Shared Resource Interference

Real-Time Industrial Systems

Marcello Cinque
Giorgio Farina

Roadmap

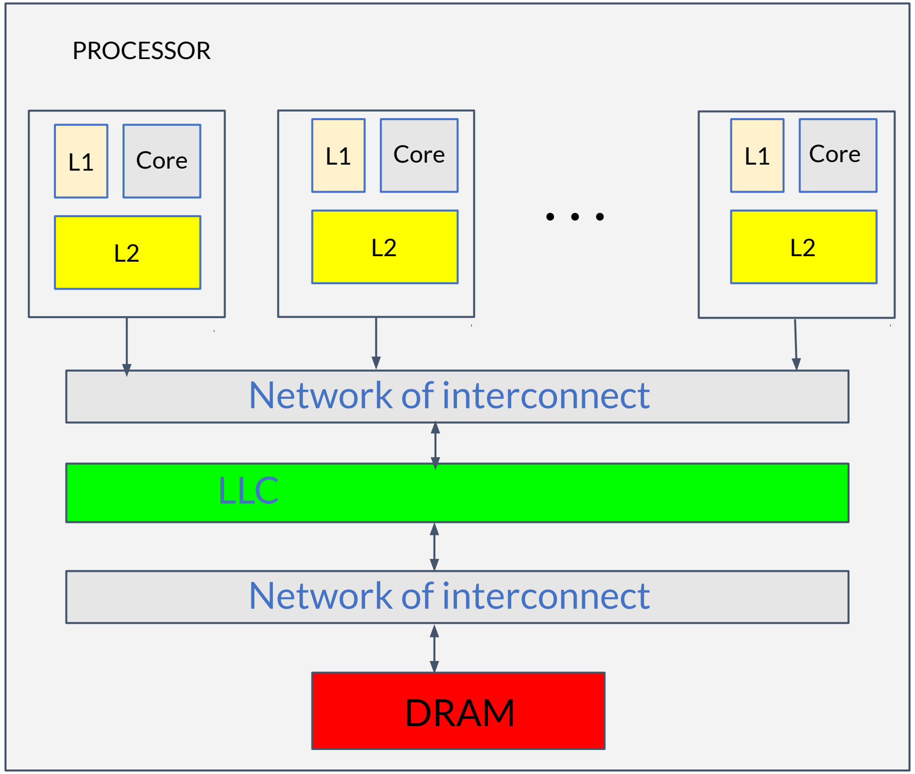
- Shared resources
- Partitioning and scheduling
- Spatial partitioning in the memory hierarchy
- Temporal partitioning in the memory hierarchy
- References:
  - G. Giovani. “A Survey on Cache Management Mechanisms for Real-Time Embedded Systems”, 2015
  - Page coloring https://www.lynx.com/embedded-systems-learning-center/what-is-cache-coloring
  - Execution models
    - R. Pellizzoni. “A predictable execution model for cots-based embedded systems.”
    - F. Boniol. “Deterministic execution model on cots hardware.”
  - Resource reservation (PALLOC)
    - H. Yun. “Palloc: Dram bank-aware memory allocator for performance isolation on multicore platforms.”
  - Interference-Sensitive WCET Analysis
    - Jan Nowotsch. “Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement”
  - Memguard
    - H. Yun. “Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms.”
  - MBA
    - https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-memory-bandwidth-allocation.html

Shared resources

Why Shared resources ?

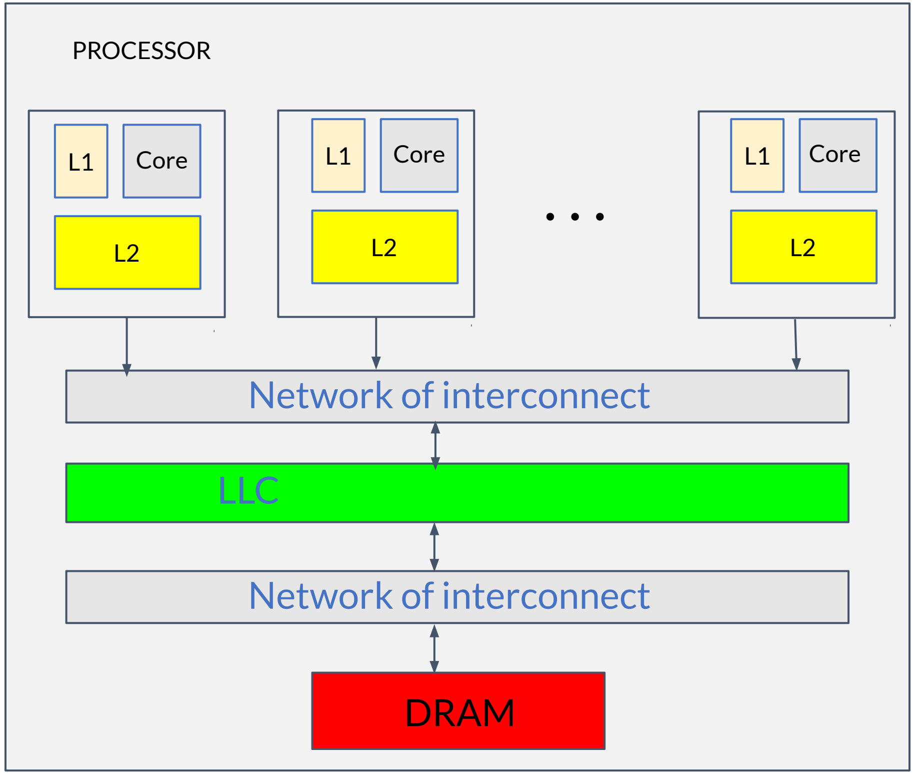
- Fast communication
  - i.e., shared memory
- Maximize the average utilization

