www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Improving Early Design Stage Timing Modeling in Multicore Based Real-Time Systems

David Trilla	Barcelona Supercomputing Center (BSC) and Universitat Politècnica de Catalunya
Javier Jalle	Cobham Gaisler*
Mikel Fernandez	Barcelona Supercomputing Center (BSC)
Jaume Abella	Barcelona Supercomputing Center (BSC)
Francisco J. Cazorla	BSC and and IIIA-CSIC

* Done while he was working at BSC

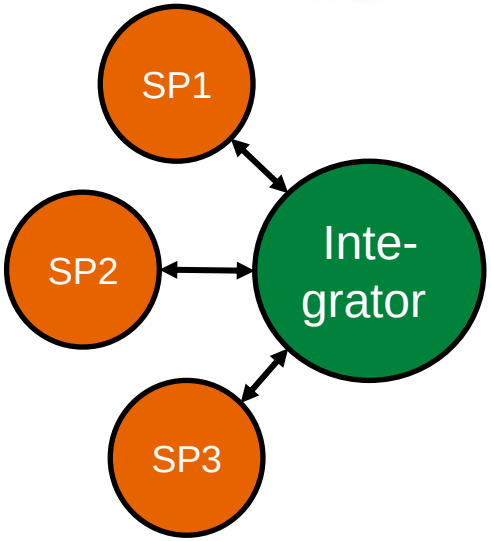
22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2016)

Vienna, Austria

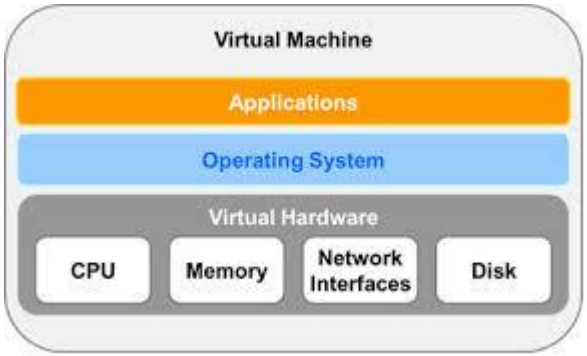
Motivation

- System integrators increasingly incorporate software (SW) from different SW Providers (SP)

- In Early-Design Phases (**EDP**) each SP is provided
  - A set of functions to implement
  - A time budget
  - A virtual machine (e.g GMV's HAIR) of the target platform



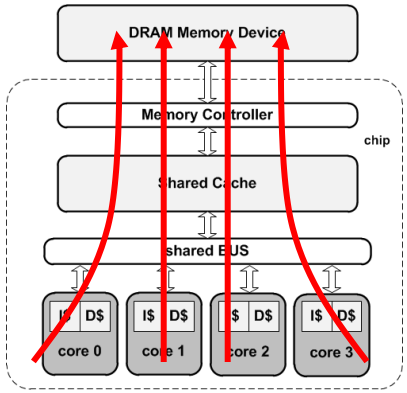
- Virtual Machines (VM)s
  - Allow functional testing
  - Allow suppliers develop SW functions in isol.
  - Fail to provide timing estimates of the executed applications



Problem

- Early Design Phases
  - Uncertainties: preliminary implementations
  - Timing requirements are bounded with early estimates
    - Overestimation → removes the uncertainties, over-provisioned system
    - Underestimation → costly changes in late design phases (LDP)

- Multicores
  - Tasks execution time: $et^{muc} = et^{solo} + \Delta t$
  - Δt depends on co-runner tasks
    - SP cannot derive et^{muc} without sharing their apps.
  - Scheduling depends on et^{muc} and vice-versa
    - How to assign and enforce timing budgets?
    - Problem for the system integrators



Proposal

- VM do not provide timing estimates of applications
  - Full-fledge timing simulator attached to the VM is too slow (100-1000x)

- We propose an approach for EDP that
  - Provides fast and accurate timing estimates of tasks' execution time when the target (virtual) hardware comprises multicore
  - Extends virtualized environments with a light-weight timing model that
    - i) provides high accuracy and low overhead
    - ii) does not require code or binaries to be shared among SW provider (keeping the confidentiality on their developed software)

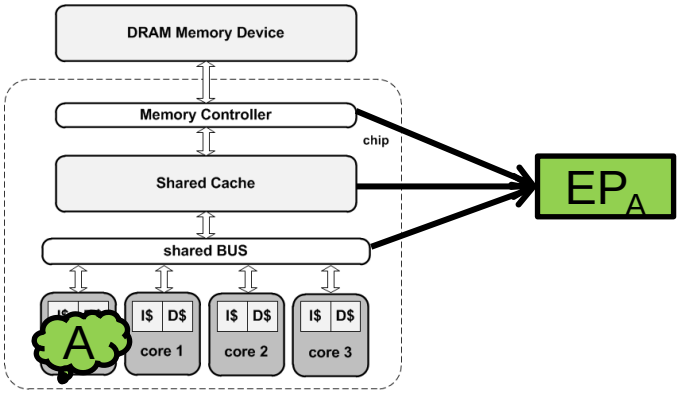
- Overall: our proposal simplifies and speeds up the process of getting timing estimates during the EDP

