www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

PMCs for real-time multicore systems: analysis of the state of the art and initial proposal

Francisco J. Cazorla, Jaume Abella
Javier Jalle, Mikel Fernandez
Leonidas Kosmidis

Luca Fossati (TO)
Marco Zulianello

Software Systems Division & Data Systems Division Final Presentation Days
ESTEC WWednesday 10th of December 2014

Goal

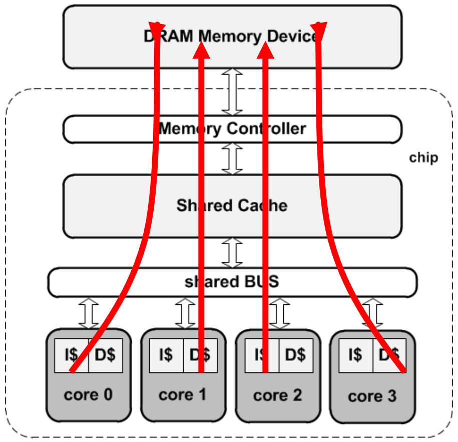
- Performance Monitoring Counter (PMC) infrastructure that helps providing insight information about the effect of contention among corunning tasks (a.k.a. corunners)
  - Focus on the NGMP

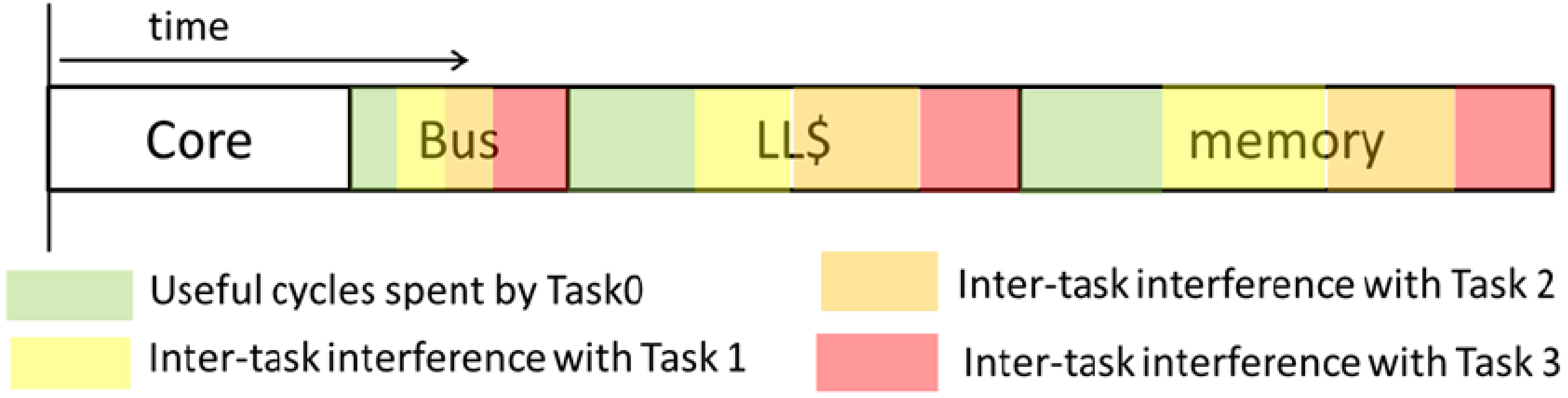
- After running my application in a multicore I want to know
  - Where my cycle went?
  - How many cycles in waste due to contention

- Contention Cycle Stack

Contention cycle stack: concept

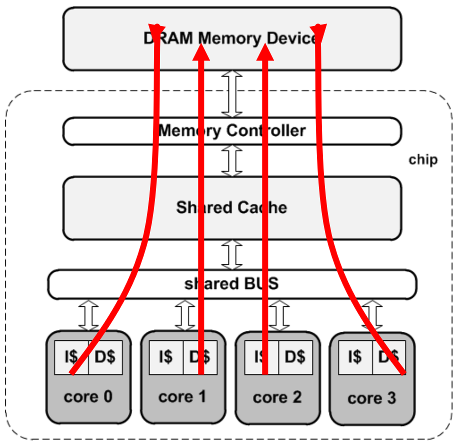
- Core cycles:
  – Cycles in which the program is proceeding with no stalls (useful cycles)
  – Cycles in which the pipeline is stalled (no instruction can be committed) due to a local stall