- Introduces variability
  - Spatial contention
  - Temporal contention



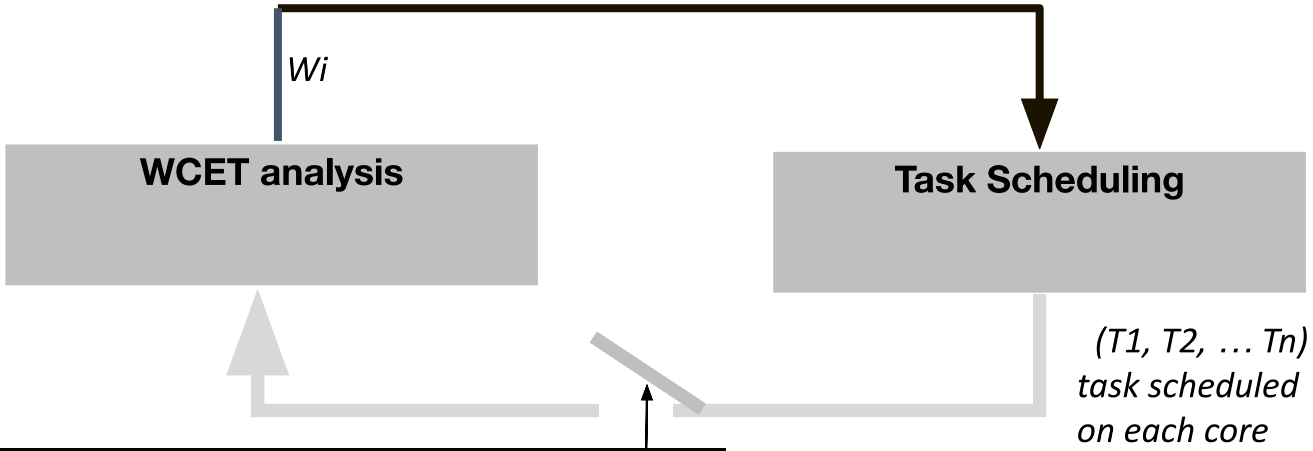
Resource Interference

Interference	Temporal contention	Spatial contention
Intra-core	i.e., Functional unit **access contention** (timing anomalies)	i.e., when a preempting task evicts the preempted task's cached data **(private cache spatial contention)**
Inter-core Tra core diversi	i.e., Memory controller and shared cache **access contention**	i.e., **shared cache spatial contention**



Partitioning and Scheduling

Partitioning concept



Spatial and temporal isolation (*partitioning)*

- Reduce the WCET analysis computational complexity
- Could reduce the average overall system utilization (It requires dynamic scheduling allocation)
- It can protect the critical task from noise neighbors
- It is requested by avionic standards
  - iD&C

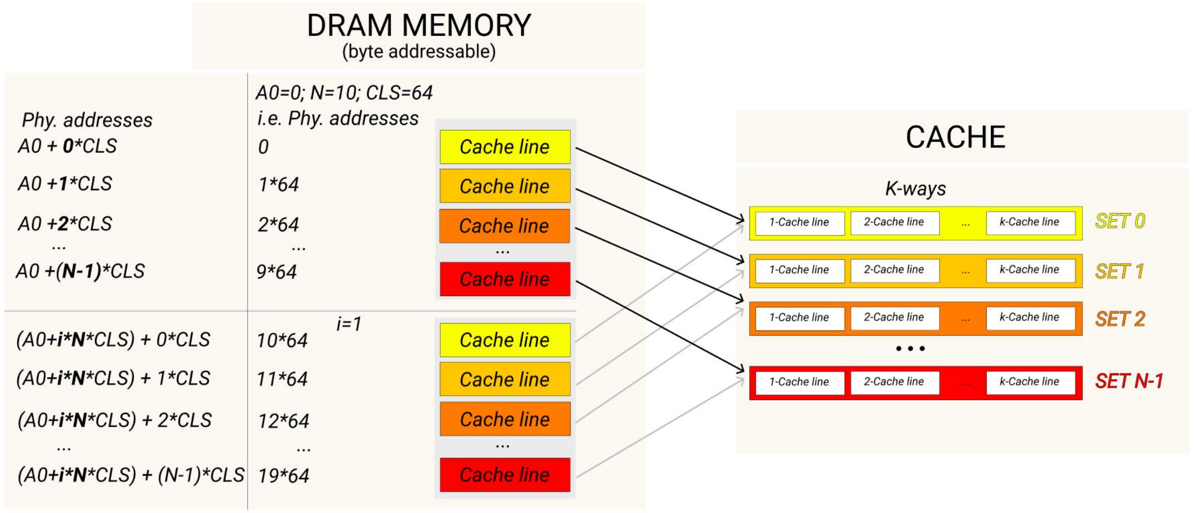
Note:
WCET analysis could be:
- static analysis
- or empirical measurements