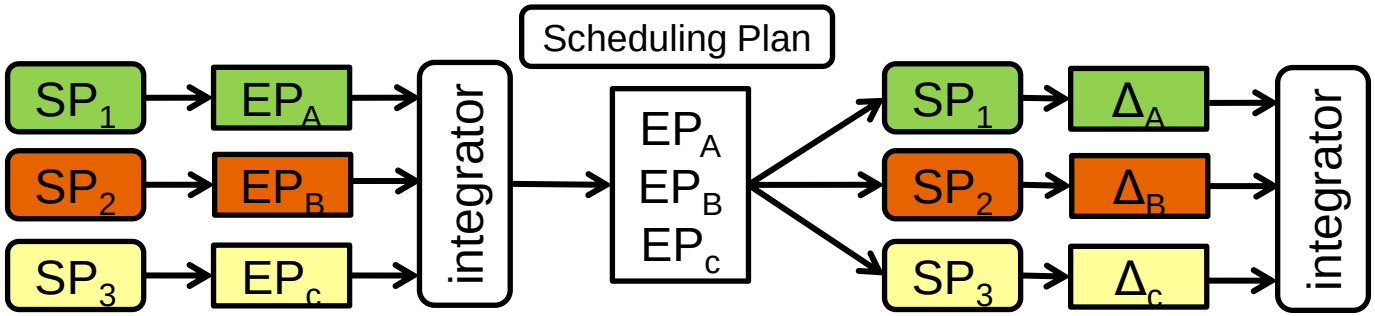
Outline

- Motivation
- **Principle and main steps**
- Contention modelling ($\Delta t = \Delta cache + \Delta bus + \Delta mem$)
  - Cache contention ($\Delta cache$)
  - Bus contention ($\Delta bus + \Delta mem$)
- Results
- Conclusions

Principle

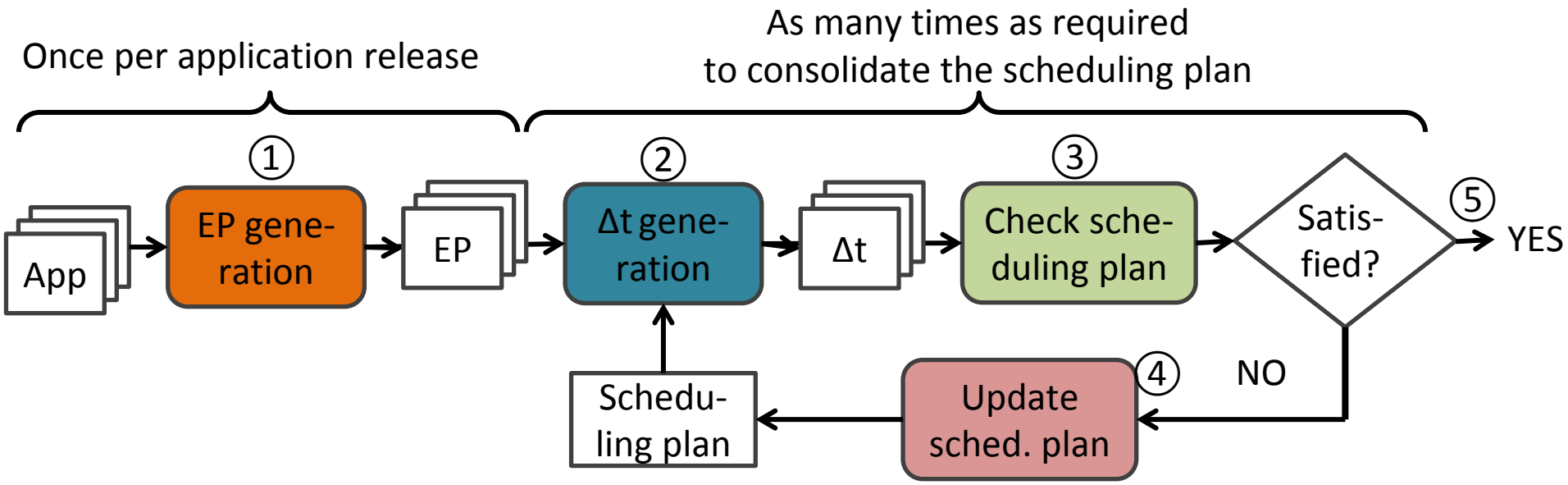
- Builds upon the concept of an execution profile (EP)
  - Derived in isolation for each task
  - Encapsulates for each task information about its resource usage





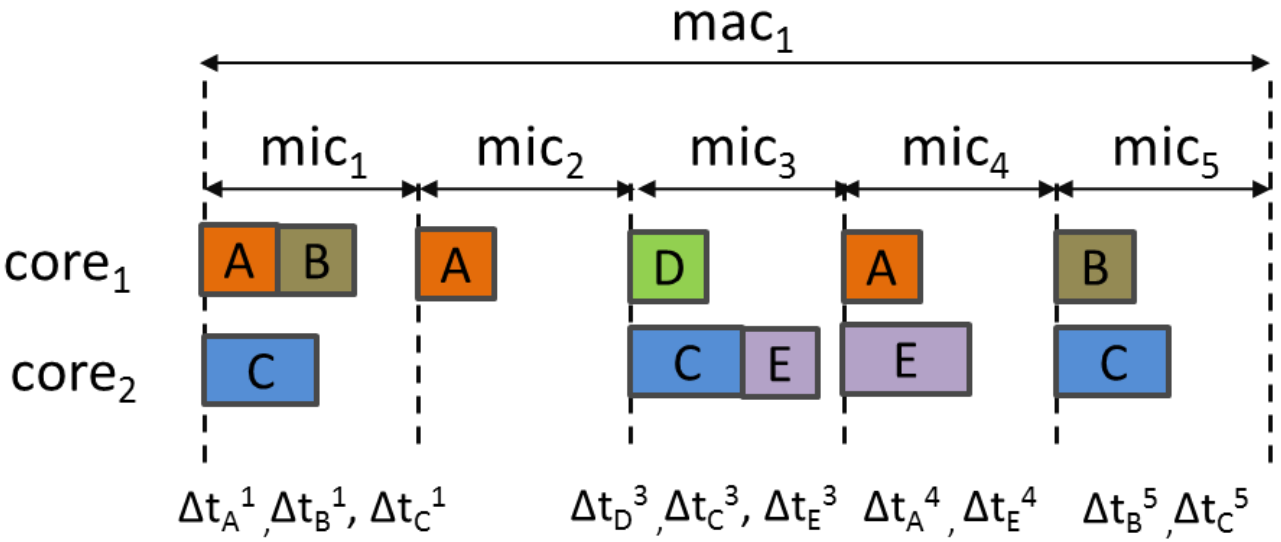
Steps

- 1) Generating the EP for each task and then
- 2) Contention modelling (Δt generation): must be fast!
- 3) Check scheduling plan and if required update it (4)



