HWP: Hardware Support to Reconcile Cache Energy, Complexity, Performance and WCET Estimates in Multicore Real-Time Systems

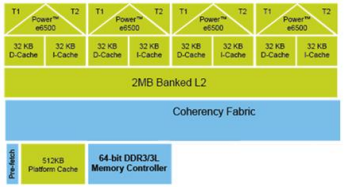
Pedro Benedicte, Carles Hernandez, Jaume Abella, Francisco J. Cazorla

Euromicro Conference on Real-Time Systems

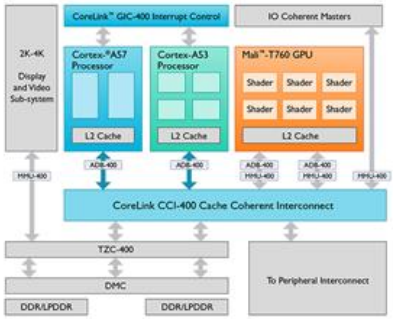
July 4th
Barcelona, Spain

Performance in Real-Time Systems

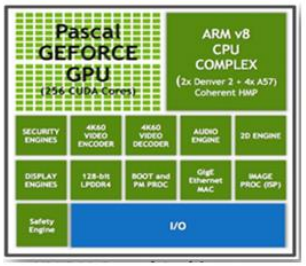
- Future, more complex features require an increase in **guaranteed performance**
- COTS hardware used in HPC/commodity domain offers higher performance
- Common features:
  - Multicores
  - Caches



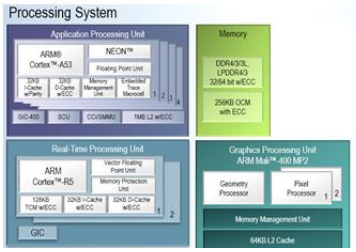
NXP T2080
(avionics/rail)



SnapDragon
(auto)



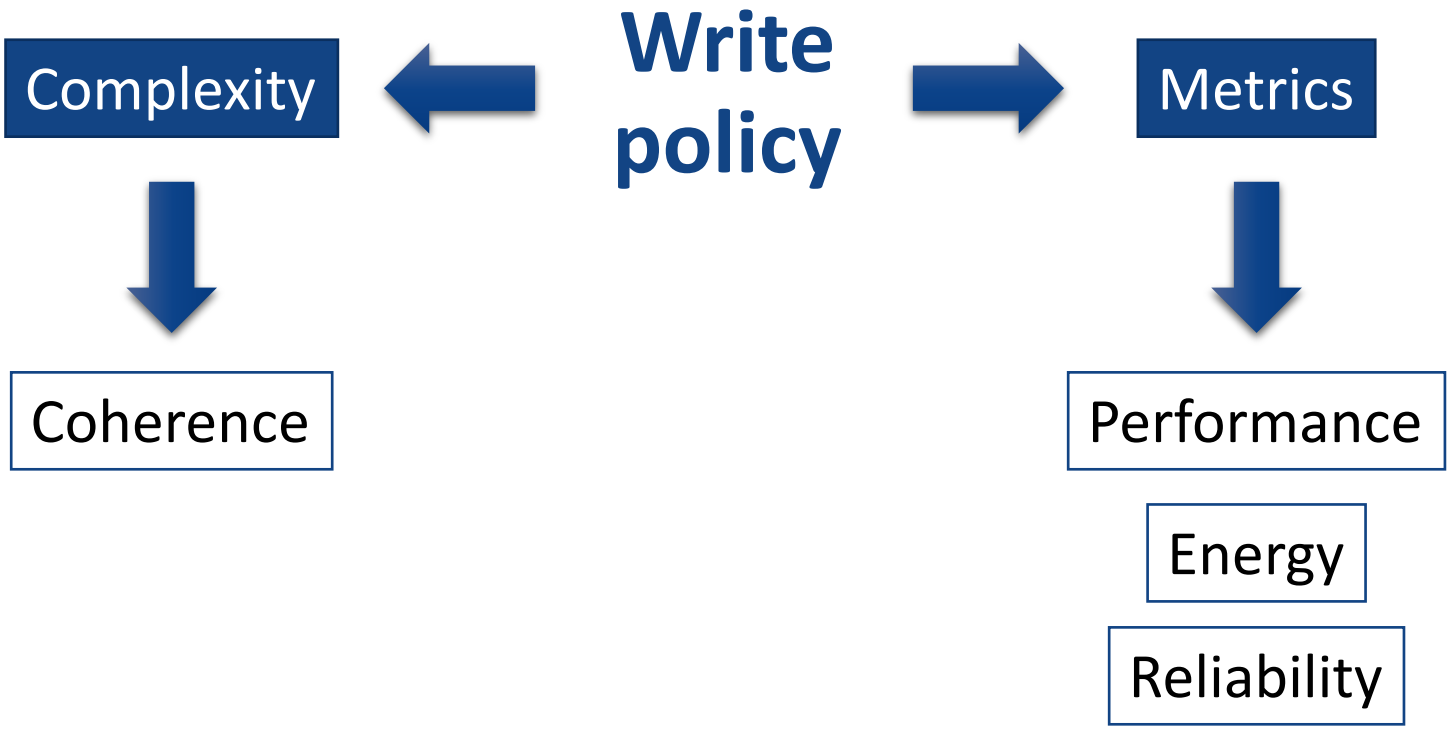
NVIDIA Pascal
(auto)