Spatial contention in the memory hierarchy

Memory Hierarchy Interference

Cache Contention

- Recap
  - On the modern architectures
    - The caches are Set-associative
    - A shared cache increases
      - the cache space
      - the average cache utilization (being shared)

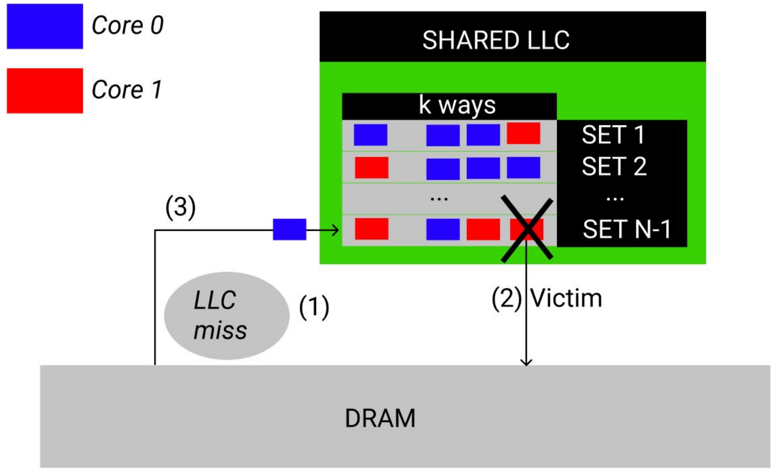


Mapping function: Module function
Address: log2(target)|log2(N)|log2(CLS)
The target A0 is a multiple of N*CLS

Memory Hierarchy Interference

Cache Contention

- i.e., shared Last Level cache interference
  1) Core 0 has an LLC miss
  2) The SET N-1 is full -> so, depending on the replacement policy (i.e., LRU), the "red" cache line is evicted
- A new "blue" line can evict the "red" line in LLC
  - **Increasing the complexity of a precise estimate**
  - **Impacting also the performance in some case** *(**NOTE** when a task working set is bigger than cache space (streaming applications) -> non-temporal-store are better avoiding the cache poisoning)*



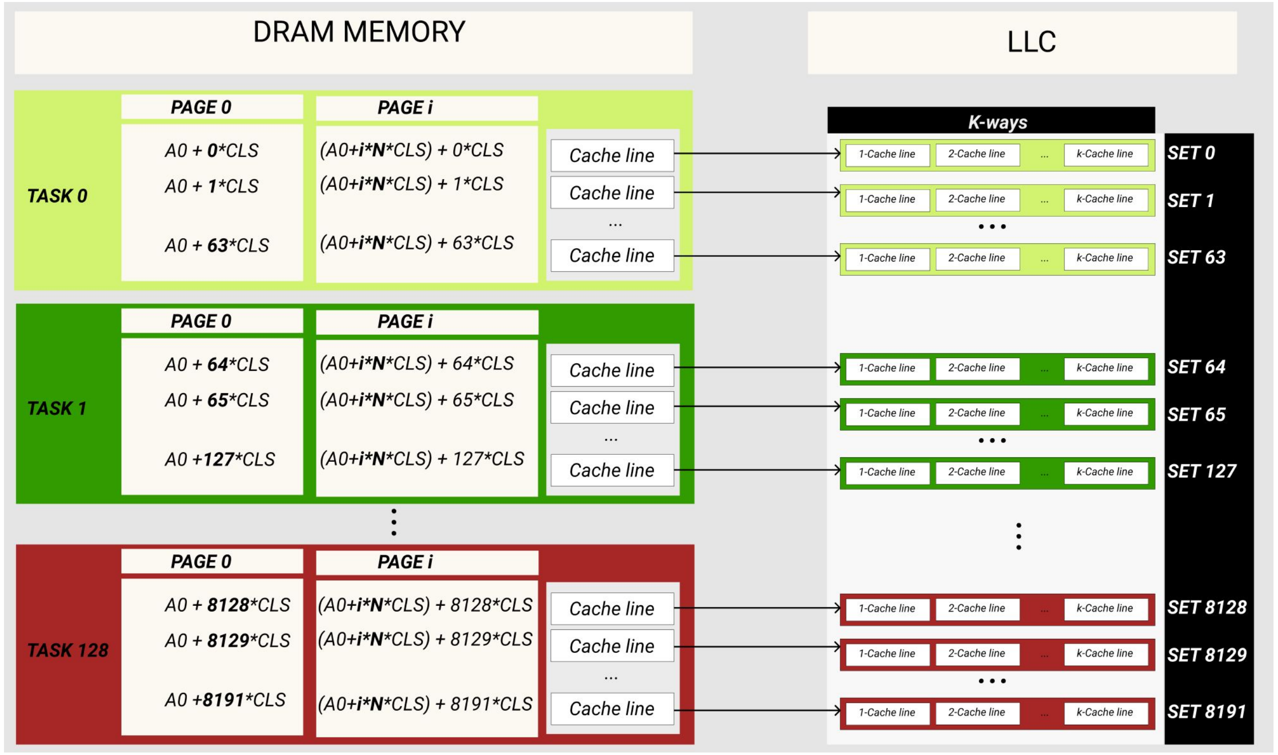
LLC= Last Level Cache

Cache Spatial partitioning

- **Horizontal partitioning (set partitioning)**
  - Cache coloring (or Page coloring) (software solution)
    - OS level
    - Hypervisor level
- **Vertical partitioning (way partitioning) and cache locking**
  - Cache locking (hardware solution)
  - CAT (Intel) (hardware solution)
- **Alternative techniques**
  - Pessimistic approaches
  - From implicit caching -> explicit caching (Scratchpad memory)

Horizontal Partitioning

"coloriamo i set e non le K-ways"



Page coloring

i.e.