Example: dual core and cyclic executive

- Cyclic executive scheduling (widely used in industry)
  - Major cycles (mac) and minor cycles (mic)
- Scheduling plan provided by the OEM
  - Suppliers can determine co-runners of their application in *mic*
- In *mic1 A* and *B* interact with *C*
  - Derive Δt_A^1, Δt_B^1, Δt_C^1,
  - If both et_C^{muc} *and* (et_A^{muc} + et_B^{muc}) fit in a mic no change to the schedule (for this first *mic*) is required



Outline

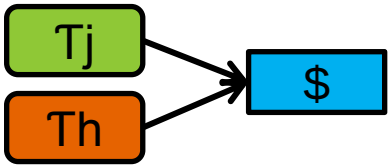
- Motivation
- Principle and main steps
- **Contention modelling ($\Delta t = \Delta cache + \Delta bus + \Delta mem$)**
  - Cache contention ($\Delta cache$)
  - Bus contention ($\Delta bus + \Delta mem$)
- Results
- Conclusions

Contention modelling /1

- Simulators
  - Keep the state of the modelled HW in SW data structures
  - On an access, data structures are searched and their internal state is appropriately updated
  - Time-consuming process
- Our approach
  - Keep no information about the execution history
  - Model each instruction in isolation
  - We use in the EPs to predict contention impact
    - Build a representative scenario so that the timing behaviour of the instruction approximates that of the real execution
    - EPs comprise distributions → histograms

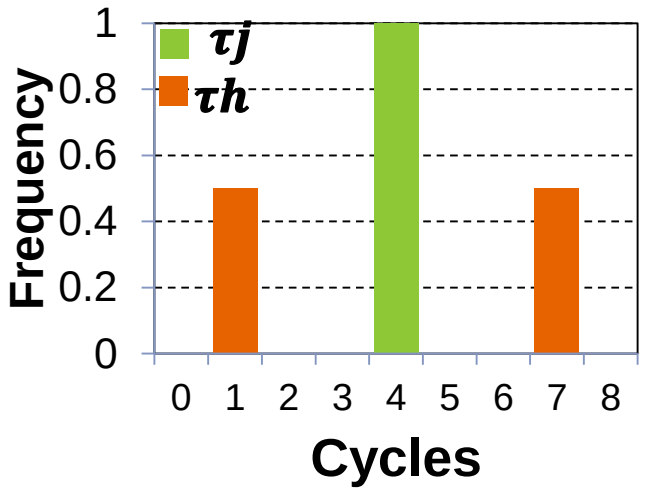
Contention modelling /2

- Single entry cache. 2 accessing tasks (each accesses 1 line)
  - Consecutive accesses from the same task → hit (1 cycle) and vice versa (miss 10 cycles)



- Example case: 100 accesses

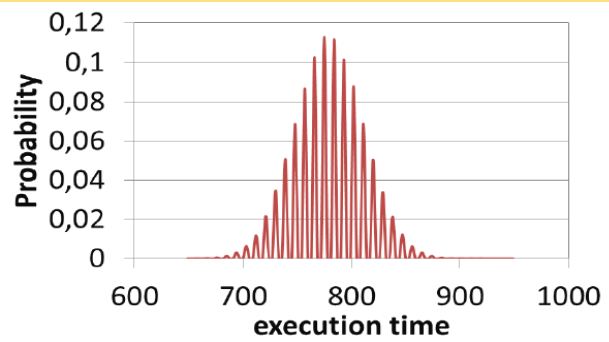
Frequency of access of each task



Average-based approach

1000 cycles

Histogram-based approach



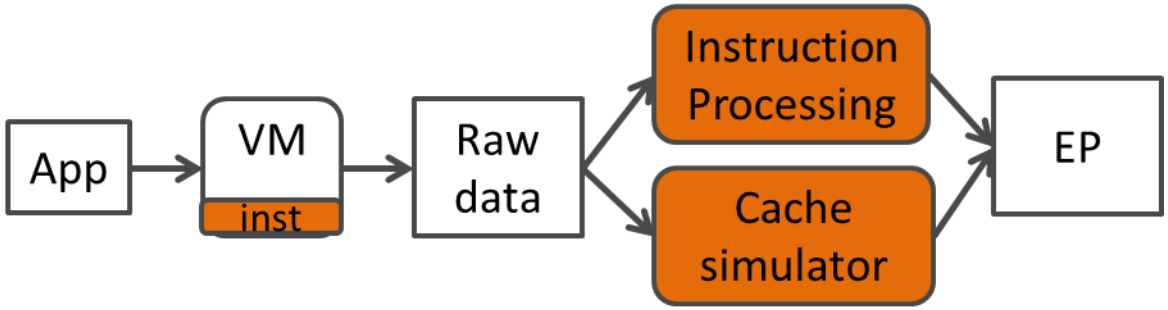
(b) τ_h's execution time prediction

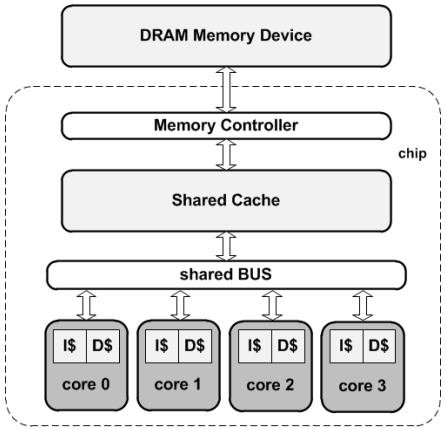
Contention modelling /2

- Histogram: sequence of pairs <value, probability>
- Realization ( ):
  - Process to get a sample from the histogram
  - Example
    - <1, 0.1><2, 0.4><3.0.5>
    - Generate a random number r from (0 to 1]
  - $outcome = \begin{cases} 1\ if\ (0.0 < r \leq 0.1) \\ 2\ if\ (0.1 < r \leq 0.5) \\ 3\ if\ (0.5 < r \leq 1.0) \end{cases}$