Zynq UltraScale+ EC
(space/auto)

- We focus on multicore with multilevel caches (MLC)

At the heart of MLC write policy

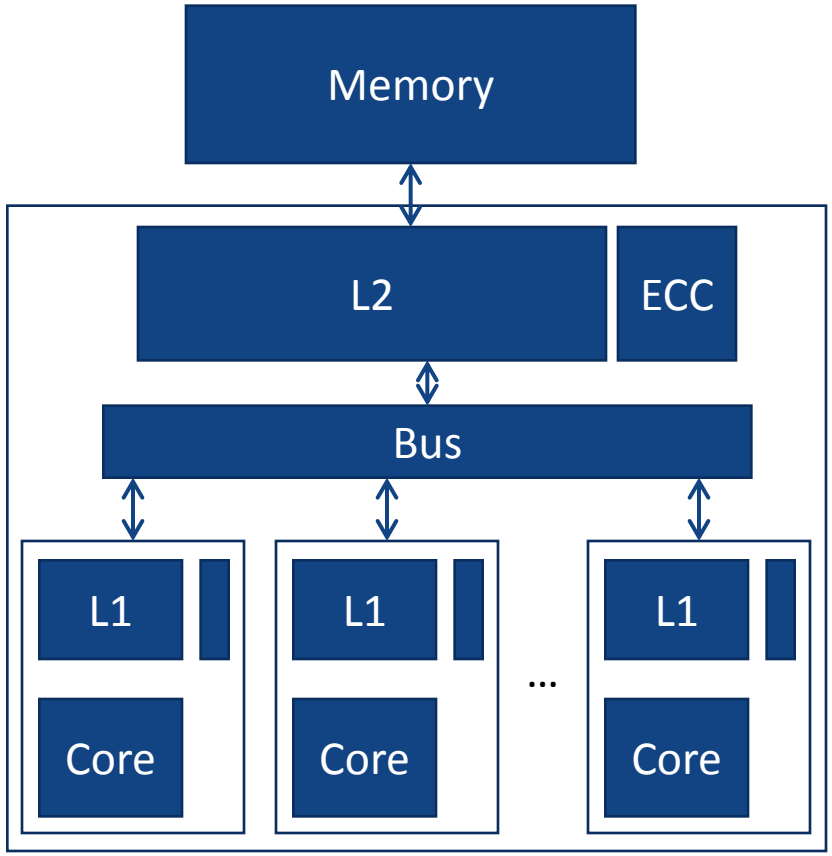


Contributions

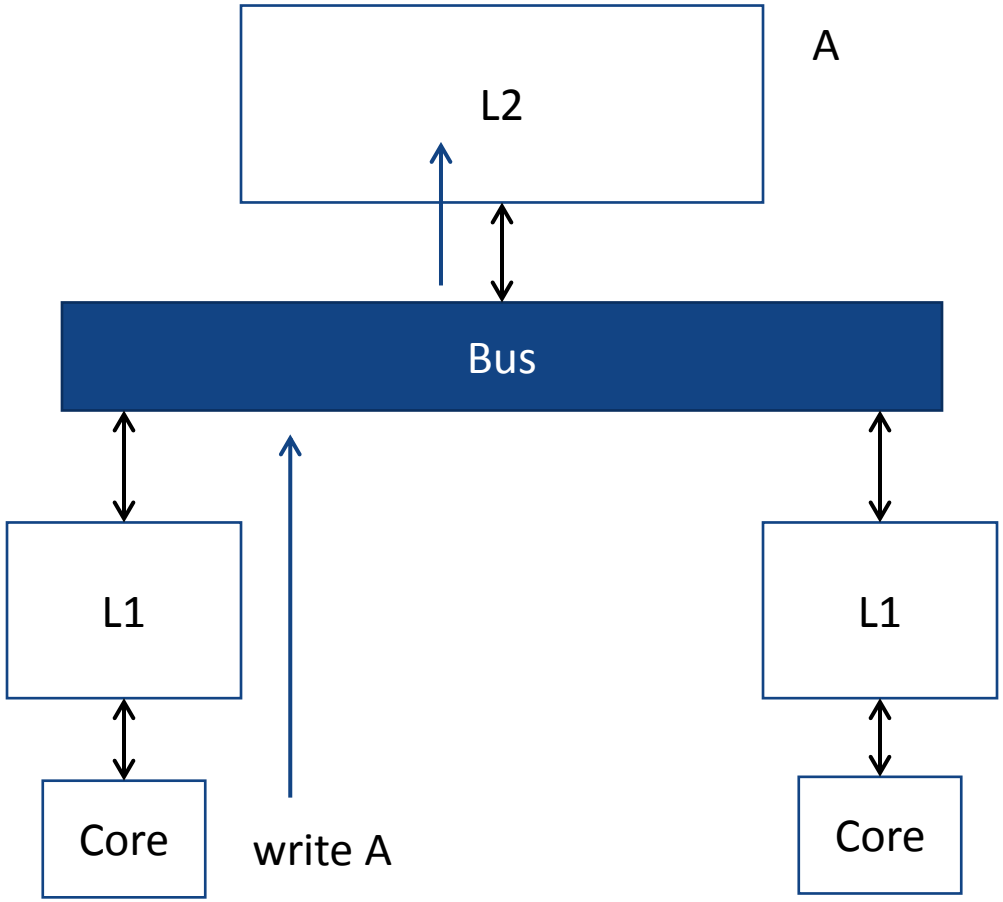
- Analysis of most used policies
  - Write-Through (WT)
  - Write-Back (WB)
  - Write policies used in commercial processors
- Proposal **HWP: Hybrid Write Policy**
  - Try to take the best of both policies
- Evaluation
  - Guaranteed Performance
  - Energy
  - Reliability
  - Coherence complexity