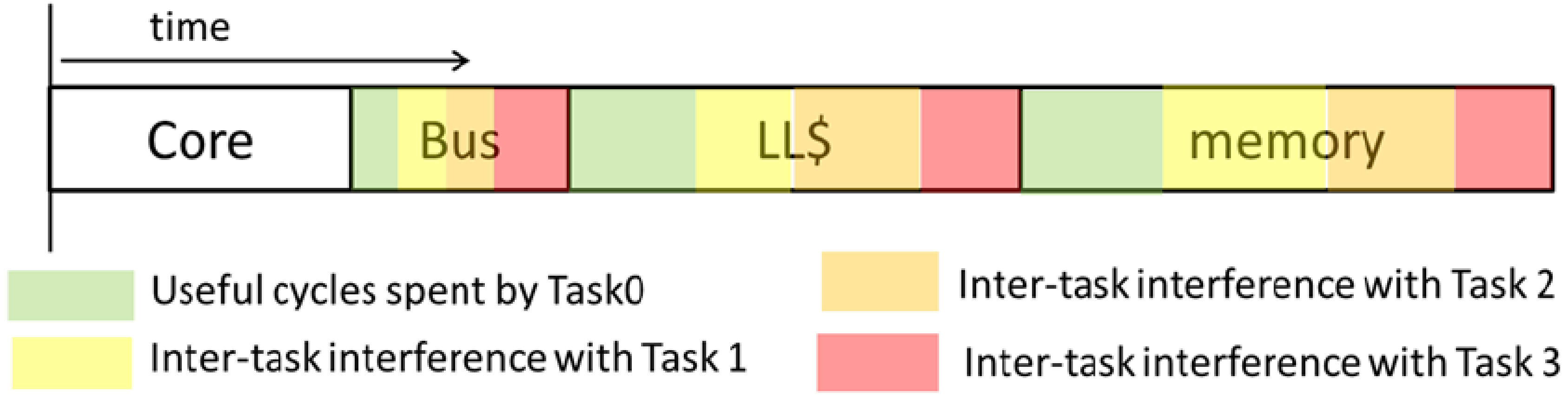



time
Core
Bus
LL$
memory
Useful cycles spent by Task0
Inter-task interference with Task 1
Inter-task interference with Task 2
Inter-task interference with Task 3

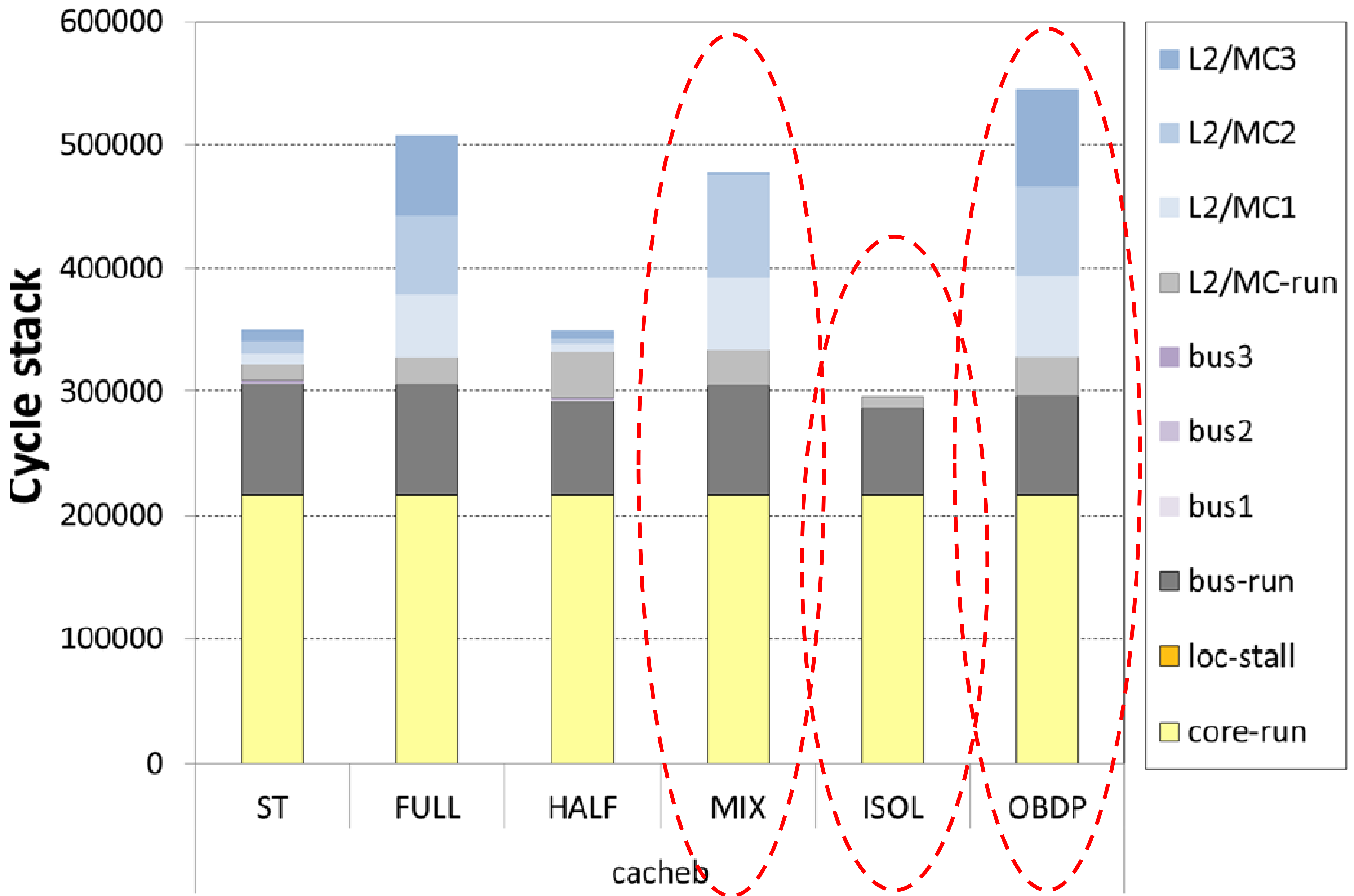
Contention cycle stack: concept

- For each shared resource (bus, L2cache, memory)
  - Cycles in which the task (t0) actually use the resource (green)
  - Cycles in which the task was stalled due to other tasks (t1, t2, t3)
    - Yellow, orange, red → cycles consumed due to contention




time
Core
Bus
LL$
memory
Useful cycles spent by Task0
Inter-task interference with Task 1
Inter-task interference with Task 2
Inter-task interference with Task 3

