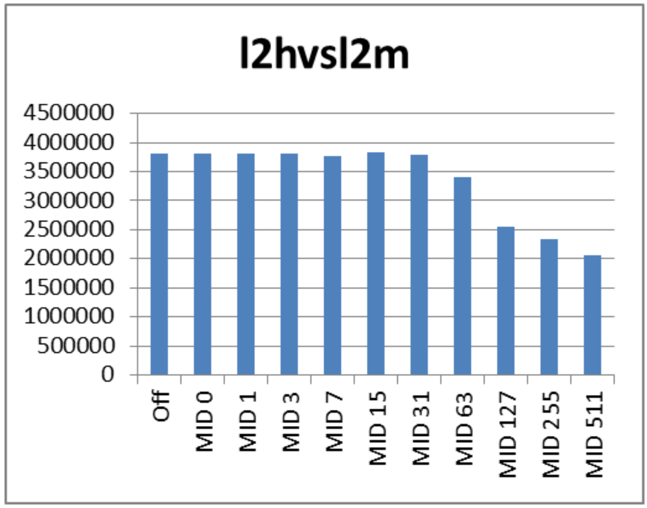
- Processor configuration
  - 4-core LEON3 incorporating the NGMP L2 module (128KB)
  - Synthetized on the ML510 board
- Software micro-benchmarks:
  - *l2h*: load operations missing in DL1 and hitting in L2.
  - *l2m*: load operations missing in DL1 and in L2.
  - *s2h*: As l2h but using store operations to access the L2.
  - *s2m*: As l2m but using store operations to access the L2.
  - *l2-64kb*: benchmarks that miss in DL1 and hit in L2 with higher footprint than l2h.
  - *l2-96kb*: IDEM, but its footprint is 96KB.
- EFL is configured using different MID values
  - We evaluate the impact of EFL on the task under analys (TUA)

L2h vs 3 copies of (l2h,l2ms2h,s2m)

- *l2h vs l2h*: l2h is quite stable under different MIDs.
  - When MID reaches the hundreds we observe how l2h increases its execution time.
    - The overhead of the contenders in cache is low, so increasing MID is not providing any benefit
    - For high MID few misses affect TUA performance



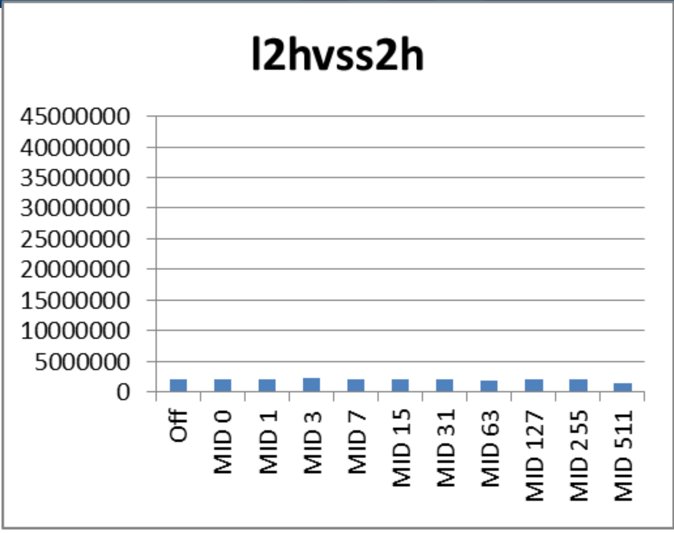
- *l2h vs l2m*: l2h is quite stable under different MIDs.
  - For MID>31 we observe how l2h increases its performance (reduction in execution time).
    - l2m evict some of l2h data from L2 when MID is configured low,
    - but being able to hit more consistently when l2m is slowed down by EFL.



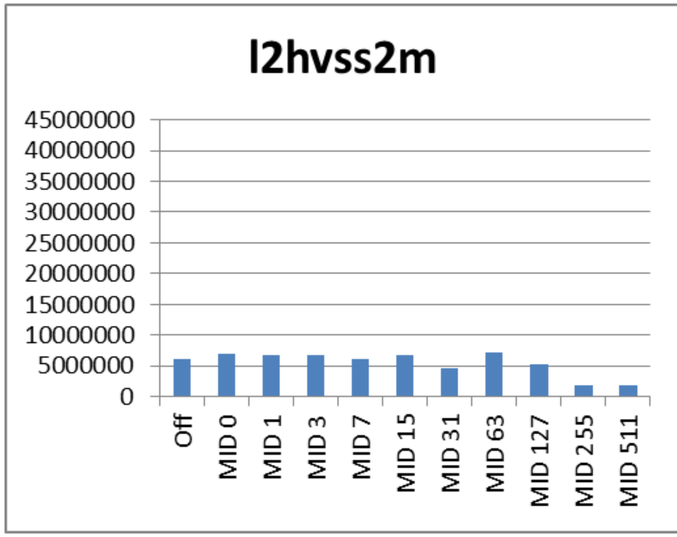
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