Assumptions

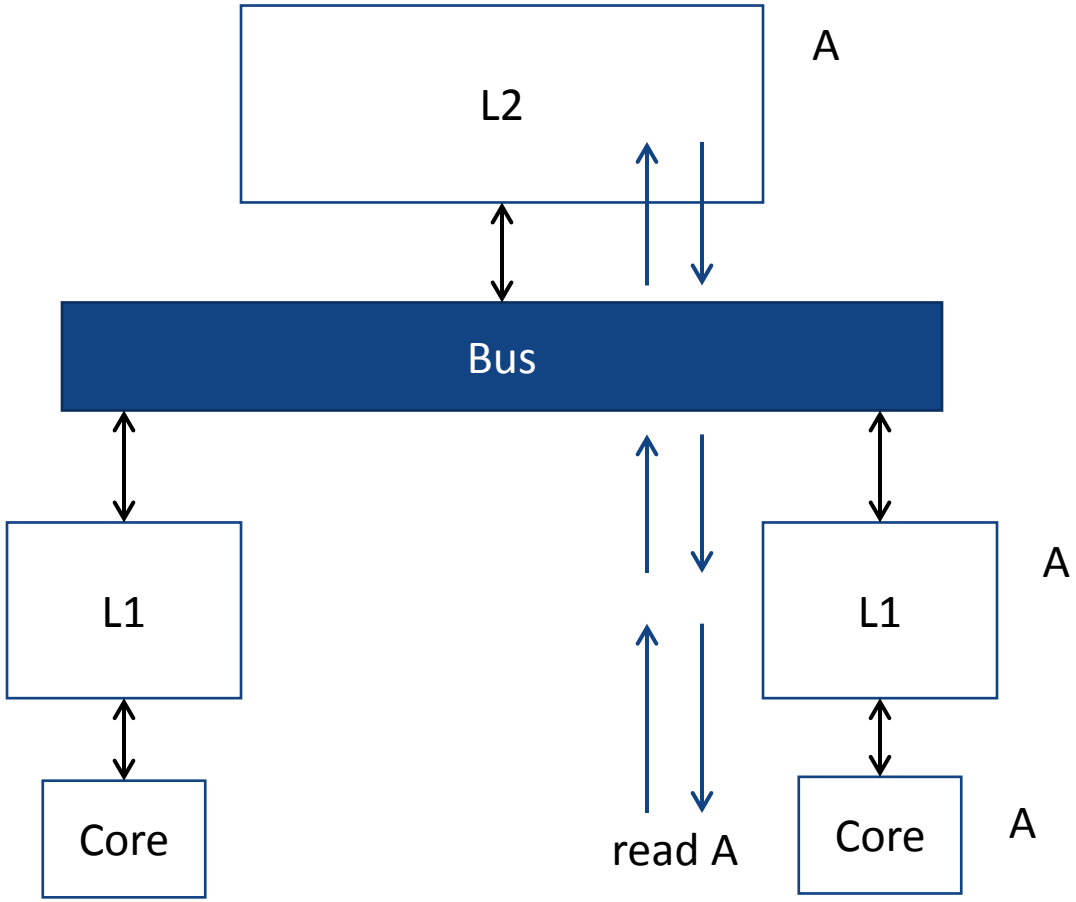
- Multi-core system
  - Private level of cache
  - Shared level of cache
  - Bus to connect the different cores
- Reliability
  - Parity when no correction needed
  - SECDED otherwise
- Coherence
  - Snooping based protocol
    - Good with a moderate number of cores
    - Also can be applied to directory based
  - We assume write-invalidate (MESI)