Contention cycle stack: example



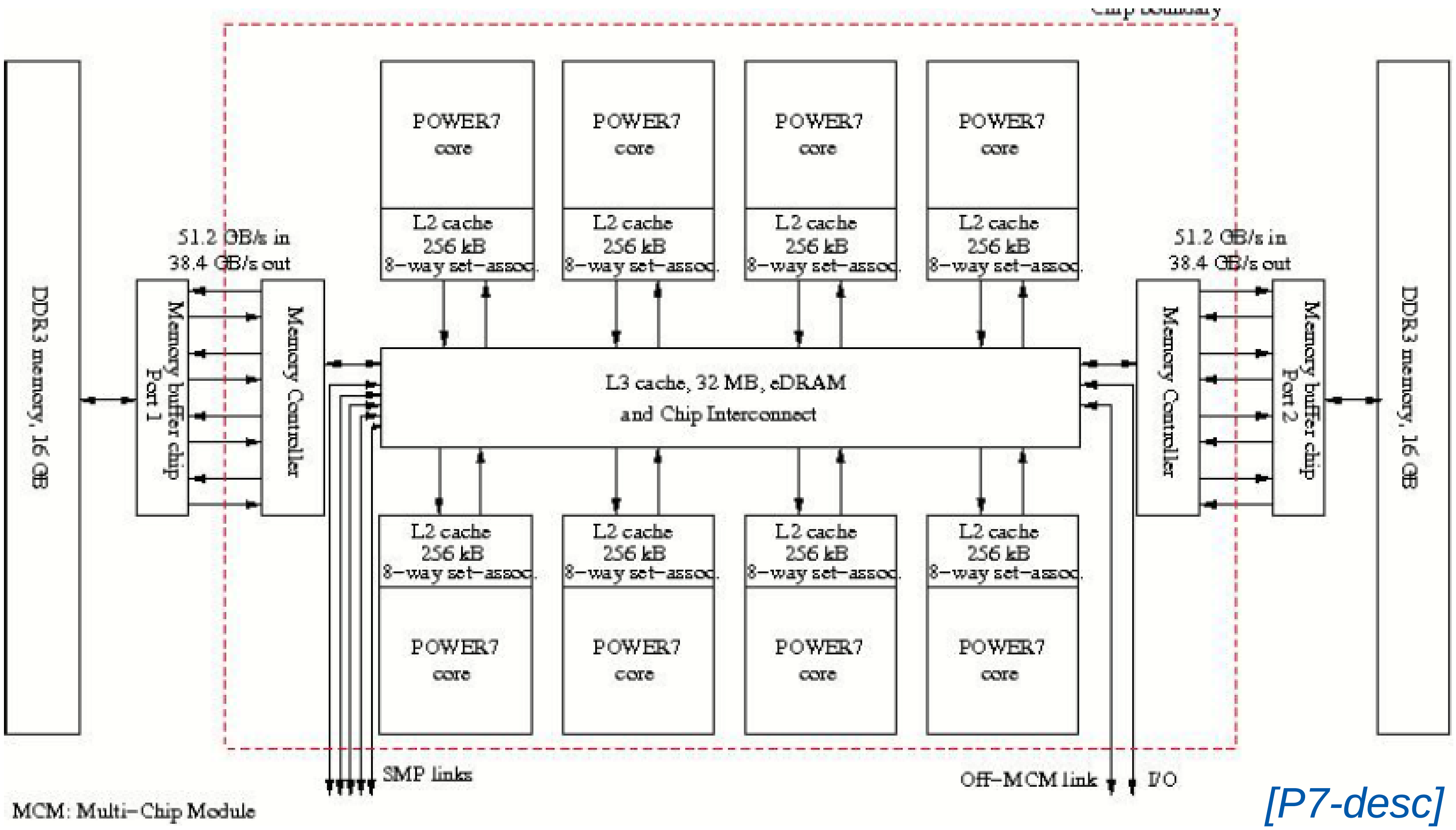
In a nutshell

- What are we doing?
  - Using PMCs to provide the end user a radiography of its application in terms of consumed cycles in the processor
  - Not interested in the breakdown of cycles at core level, but breakdown contention cycles
- Contention cycle Stack Benefits
  - Scheduling
    - Determine bad/good corunners
  - Timing Analysis
    - Determine in the worst-case path(s) how cycles are consumed

Outline

- Analysis of current PMCs
  - IBM POWER7
  - Intel Family
  - ARM v7
  - P4080
  - NGMP
- Initial thoughts about contention cycle stack