Parameters

- *LLC SIZE = 8MB*
- *CACHE LINE SIZE (CLZ) = 64B*
- *# WAYS = 16*
  - *The number of sets (way size)=*

  LLC SIZE/(CLZ#WAYS) = 8192*
- *PAGE SIZE = 4K (equal to 64 CL)* CL= Cache Line

Number of possible partitions (colours):

Way size / (Page size/CLS) =128

Horizontal partitioning

- Page coloring
  - OS perspective considerations:
    - Linus Torvalds https://yarchive.net/comp/linux/cache_coloring.html
    - it artificially limits your page choices, causing problems at multiple levels (also in page allocators and freeing).
      - If you limit the cache space of a “x” factor , also the available memory is reduced of a “x” factor
      - possible memory pressure and average performance degradation
    - memory utilization degradation

  - Coloring at Hypervisor level
    - Jailhouse https://git.hipert.unimore.it/rtes/jailhouse/-/commit/ef4ac0fc80a68132b639440905d415194979d90d
    - Leverage the new hardware virtualization support for guest physical address translation (i.e, Extended Page Table EPT)
      - In this way, the guest OS sees a uniform and contiguous guest physical address space.
      - Note: the added layer of translation increases the access latency

- Advantages
  - Supported by software

- Disadvantages
  - Cache partitioning implies also memory partitioning

Vertical Partitioning and cache locking

Hardware solutions

- Cache allocation technology (CAT)
  - Resource Director technology (to be explored …)
- Cache locking
  - There are two ways to lock a cache content:
    - Atomic instruction (such as Freescale P4040 and P4080 platforms)
      - through an atomic instruction to fetch and lock a given cache line into the cache
    - Lockdown by master (such as Xilinx Zynq-7000 and so on)
      - Each core defines a mask specifying what cache-ways are available

- Advantages
  - Does Not imply memory address space partitioning

- Disadvantages
  - Requires hardware support

Alternative techniques

- Pessimistic approaches
    - disable the shared cache (if it is possible)
    - consider the worst case (each access is a memory access)
- Explicit caching (Scratchpad memory)
    - Implicit caching -> explicit caching
    - Advantages
        - Being explicit, It is not subject to intra-core poisoning
    - Disadvantages
        - Attention to the data coherency
        - External fragmentation (contrary to internal of normal cache)

Memory Hierarchy Interference

Memory contention

- MLP Resource Reservation: leverage the MLP (Memory level parallelism) reducing the memory access latency variability
- PALLOC: Map the parallel memory component (the bank of memory) to the individual partitions (like page colouring) dynamically
  - GitHub: https://github.com/heechul/palloc

- Advantages
  - Reduce the access latency variability

- Disadvantages
  - Reduce the MLP
  - Not resolve the access contention to the previous resources such as the network of interconnect and memory controller

Conclusion

Memory hierarchy spatial partitioning

- The MMU is an important feature for supporting software solution

The **software** cache spatial partitioning increases the OS overhead

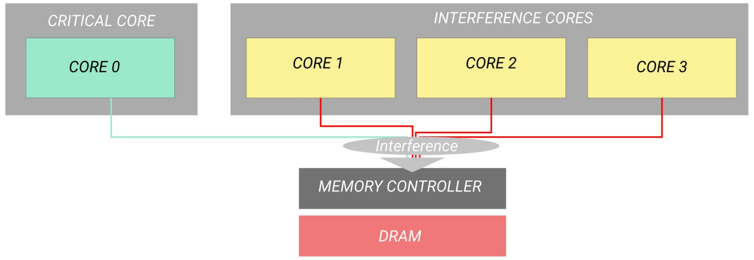
The cache partitioning improves the predictability and defends by the cache poisoning and noise neighbors (also from security attack as the side channel attack…).

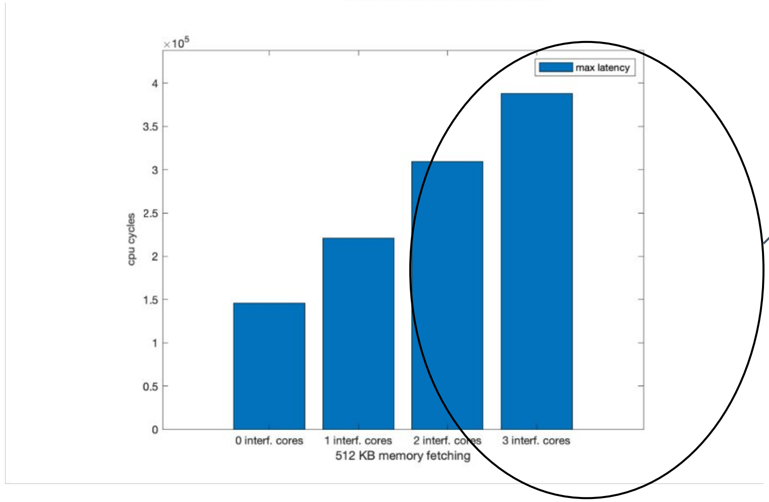
The static spatial partitioning (as any partitioning technique) does not improve the average spatial utilization in a mixed criticality context.