Write-Through

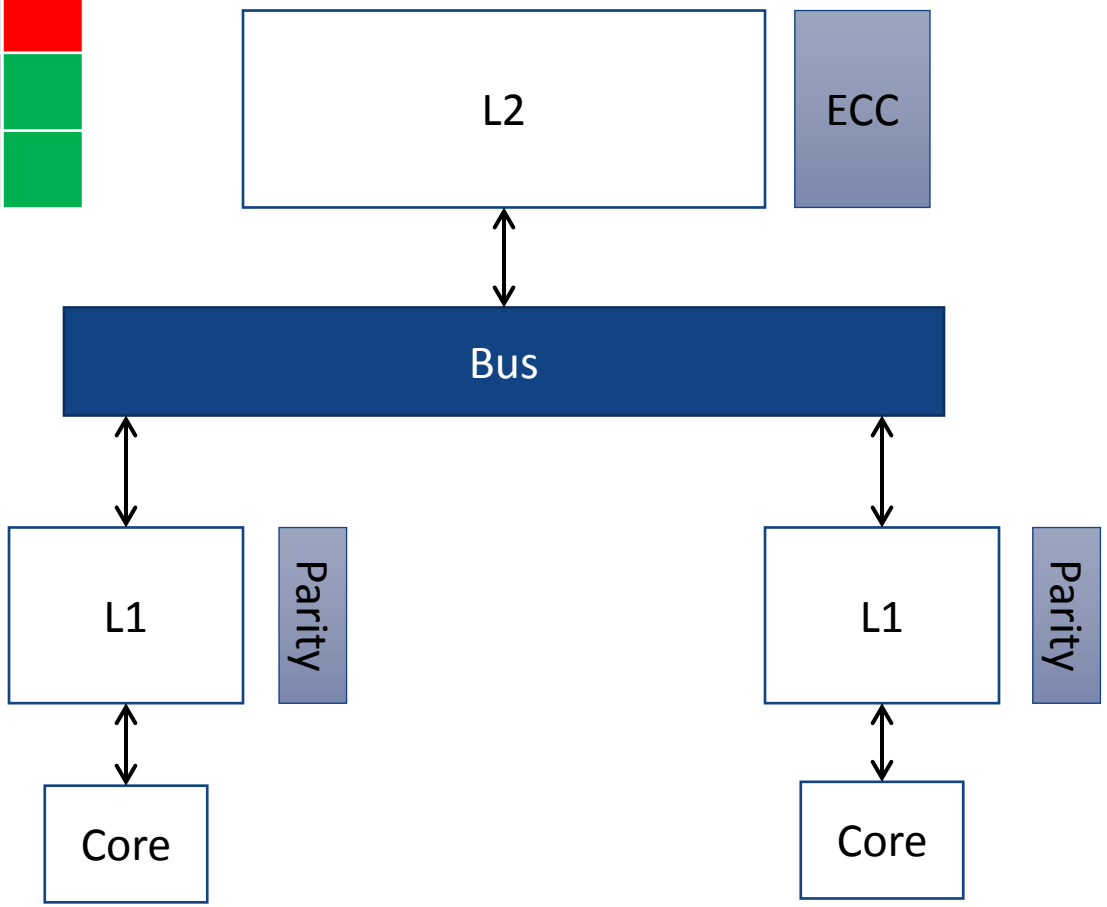


Write-Through



Metric	
Performance	
Energy	
Coherence simplicity	
Reliability cost	

Write-Through



Shared bus writes

- Each write is sent to the bus
  - Store takes *k* bus cycles
  - Bus admits 1/*k* accesses per cycle without saturation

  - 4 cores accessing
  - Bus admits 1/(4·k)