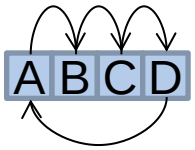
- Each EP comprises several histograms

Execution Profile

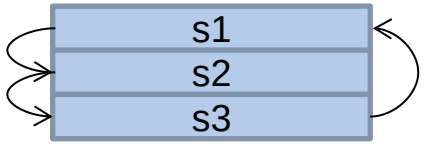




- Histograms:
  - Time between accesses going to the same set in uL2
  - Memory Stack Distance: # of unique addresses to the same set between two accesses to the same line.
  - Cache Set Distance: # of accesses to different sets between two accesses to the same set.
- Other relevant Data:
  - Hit rates of caches
  - Number of instructions
  - Instruction Mix
  - Etc…



A – B – C – D – A (3)



s1 – s2 –s3 –s1 (2)

Execution time in isolation

- Execution time = (pipeline) frontend + (pipeline) backend lats.

$$et_j^{solo} = \sum_{y \in \mathcal{Y}} [n_y \times (fend(y) + bend(y))]$$

Fixed

Bounded

Backend latencies: Jittery units → worst-case

- Jittery back-end operations
  - In general caused by the particular values operated
- For operations with jittery back end latency we assume the worst case latency

TABLE II
OPERATION TYPES IN THE NGMP AND THEIR ASSUMED LATENCIES

operation type	jitter	min-max latency	Assumed latency
int. short latency	NO	1	1
int. long latency	YES	1-35	35
control	NO	1	1
fp. short latency	NO	4	4
fp. long latency	YES	16-25	25

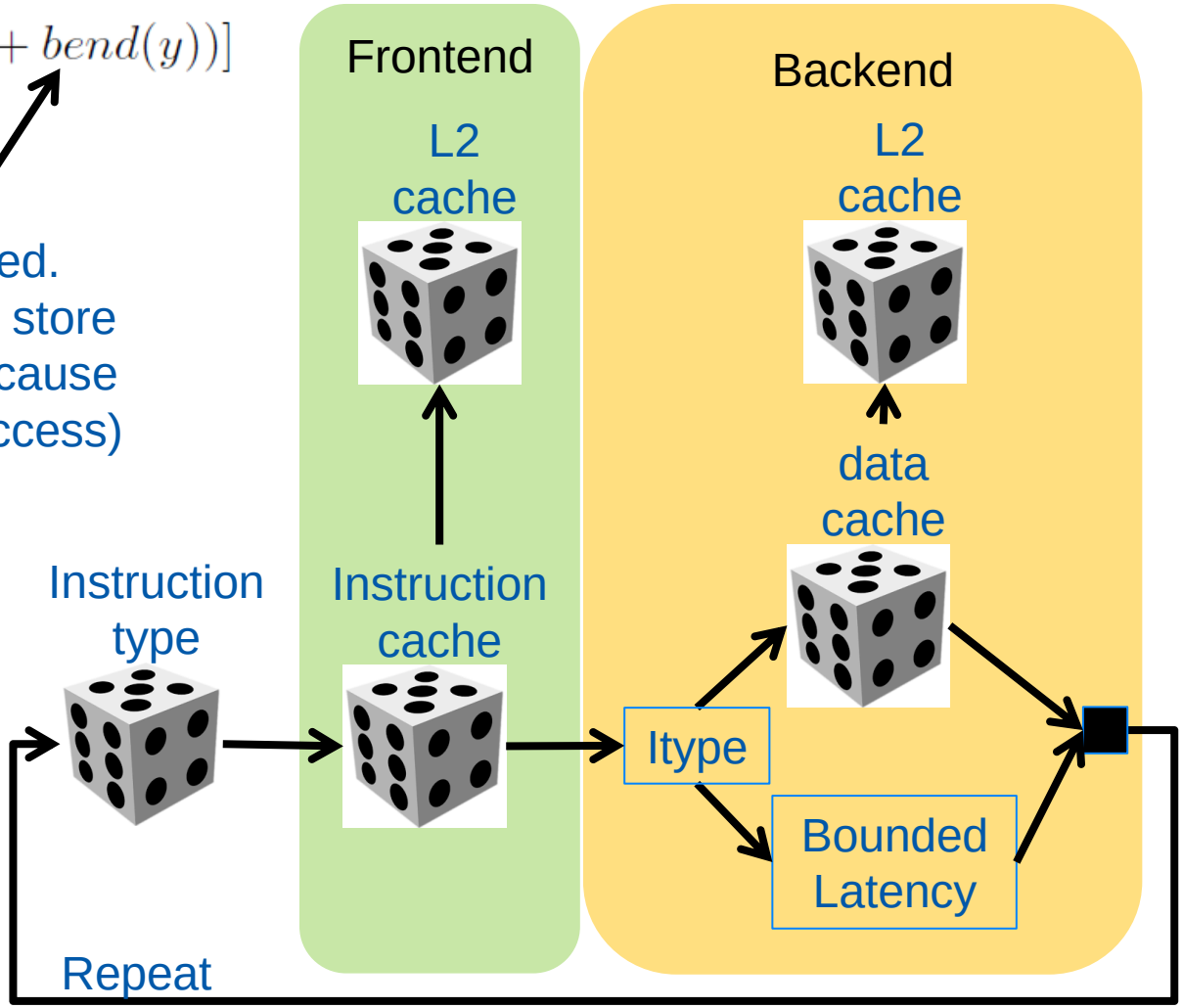
Execution time in isolation

- Execution time = (pipeline) frontend + (pipeline) backend lats.