NOTE: we didn't speak about the coherency protocol and the inclusivity (or not) of the LLC. There are other point of spatial interference. i.e., inclusive victim or directory victim.

Temporal partitioning in the memory hierarchy

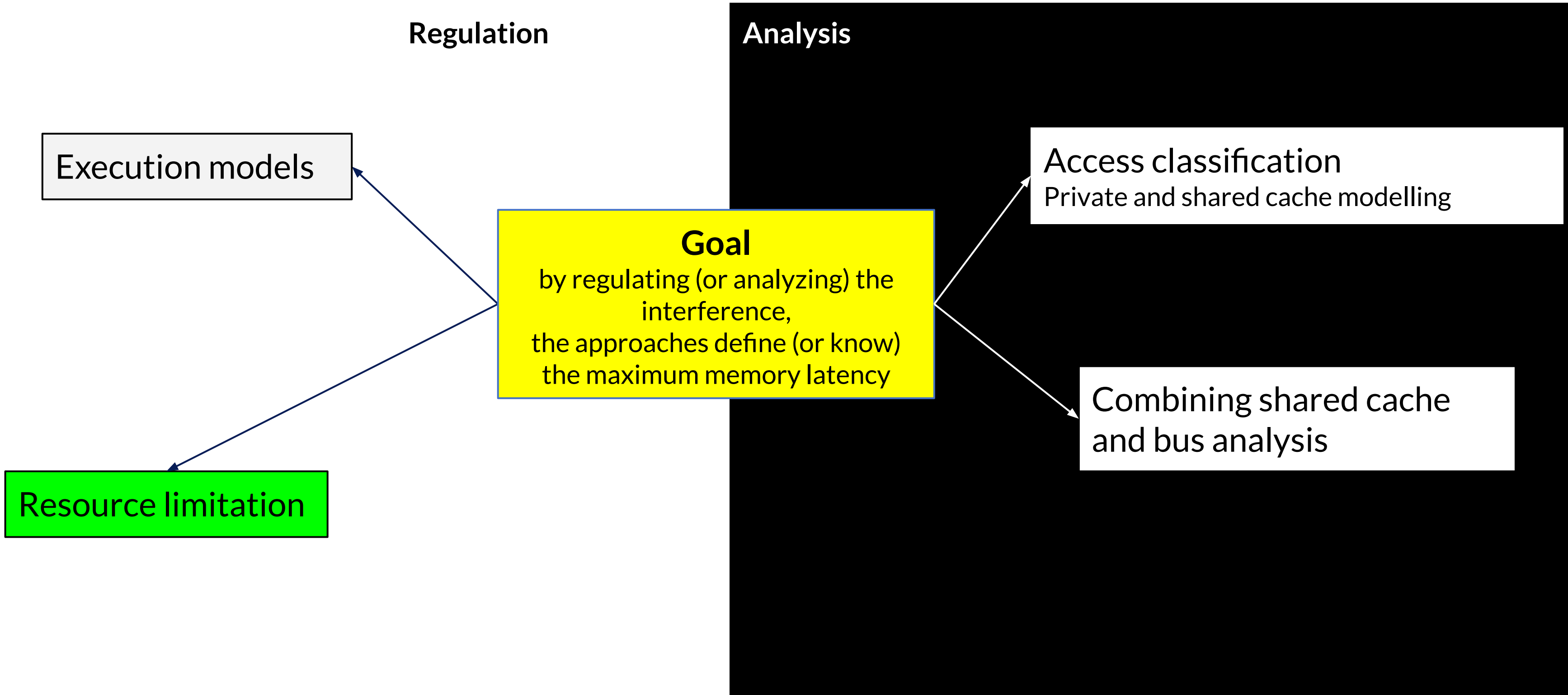
Introduction





- The memory controller access is a well-known critical point in the real-time multi-core systems.
- Memory access latency increases proportionally with the number of interference cores
- The literature proposes several solutions to regulate/analyze the memory accesses, avoiding the maximum interference in the WCET analysis.

Approaches



Goal
by regulating (or analyzing) the interference,
the approaches define (or know) the maximum memory latency

Execution models

- Serialize all the accesses in a particular program phase
- Schedule these access phases by a co-scheduler
  - reproducing a TDMA policy (the TDMA approach is not scalable with the number of cores)
- NOTE: the execution model concept is applied also to handle other interference sources (such as IO)