L2h vs 3 copies of (l2h,l2ms2h,s2m)

- *l2h vs s2h*: Performance of l2h is unaffected
  - Both benchmarks are able to hit the majority of the time in L2

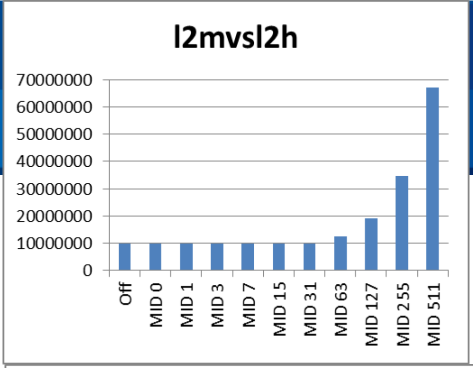


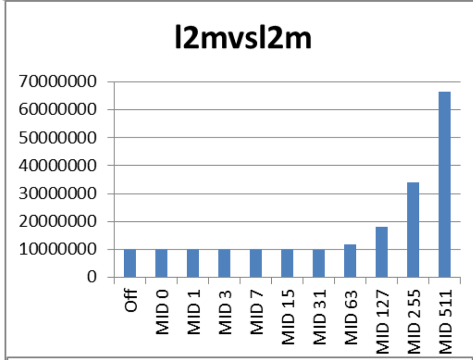
- *l2h vs s2m*: makes a wave form
  - EFL has little impact for low MID but in general makes the TUA to run slightly slower → not preventing s2m to evict data and extra misses are penalized by EFL
  - MID=31 → reduces TUA evictions and improves its performance since high MID values do not affect it much
  - Until high MID values are reached performance can be reduced (MID=31) or increased (MID=63)
  - Very high MID values improve the performance of the TUA since they prevent data to be evicted.

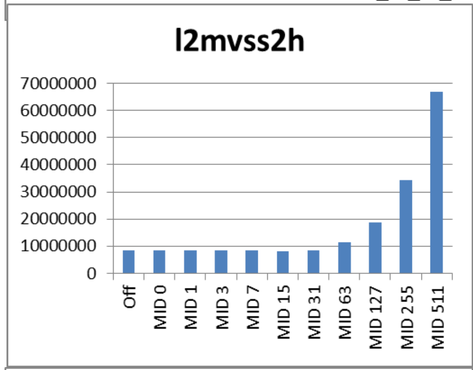


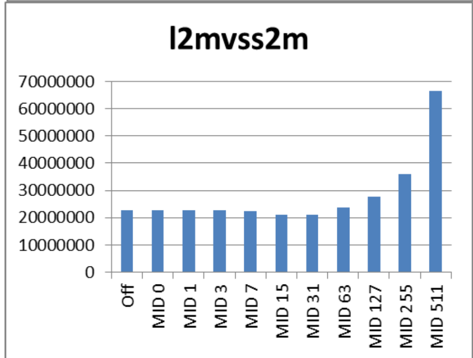
L2m vs 3 copies of (l2h,l2ms2h,s2m)

- *l2m vs l2h*: l2m causes several L2 cache evictions.
  - Increasing values of MID cause the TUA to run slower.
- *l2m vs l2m*: The TUA behaves almost identically as in the l2m vs l2h experiment,
- *l2m vs s2h*: Similar results to l2m vs l2h, as the contender tasks are not slowed down by EFL
- *l2m vs s2m*:
  - The TUA is slowed down by the contenders for low MIDs.
  - MID=15 where l2m is able to improve its execution time, as s2m contenders get stopped by EFL.
  - For big values of MID, l2m performs worse, as EFL noticeably stops the TUA too.