$$et_j^{solo} = \sum_{y \in \mathcal{Y}} [n_y \times (fend(y) + bend(y))]$$

Fixed (may cause IL1 access)

Bounded. (load and store ops. can cause an DL1 access)



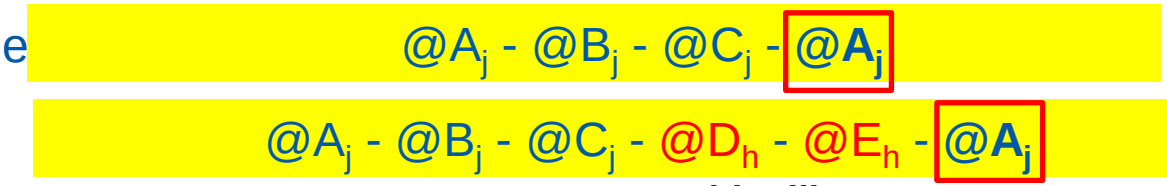
Outline

- Motivation
- Principle and main steps
- Contention modelling ($\Delta t = \Delta cache + \Delta bus + \Delta mem$)
  - **Cache contention ($\Delta \boldsymbol{cache}$)**
  - Bus contention ($\Delta bus + \Delta mem$)
- Results
- Conclusions

Cache Contention modelling /1

- Setup
  - Private DL1 and IL1. Shared UL2
  - LRU replacement and non inclusive caches
- Appreciation
  - DL1 & IL1 hits; and UL2 misses in isol. → not affected by contention
  - UL2 hits in isolation → can become misses. For an access, with LRU:
    - If its stack distance is smaller to W (number of ways) → hit
    - If it is greater or equal than W → miss
- Goal
  - Derive stack distance in isolation for every access of τ_j
  - Derive delta in stack distance due to its corunners (τ_h) contention
  - Determine τ_j which hits become misses.