TDMA= Time Division Multiple Acces

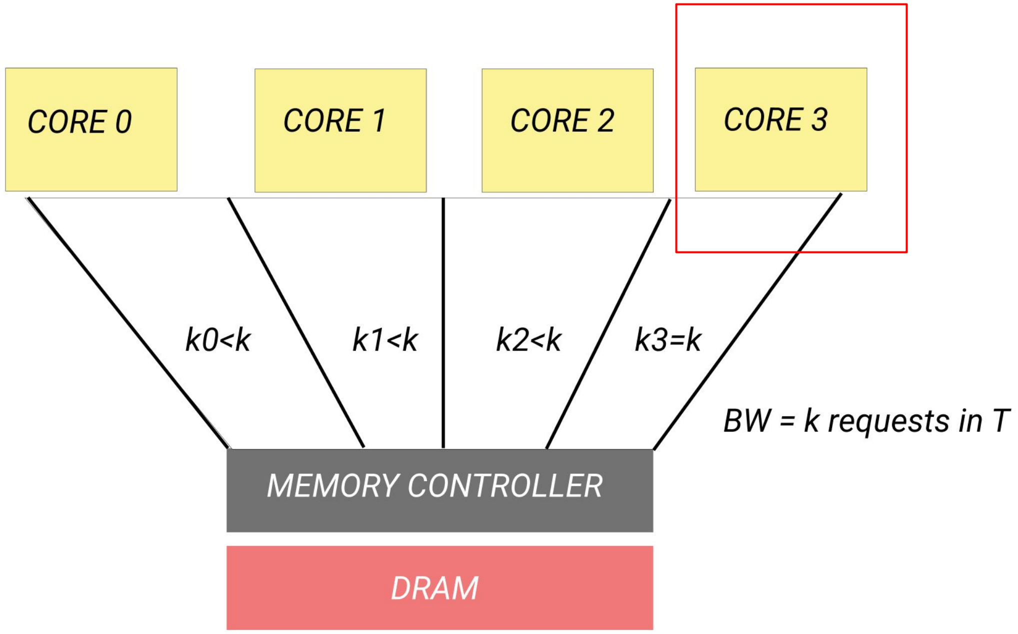
- Advantages
  - a TDMA approach (useful in some cases)

- Disadvantages
  - Require program (or compiler) modification
  - Not suitable for a mixed criticality context
  - Not scalable

Resource limitation

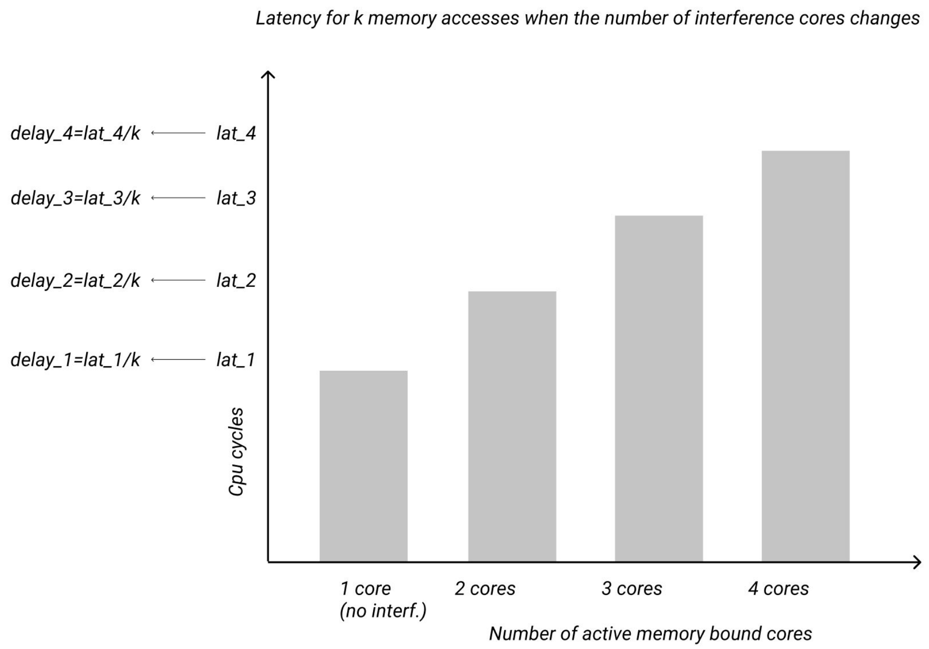
- Defines and limits the number of per-core memory access into a regulation period
- In this way, we can define a maximum memory latency for each core

WCET interference sensitive



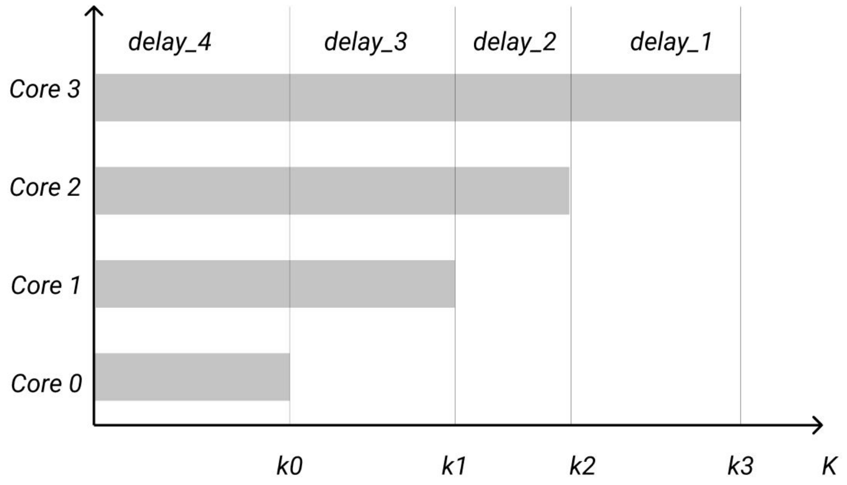
- The core 3 executes k consecutive requests in T (k/T is our available bandwidth)
- Contrarily, the other cores are throttled in T
    - i.e., the core 0 at maximum in T can execute k0<k requests

WCET interference sensitive



- The memory fetching latency changes when the number of interference cores (without throttling) varies
- Assuming a fair arbitration