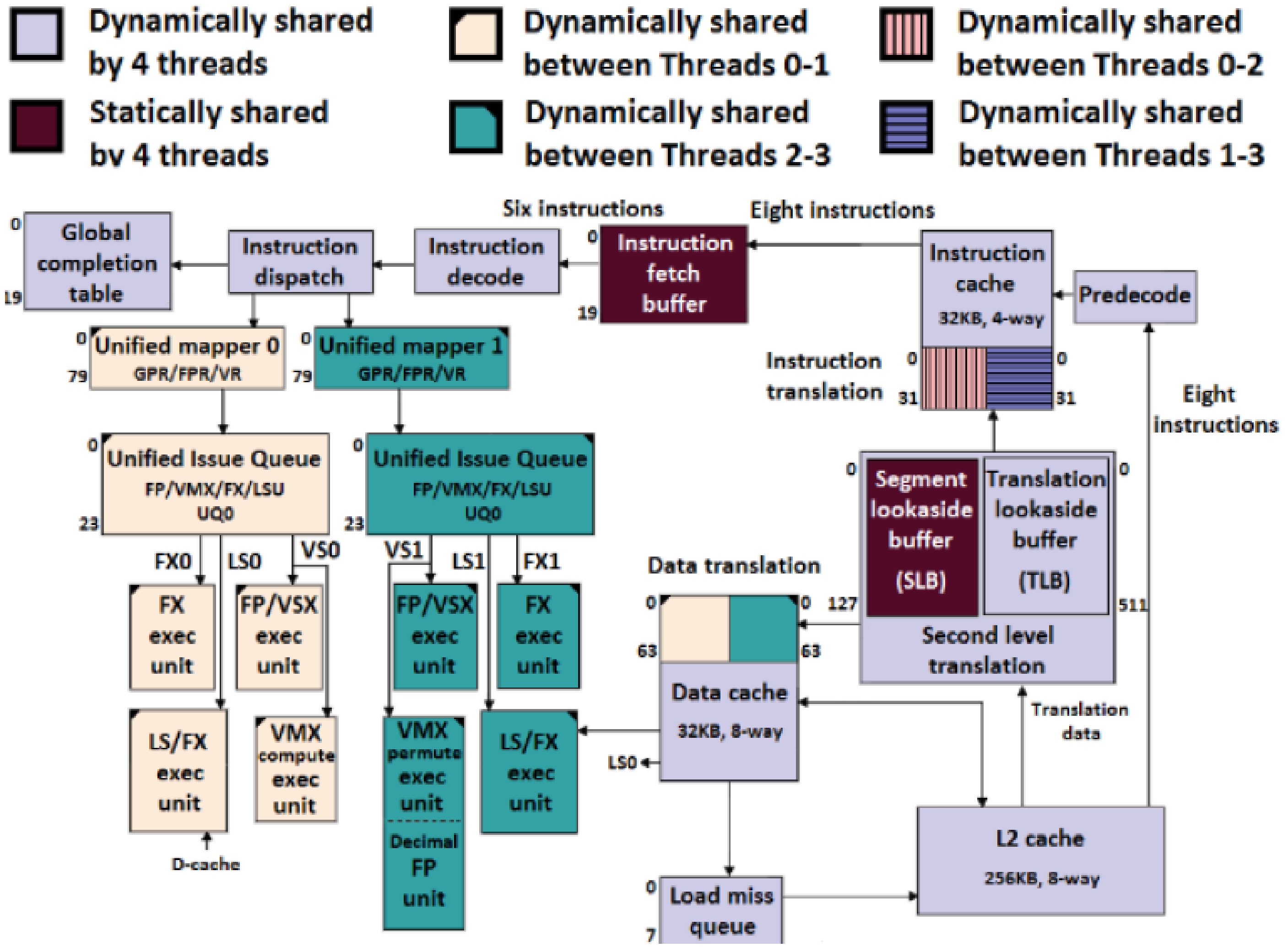
IBM POWER7



[P7-desc]

[P7-desc] B. Sinharoy et al. IBM POWER7 multicore server processor. IBM J. Res. Dev. , 55:191–219, May 2011

IBM POWER7



Dynamically shared by 4 threads
Dynamically shared between Threads 0-1
Dynamically shared between Threads 0-2
Statically shared by 4 threads
Dynamically shared between Threads 2-3
Dynamically shared between Threads 1-3
Global completion table
Instruction dispatch
Instruction decode
Six instructions
Instruction fetch buffer
Eight instructions
Instruction cache 32KB, 4-way
Predecode
Unified mapper 0 GPR/FPR/VR
Unified mapper 1 GPR/FPR/VR
Instruction translation
Eight instructions
Unified Issue Queue FP/VMX/FX/LSU UQ0
Unified Issue Queue FP/VMX/FX/LSU UQ0
Segment lookaside buffer (SLB)
Translation lookaside buffer (TLB)
Second level translation
FX0
LS0
VS0
VS1
LS1
FX1
Data translation
FX exec unit
FP/VSX exec unit
LS/FX exec unit
VMX compute exec unit
VMX permute exec unit
Decimal FP unit
Data cache 32KB, 8-way
Translation data
D-cache
Load miss queue
L2 cache 256KB, 8-way

[P7-desc]

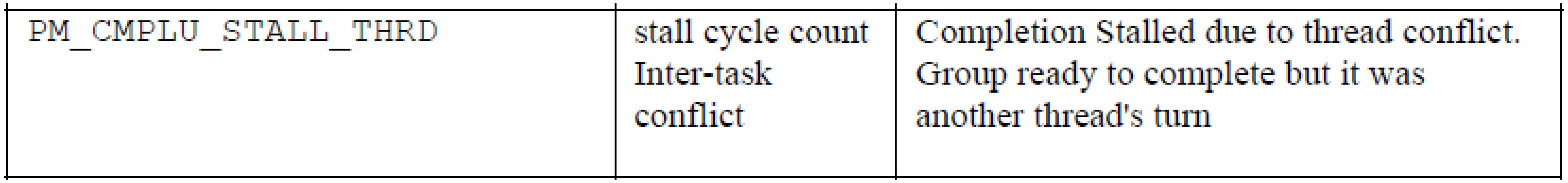
[P7-desc] B. Sinharoy et al. IBM POWER7 multicore server processor. IBM J. Res. Dev. , 55:191–219, May 2011