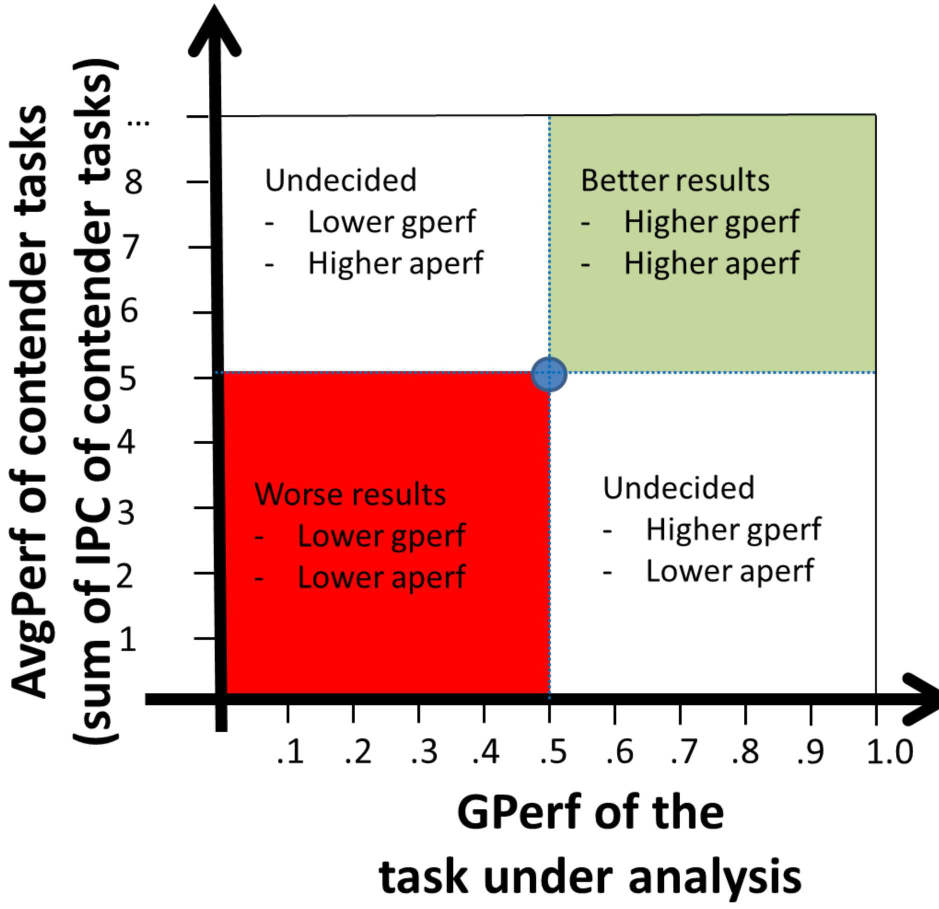




Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Quality of Service (QoS) results

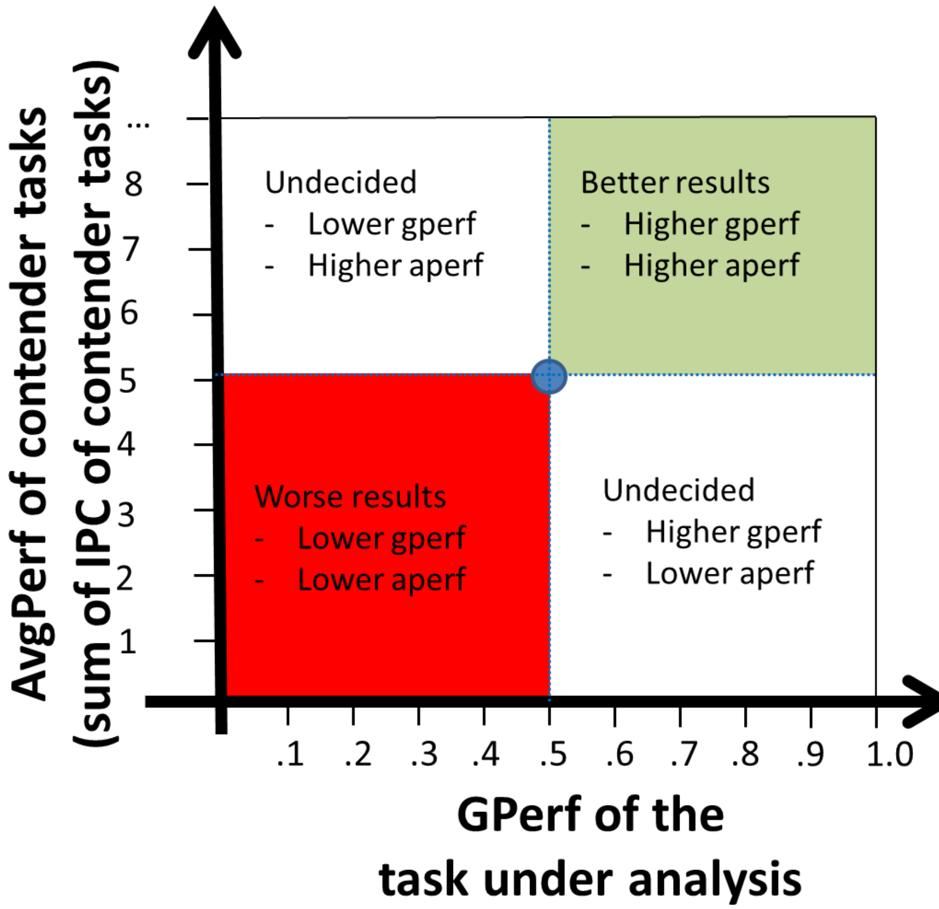
- EFL can improve the performance of the TUA but penalizes overall system performance
  - x-axis:
    - performance task under anal.
    - Shown in percentage
    - 100% means as much performance as the *TUA* gets when it runs in isolation.
  - The y-axis
    - IPC throughput, that is, the addition of the Instructions Per Cycle (IPC) executed by contender tasks