4-way cache example



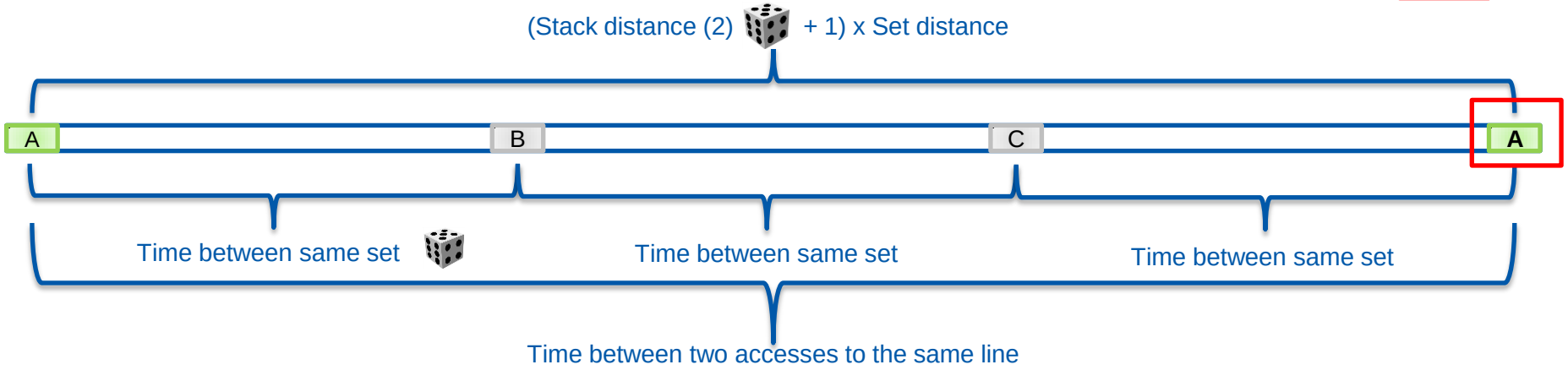
@$\mathbf{A_j}$ → hits

@$\mathbf{A_j}$ → misses

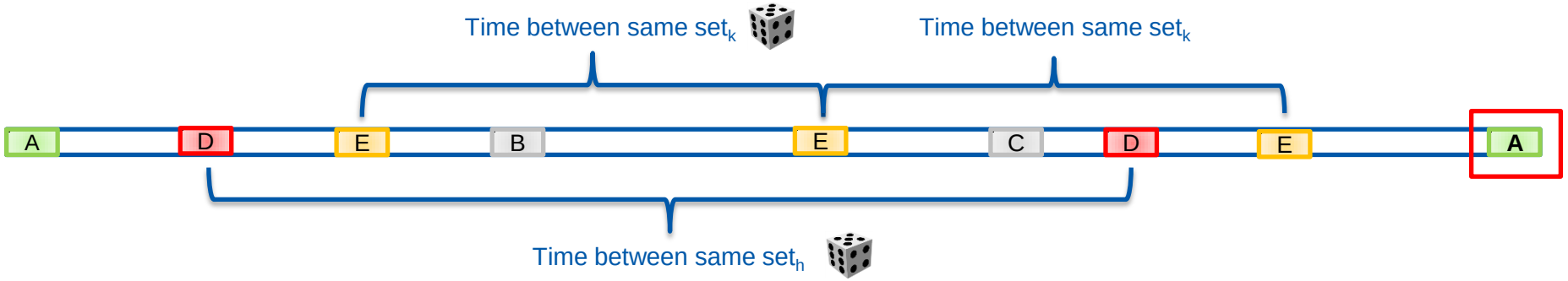
Cache Contention modelling /2

τ_j Analysis - Time between accesses

$@A_j$ - - $@A_j$



$\tau_{h,}\tau_k$Contenders - Requests between τ_j accesses



$@A_j$ - $@D_h$ - $@E_k$ - $@B_j$ - $@E_k$ - $@C_j$ - $@D_h$ - $@E_k$ - $@A_j$

Cache Contention modelling /3

- Set Dispersion or Average set distance (ASD)
  - Contender's Accesses can be mapped to different sets
  - Probability that τ_hs' intermediate accesses maps to @A_j's same set

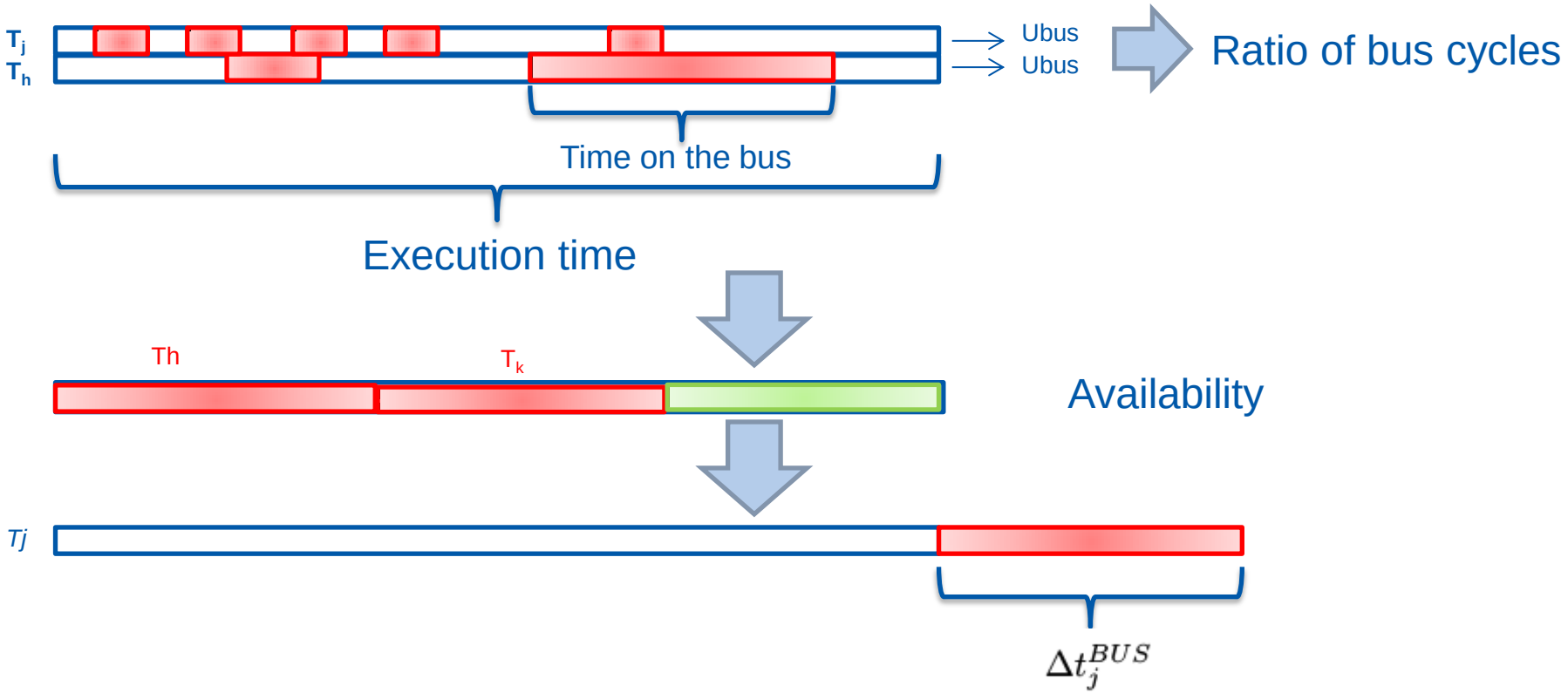
- Increment in stack distance
  - Not all contender's accesses to the same set increase stack distance:
    - Contender's lines can be accessed repeatedly
    - Realization over contender's stack distance
  - Assumption: all τ_hs' intermediate accesses have the same stack dist.

Outline

- Motivation
- Principle and main steps
- Contention modelling ($\Delta T = \Delta cache + \Delta bus + \Delta mem$)
  - Cache contention ($\Delta cache$)
  - **Bus contention** ($\boldsymbol{\Delta bus + \Delta mem}$)
- Results
- Conclusions

Bus Contention modelling /1

- Goal: Derive how many cycles of contention T_j will pay for each cycle of bus usage ($Increase\ in\ \ bus\ cycles$)
- No bus split accesses ($\Delta bus + \ \Delta mem$)