IBM POWER7

- Performance Monitoring Unit (PMU)
  - Six thread-level Performance Counter Monitors (PCMs).
  - Four of these are programmable from software to monitor the desired (four) events at the same time.
  - There are more than 500 possible performance events that can be read.
  - However, performance counters are defined by groups and the PMU can only watch events of the same groups at one time.
  - Some counters are per-thread and others are per-core

IBM POWER7

- Type of PMCs
  - Number of cycles a resource is full (with this causing the stall of the processor),
  - Number of cycles a resource is empty.
    - This can be of a private resource of a shared resource. The latter meaning that none of the threads that can generate a
  - request to that resource have done so.
  - Number of instructions of a given type
  - Number of events of a given type (e.g. prefetch requests sent, …)
  - Number of references to a given resource (e.g. L2 accesses, …)
  - Quantity of data transferred
  - Stall cycles due to inter-task conflict



PM_CMPLU_STALL_THRD	stall cycle count Inter-task conflict	Completion Stalled due to thread conflict. Group ready to complete but it was another thread's turn

Intel

- Processors:
  - superscalar execution, complex branch predictors, out of order execution, and several levels of cache memories (up to three).
- PMCs provided by Intel:
  - Focus on providing performance metrics for a single process (branch predictor effectiveness, cache misses due to speculative execution, coherence protocol metrics, etc)
  - Not designed to help identify bottlenecks in resources shared between cores
  - Nor to quantify the magnitude of interactions in the shared resources.
    - A lot of counters provide measurements for time spent accessing a shared resource
    - They measure average or total accumulated time, and they do not identify possible interferences caused by other cores.

Intel

- Many shared resources have PMCs that measure their behaviour number of accesses, average latency, etc)
- With some experimentation, information about inter-processor conflicts can be guessed:
- Number of L2 cache lines evicted by another process
  - Comparing the observed number of L2 misses when running in isolation with the number of L2 misses when running concurrently with a competing process.

Intel

- We have classified PMCs in:
  - Instruction type: no. of retired instructions of a type (branch, load, …)
  - Event count: microinstructions issued, branch instructions executed ...
  - Reference count: number of referenced lines in each MESI state…
  - Threshold exceeding event count: number of times a threshold specified in number of cycles has been exceeded for a given event.
  - Number of outstanding requests per core: cache requests, all offcore requests, etc. in the moment of reading the counter.
  - Busy resource cycles: number of cycles in which the caches are busy, a resource is unavailable, stalled core, etc.
  - Cycle count: unhalted core cycles, unhalted thread cycles, cycles with outstanding cache misses, TLB walk duration, etc.

ARMv7-A architecture

- Provides 6 different 32-bit counters
- Used by several different processors: Cortex-A7, Cortex-A9, Cortex-A15, the big.LITTLE system, and others.
- Includes a set of control registers to allow performance monitoring. The most important of them are:
  - PMXEVCNTR, which holds the value of the configured PMC.
  - PMSELR, which configures the counter that will be used when counting an event.
  - PMXEVTYPER, which selects the event that will increment the selected counter.
  - PMCCNTR, which counts the amount of cycles (or cycles/64).
  - Counters are set and cleared using the PMCNTENSET and PMCNTENCLR registers.

ARMv7-A architecture

- The events counted by ARM architectures can be classified in the following types:
  - Instruction type: number of retired instructions of a certain type (branch, load, store, etc).
  - Event count: miss-predicted branches, number of exceptions, etc.
  - Reference count: number of L1 accesses, bus accesses, data memory accesses, unaligned accesses to memory, etc.
  - Cycle count: CPU cycles, bus cycles.
  - Resource-specific event count: L1 write backs, number of L1/L2 refills.

NGMP*

- NGMP architecture is continuously evolving and, indeed, also the counters are changing
  - Data about LEON4 is that by the time the activity was carried out
- It
- L t
  t