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

QoS results

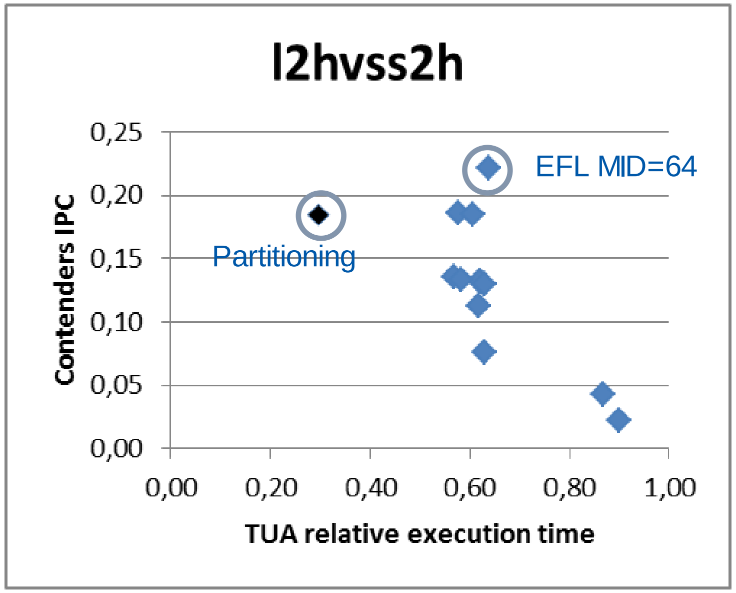
- Results
  - Top-right corner
    - better since they achieve higher aperf and gperf.
  - bottom-left corner
    - worse since the result in lower aperf and gperf.
  - top-left bottom-right diagonal
    - cannot be categorized as better or worse
    - they obtain lower gperf and higher aperf or vice-versa.

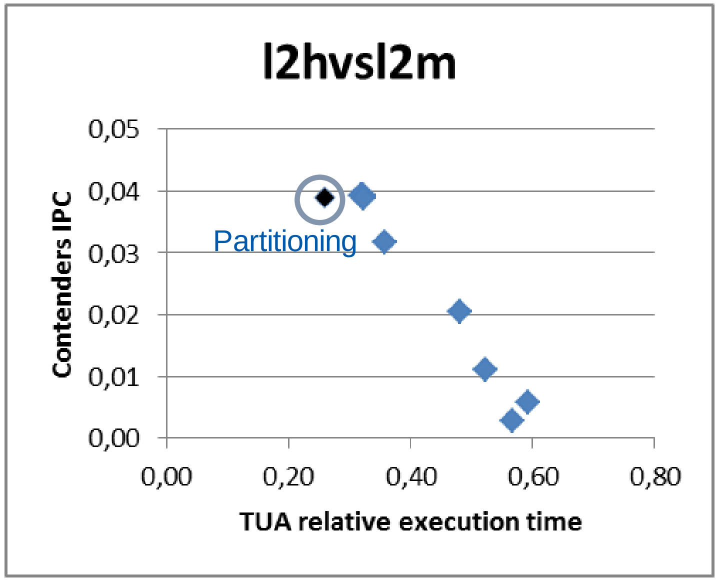


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Some examples

- We need to understand how our application will behave to asses the suitability of the EFL mechanism
  - For applications with high average L2 hit rates EFL provides good results





MID=64 improves both TUA guarantees and multicore throughput

High MID values increase TUA guarantees but degrade multicore throughput

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

www.bsc.es

Conclusions

Conclusions

- In this activity we have evaluated the suitability of the EFL mechanism in a NGMP-like processor
- EFL reduces the problems of cache partitioning in terms of data sharing and task mapping:
  - tasks may have shared pages or libraries whose actual sharing is complicated since cache partitioning does not allow tasks to share data on-chip.
- Our results show that EFL is a very effective technique to achieve good performance guarantees in a non-partitioned L2 cache setup at a reasonable hardware cost
  - Only few counters and minor modifications in the L2 were required

www.bsc.es

Assessment of the Implementation of the EFL Time-Randomised Cache in the NGMP Architecture

Francisco J. Cazorla (TC),
Carles Hernandez,
Jaume Abella,
Leonidas Kosmidis

Jan Andersson,
Nils-Johan Wessman

Marcel Verhoef (TO)

Software Systems Division & Data Systems Division Final Presentation Days
09/05/2017