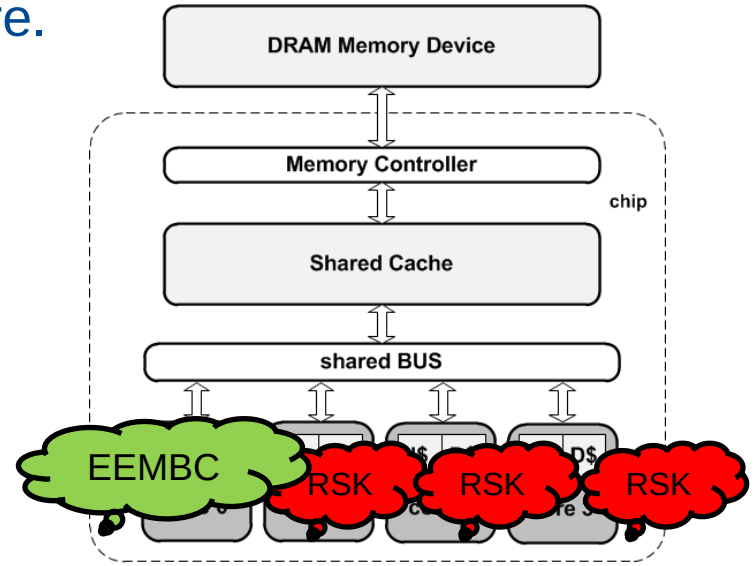


Outline

- Motivation
- Principle and main steps
- Contention modelling ($\Delta t = \Delta cache + \Delta bus + \Delta mem$)
  - Cache contention ($\Delta cache$)
  - Bus contention ($\Delta bus + \Delta mem$)
- **Results**
- Conclusions

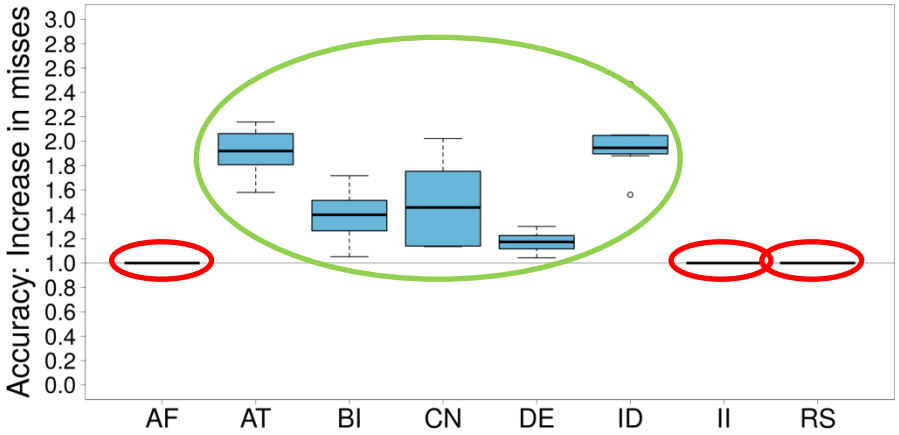
Results /1 (Setup)

- Reference architecture
  - NGMP-model developed in the SoCLiB framework
  - Accuracy assessed against real hardware.
- Key Parameters:
  - Level 1 instruction & data cache:
    - 16kB, 4-way associative
  - Level 2 unified cache:
    - 256 kB, 4-way associative
  - Round-Robin arbitrated bus



- Metrics:
  - Accuracy of the model predicting the slowdown of most sensitive EEMBC when running against resource stressing kernels (RSK)
  - Time overhead of the model

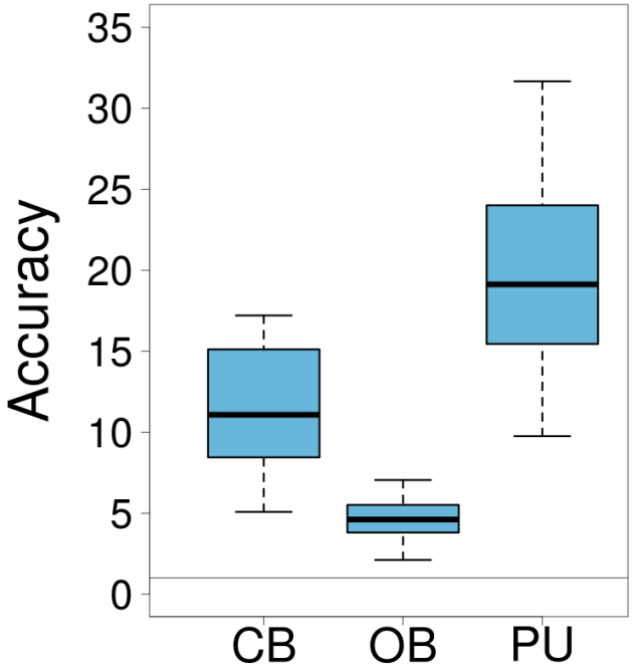
Results /2 (Cache Interference Accuracy)



- The model detects cases with “no interference”.

- Upper bounded because of access pattern behavior

Accuracy: [1,2.47]