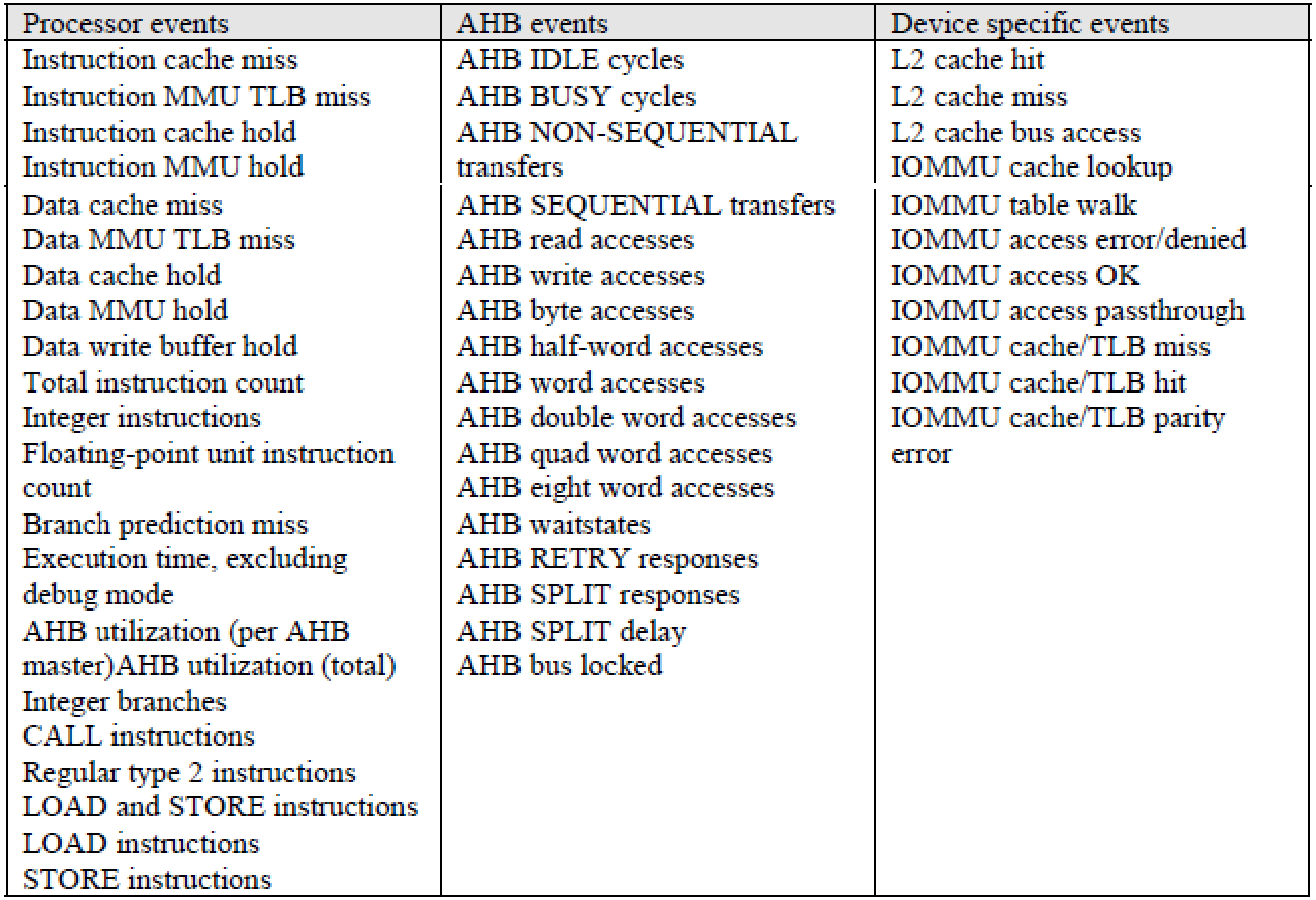
The counters and stat unit has evolved with respect to what presented in the slides

(in part also thanks to the results of this activity)

e

2

NGMP



Processor events	AHB events	Device specific events
Instruction cache miss	AHB IDLE cycles	L2 cache hit
Instruction MMU TLB miss	AHB BUSY cycles	L2 cache miss
Instruction cache hold	AHB NON-SEQUENTIAL transfers	L2 cache bus access
Instruction MMU hold	AHB SEQUENTIAL transfers	IOMMU cache lookup
Data cache miss	AHB read accesses	IOMMU table walk
Data MMU TLB miss	AHB write accesses	IOMMU access error/denied
Data cache hold	AHB byte accesses	IOMMU access OK
Data MMU hold	AHB half-word accesses	IOMMU access passthrough
Data write buffer hold	AHB word accesses	IOMMU cache/TLB miss
Total instruction count	AHB double word accesses	IOMMU cache/TLB hit
Integer instructions	AHB quad word accesses	IOMMU cache/TLB parity error
Floating-point unit instruction count	AHB eight word accesses	
Branch prediction miss	AHB waitstates	
Execution time, excluding debug mode	AHB RETRY responses	
AHB utilization (per AHB master)AHB utilization (total)	AHB SPLIT responses	
Integer branches	AHB SPLIT delay	
CALL instructions	AHB bus locked	
Regular type 2 instructions		
LOAD and STORE instructions		
LOAD instructions		
STORE instructions		

NGMP

- Available counters can be as:
  - Busy resource cycles: the resource is unavailable because it is busy. For example AHB busy cycles
  - Idle resource cycles: the resource is not being used. For example AHB idle cycles
  - Cycle count: other events counting cycles. For example, CPU cycles.
  - Instructions of a given type: Load, store, floating point, integer, total count.
  - Event count: number of mispredictions, IOMMU errors,
  - Reference count: AHB accesses L1 and L2 accesses and misses
  - Maximum Count mode

NGMP

- Maximum Count mode (Mcm)
  - While in this mode, the counter keeps the maximum amount of time the selected event has been asserted.
  - Count the maximum amount of time between two event assertions. Maximum count
- Using *Maximum Count Mode* it could be possible, e.g.:
  - To count the longest burst of AHB busy cycles
  - The longest amount of time the bus has been without having a read access.
- The availability of this counter is implementation dependant

Summary

- In all the studied architectures:
  - PMCs are used to improve average system,
  - performance by monitoring software execution,
  - characterizing processors behaviour, and/or
  - helping system developers bring up and debug their systems.
- Few exceptions of counters exist that help understanding the effect of inter-task interferences.
  - The POWER7, Intel, and the NGMP have been identified to have PMCs that provide some information about inter-task interferences and maximum (worst) duration of a stall event (situation).