WCET interference sensitive



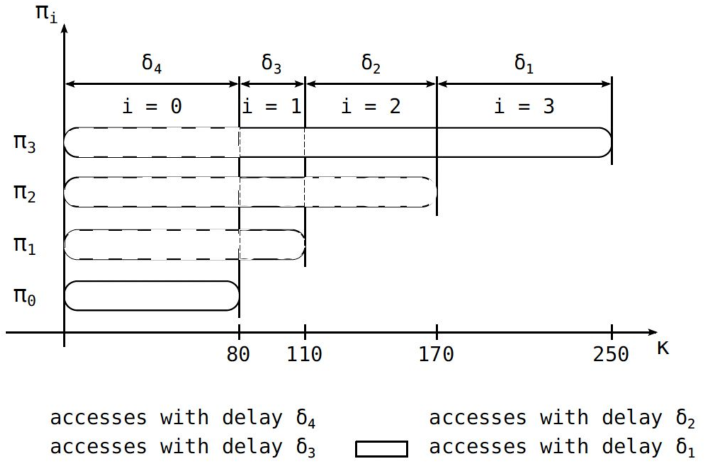
The maximum k3 memory fetching latency for the core 3 is:

- *k0 * delay_4 +*
- *(k1 - k0) * delay_3 +*
- *(k2 - k1) * delay_2 +*
- *(k3 - k2) * delay_1*

The minimum k3 memory fetching latency for the core 3 is:
*(k3*delay_1) without interference*

- Represents the worst case
- Improve the precision of the maximum memory fetching latency:
  - K3*delay_4
- Flexible to capacity load
- Can be applied to other shared resources

WCET interference sensitive

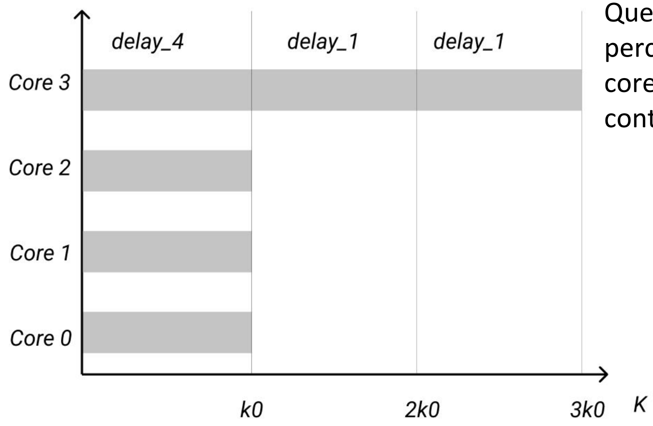


π_i
δ_4
δ_3
δ_2
δ_1
i = 0
i = 1
i = 2
i = 3
π_3
π_2
π_1
π_0
80
110
170
250
κ
accesses with delay δ_4
accesses with delay δ_3
accesses with delay δ_2
accesses with delay δ_1

HP. Hypothesis Interpretation: no two delays delay_i, delay_i+1 exist, where the relative delay_i is greater than delay_i+1, normalising to the number of requesters.

$$\frac{\delta_i}{i} \leq \frac{\delta_{i+1}}{(i+1)} \quad \forall i \in \mathbb{N}^+, 1 \leq i \leq \left|\Pi_{||}\right|$$

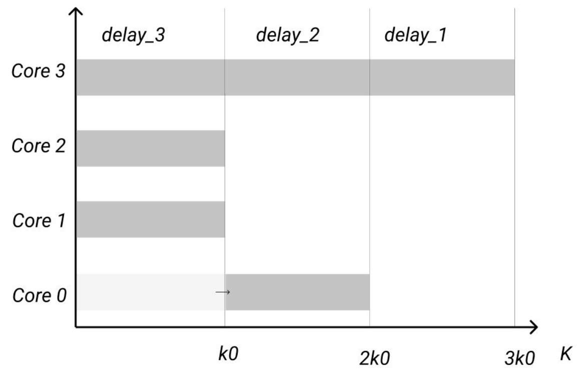
$$\tau_{is}(\pi_x) = \delta_{\left|\Pi_{||}\right|} \cdot \kappa_{\pi_0}^{\phi_k} + \sum_{i=1}^{x} \left(\delta_{\left|\Pi_{||}\right|-i} \cdot \left(\kappa_{\pi_i}^{\phi_k} - \kappa_{\pi_{i-1}}^{\phi_k} \right) \right) + \tau_s(\pi_x)$$



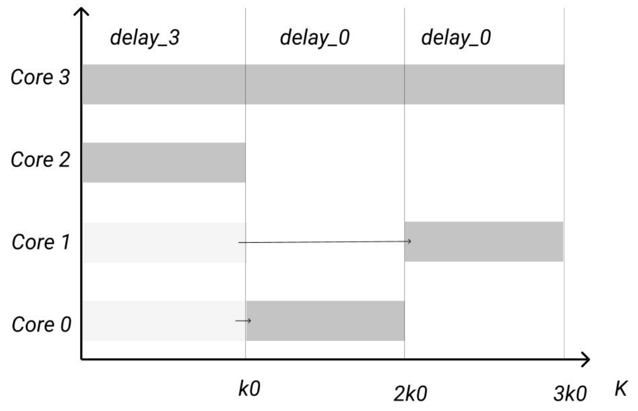
Questo è il worst case perchè abbiamo tutti i core che fanno accessi contemporaneamente

Our formula says that the core 3 worst case latency is:
delay_4*k0+delay_1*(2*k0)

However, we can have also these other cases



latency:
delay_3*k0+delay_2*k0+delay_1*k0



latency:
delay_2*3k0

The correctness of our formula implies that
(delay_4*k0+delay_1*(2*k0))
>= (delay_3*k0+delay_2*k0+delay_1*k0)
and
>= (delay_2*3*k0)

---

if delay_4>=4l; delay_3>=3l; delay_2>=2l; delay_1>=l
the desequalities are satisfied!