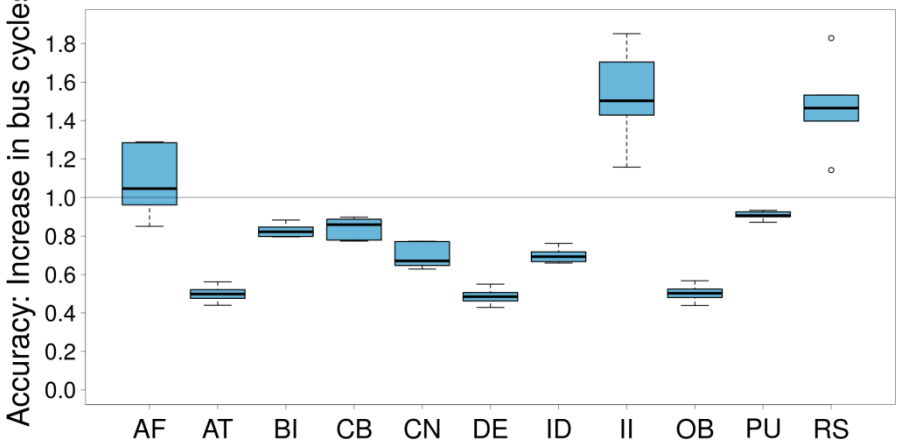


0.3% 0.4%

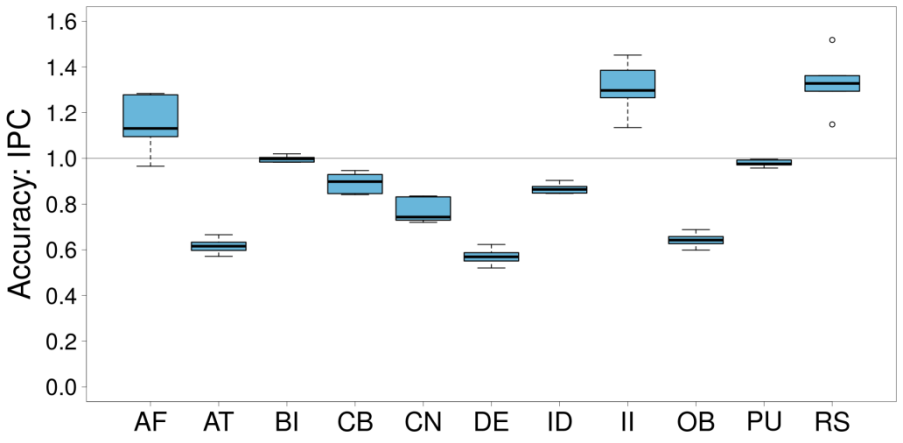
0.4%
Time impact of extra misses

Results /3 (Bus & Global Accuracy)

- Bus Accuracy:



- Global Accuracy



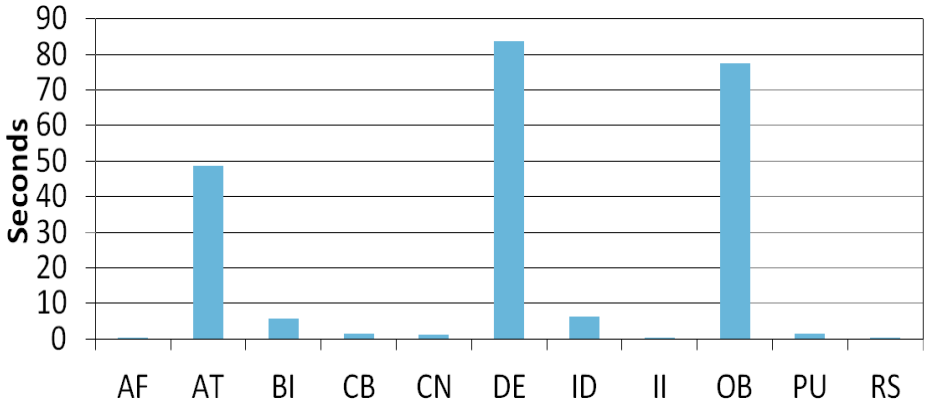
- Super linear effect introduced by store buffers.

Accuracy: [0.43, 1.85]

Global Accuracy: [0.6,1.4]

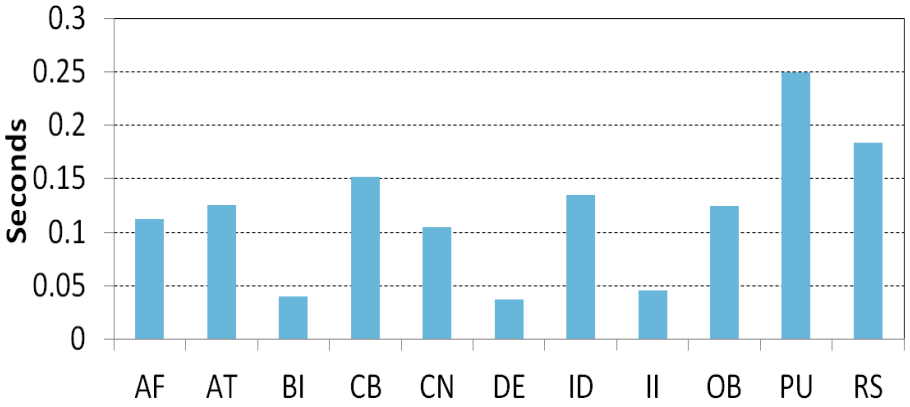
Results /4 (Execution Time Overhead)

- EP Generation



- Average: 41 seconds

- Execution Time:



- Average: 0.12 seconds

- Multiple schedule plans can be tested in very short time.

Outline

- Motivation
- Principle and main steps
- Contention modelling ($\Delta t = \Delta cache + \Delta bus + \Delta mem$)
  - Cache contention ($\Delta cache$)
  - Bus contention ($\Delta bus + \Delta mem$)
- Results
- **Conclusions**

Conclusions and Future work

- Modeling multicore scenarios based on histograms
- Fair good results for the NGMP:
  - Accurate enough
  - Fast predictions
- No need to exchange sensible information amongst SP.

- Future Work:
  - Model the behaviour of store buffers
  - Improve the impact of simplified scenarios

Q & A

- The research leading to these results has received funding from the European Space Agency under Project Reference AO/1-7722/13/NL/LvH

- Spanish Ministry of Science and Innovation TIN2015-65316-P.
- MINECO under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717.

www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Improving Early Design Stage Timing Modeling in Multicore Based Real-Time Systems

David Trilla	Barcelona Supercomputing Center (BSC) and Universitat Politècnica de Catalunya
Javier Jalle	Cobham Gaisler*
Mikel Fernandez	Barcelona Supercomputing Center (BSC)
Jaume Abella	Barcelona Supercomputing Center (BSC)
Francisco J. Cazorla	BSC and and IIIA-CSIC

* Done while he was working at BSC

22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2016)

Vienna, Austria