Summary

- In general, we observe lack of detailed inter-task interference PMC support.
- Some information about inter-task interference can be derived in controlled scenarios.
  - In a first run the program under study is run in isolation recording PMCs.
  - In a subsequent run the program under study is run again, maintaining the same input data sets as part of the workload.
  - By subtracting the PMCs in the first run for those in the second run some inter-task interference information can be obtained.
  - Complexity
    - Analysing the program with the same input data in all runs
    - Alignment among tasks

Outline

- Analysis of current PMCs
  - IBM POWER7
  - Intel Family
  - ARM v7
  - P4080
  - NGMP
- Initial thoughts about contention cycle stack

Theoretical approach

- Break total execution cycles, t_i, into useful, u_i, and stalled, s_i
  - $t_i = u_i + s_i$
- s_i breakdown into
  - stalls due to local activities (e.g. handling a long latency FPU instruction, pipeline stalls, etc…), l_i
  - Stalls due to contention in shared resources, or *external stall,* e_i.
  - $s_i = l_i + e_i$

Theoretical approach

- e_i : For each resource n in N_R → inter-task interferences, int_i^n, covers cycles in which the processor *i* was stalled due to some contention from core *j*
  - $e_i = \sum_{n=0}^{N_R-1} u_i^n + int_i^n$
- int_i^n interference suffered by core i on resource n because of each core j of all N_C cores is defined as $int_{i \leftarrow j}^n$,
  - $int_i^n = \sum_{j=0}^{N_C-1} int_{i \leftarrow j}^n$
- The final *itiCPIstack* is given by:
  - $t_i = u_i + l_i + \sum_{n=0}^{N_R-1} \left[u_i^n + \sum_{j=0}^{N_C-1} int_{i \leftarrow j}^n \right]$

Implementation

- NGMP Simulator validated with a real processor implementation[1]
  - EEMBC Benchmark suite only 3% error at cycle level.
  - Real application only 0.9% error at cycle level
- Focus:
  - Bus and memory
  - L2 cache is partitioned

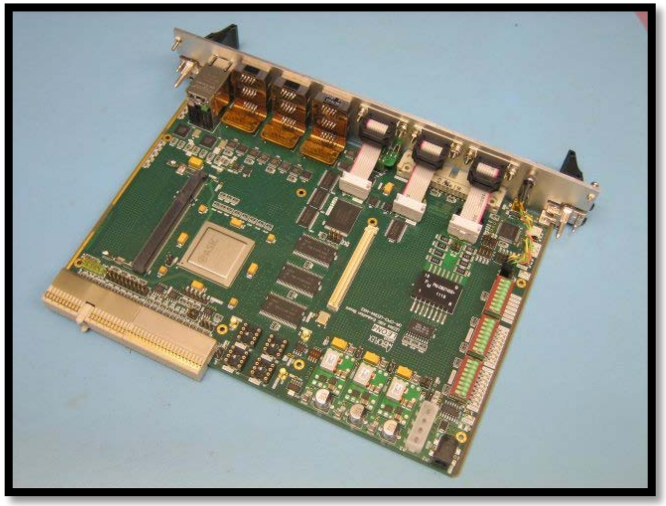
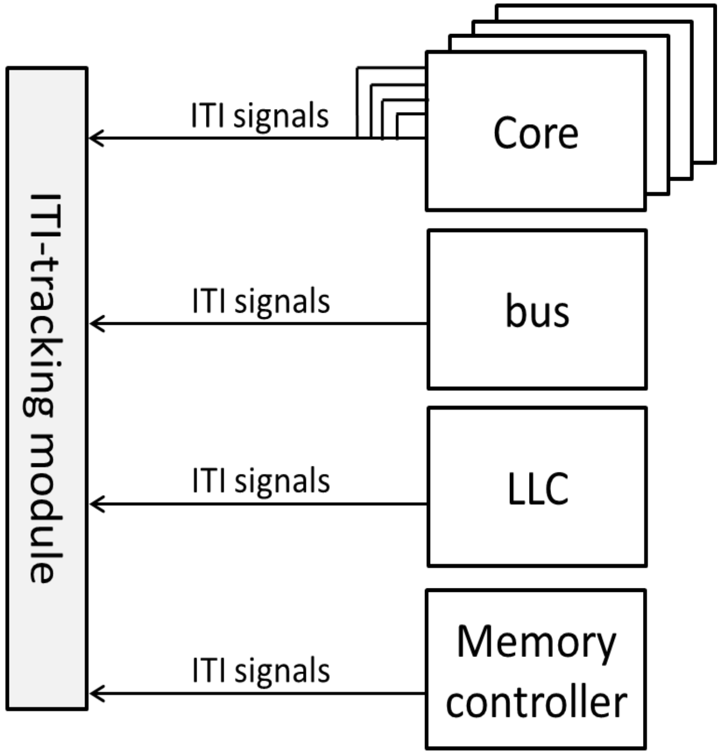


Figure: NGMP N2X Evaluation Board. Source: Aeroflex Gaisler

[1] ESA NPI. 1322010. Architectural solutions for the timing predictability of next-generation multi-core processor. Javier Jalle.

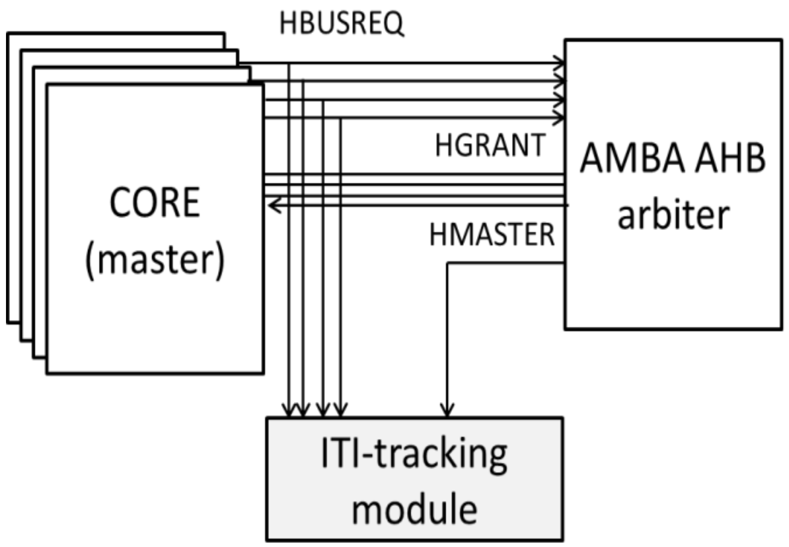
Implementation

- We envision having an inter-task interference module (*iti-module*), which is similar to a statistics unit, which concentrates the main logic required for computing the CPIstack.