---

so delay_4>=4l --> delay_4>=delay_3+delay_3/3 -->
delay_4/4>= delay_3/3 -> Generalizing, we have our hypothesis.
So our hypothesis satisfies our desequalities

Memguard

- Resource limitation approach
- Monitoring policy:
  - Monitors the per-core LLC miss (synchronous memory requests)
- Throttling policy:
  - When a core consumes its memory budget
    - Interrupt the core execution until the new regulation period
- GitHub: https://github.com/heechul/memguard

iD&C= incremental development and certification

- Advantages
  - Flexible
  - Does not require program modification or analysis
  - Enables the iD&C
    - The maximum memory fetching latency does not depend on the individual task in execution on the other cores

- Disadvantages
  - By stopping the core execution, it impacts the private resource utilization
    - It could be a problem for the no-critical task in a mixed criticality context
  - The LLC access interference?
  - The asynchronous write-back?

Asynchronous accesses

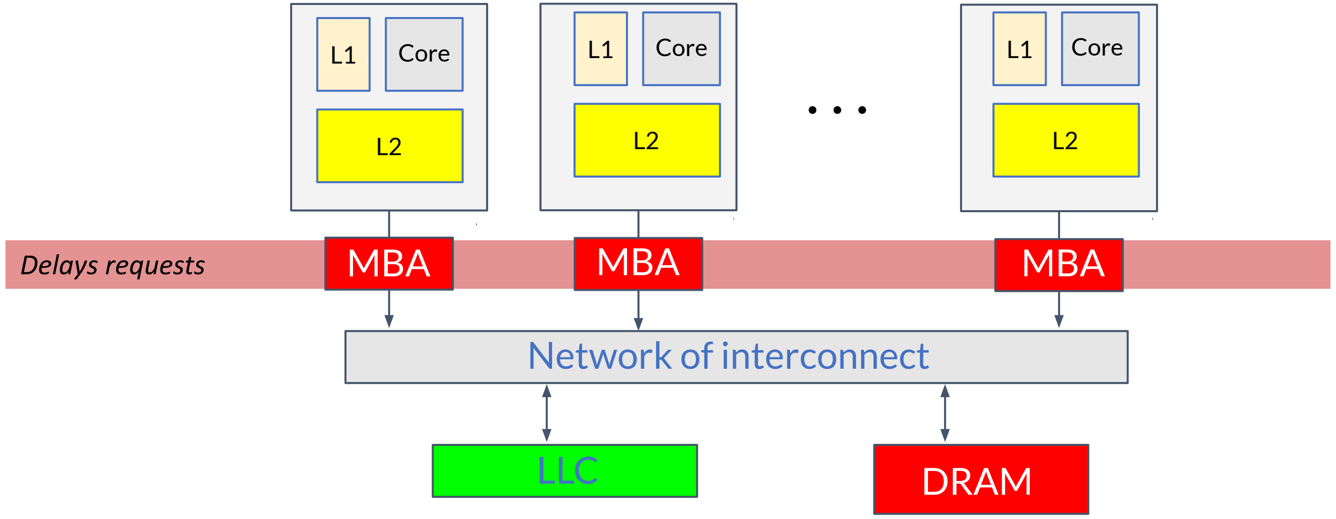
- LLC miss are available as per-core event on the modern architectures
  - Being synchronous
- LLC WB are not available as per-core event
  - Why? The shared cache
    - Spatial interference causes new asynchronous memory accesses
      - The victim cacheline from what core budget should be scaled?

Possible solutions:

- Serialize the write-back by flushing the caches
- Write-through policy rather than write-back policy
- Account the private cache WB rather than LLC WB pessimistically

Intel MBA

Memory Bandwith Associaotion



By these delays, we can apply an indirect throttling policy for the memory access

Intel MBA and Memguard

	Memguard	Intel MBA
flexibility	Software solution	Limited number of delays
Private resource utilization impacting	Stops the execution as throttling policy	The delays are inserted only between requests going versus the shared context
Asynchronous write back	Does not account	Halves the bandwidth when there are L2 (private cache) WB
Overhead and resolution	Handling function overhead (coarse grained regulation period)	Hardware overhead (fine-grained regulation period)

Conclusion

We should design our access interference model considering several factors

- Number of contenders
- Context of criticality
- Our architecture

Examples:

- *The execution models could be suitable in a critical context because, by a TDMA approach, we avoid completely the interference variability*
    - *With a limited number of cores*
- *The resource limitation approach is suitable for a Mixed criticality context (i.e., where no-critical and critical cores share resources) because enables the incremental development and certification (iD&C))*
    - *The tested maximum interference is independent on the individual task in execution on the other cores*

A static temporal partition does not improve the average utilization in a mixed criticality context