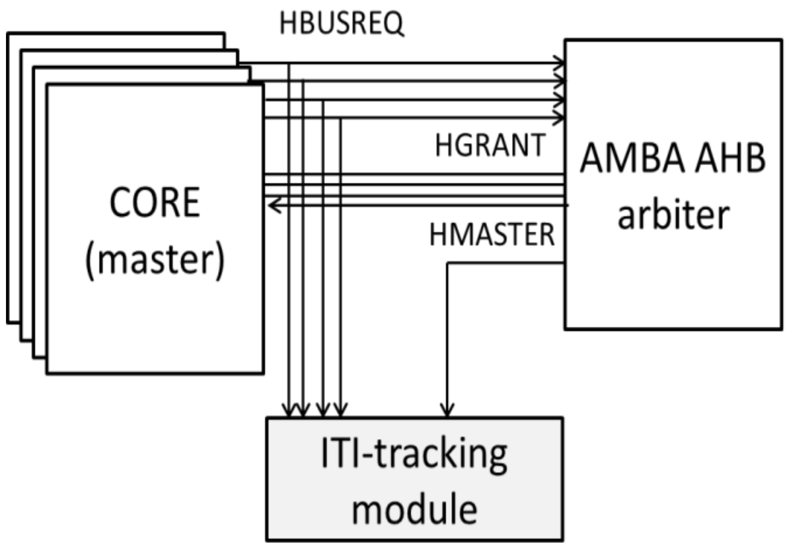
Example: breaking down contention in AMBA bus

- Master ready to send a request → HBUSREQ signal asserted
- When granted access:
  - HGRANT signal for that master and puts the
  - master id in the HMASTER signal.
- $t_{grant\text{-}up}$ - $t_{req\text{-}down}$ gives the time a request from a given task T_A is waiting to get access to the bus ($t_{A\text{-}iti}$) → contention delay



Example: breaking down contention in AMBA bus

- We propose to send HBUSREQ signals from each master and the HMASTER signal from the arbiter to the *iti-module*.
- By checking these signals the *iti-module* can infer which master is using the bus, thus the useful cycles, and the time a master has been waiting for another master.



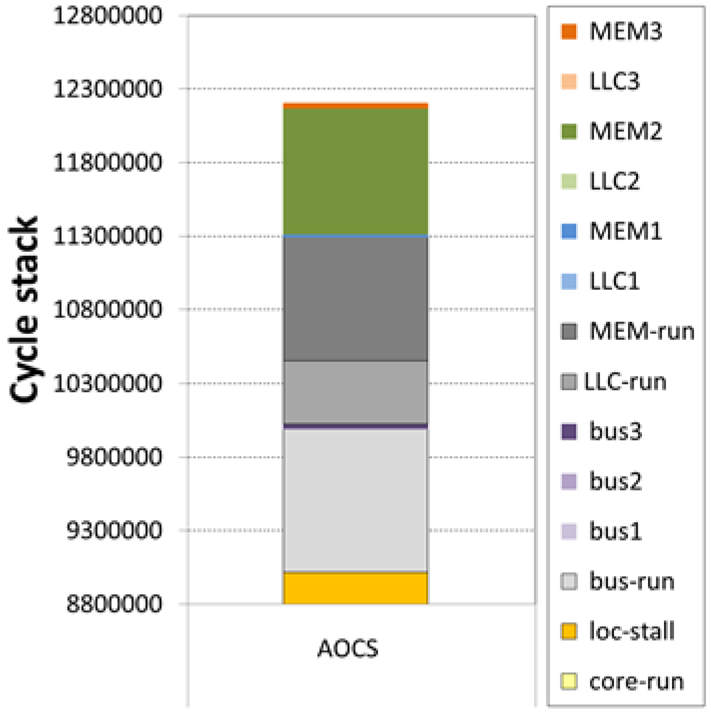
Memory

- Memory controller design:
  - Memory bus and memory controller only accept one request at a time
  - FIFO request queue to store L2 misses coming from the cores
- Request from a core i arrives to the
  - Queue empty → put on the top of the FIFO and accesses the memory immediately consuming the intrinsic latency or useful cycles ru_i^{mc}.
  - Queue not empty → wait other requests to finish since the
  - The cycles on the memory are:

$$cyc_i^{mem} = ru_i^{mem} + \sum_{j=0}^{N_C-1} rint_{i\leftarrow j}^{mem}$$

Results

- Cycle contention stack does not have LLC1, LLC2 or LLC3 interference components.
- AOCS cycles spent in memory affected by corunner in core2



www.bsc.es

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

PMCs for real-time multicore systems: analysis of the state of the art and initial proposal

Francisco J. Cazorla, Jaume Abella
Javier Jalle, Mikel Fernandez
Leonidas Kosmidis

Luca Fossati (TO)
Marco Zulianello

Software Systems Division & Data Systems Division Final Presentation Days
ESTEC WWednesday 10th of December 2014