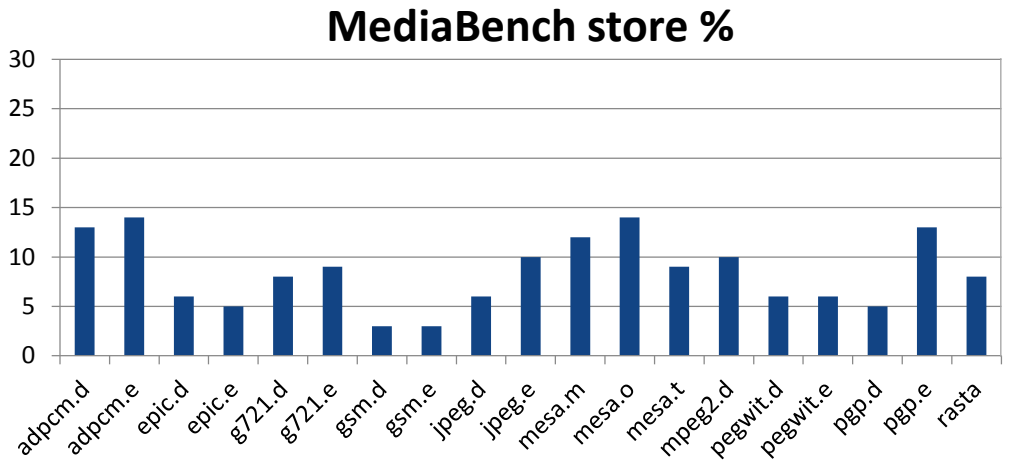
- WT increases load on bus with writes

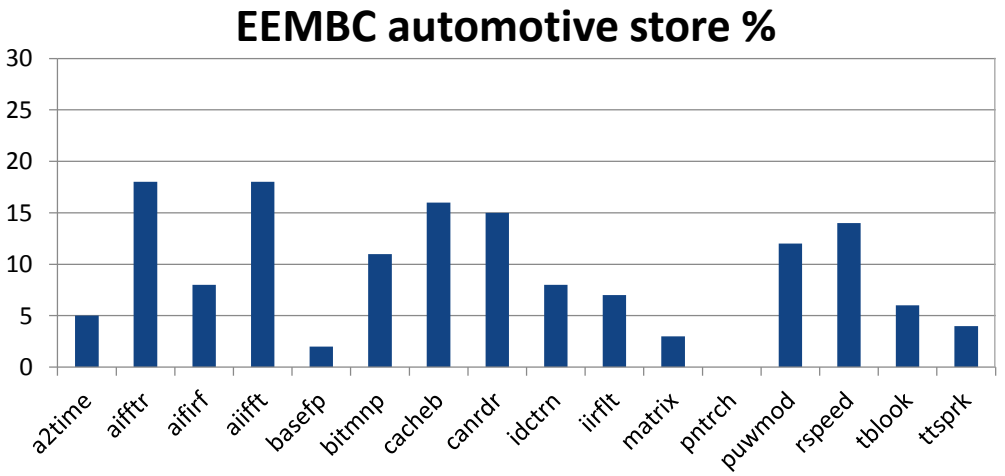
Store percentage in real-time applications

- 9% stores on average
  - Data-intensive real-time applications have a **higher** percentage of **memory operations**
- 4 cores: 36% stores
- If store takes > 3 cycles

↓

Bus saturated





WT: reliability and coherence complexity

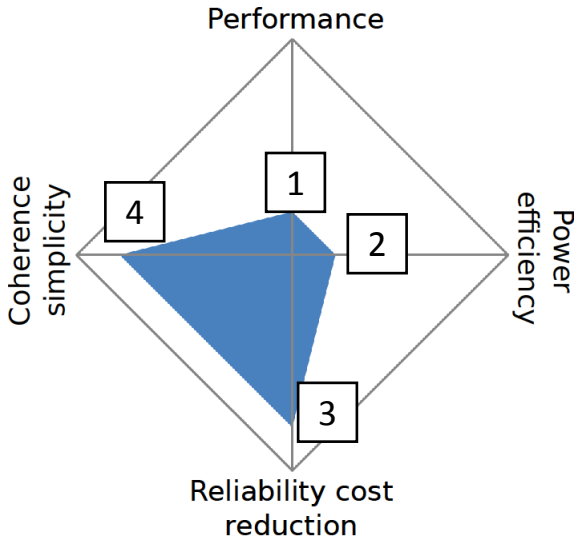
- Reliability:
  - dL1 does not keep dirty data
  - No need to correct data in dL1
    - Just detect error and request to L2
  - Parity in dL1
    - 1,6% overhead

  - Data in L2 is always updated
  - SECDED in L2
    - 12,5% overhead

- Coherence:
  - Data is always in L2, no dirty state
  - A simple valid/invalid protocol is enough

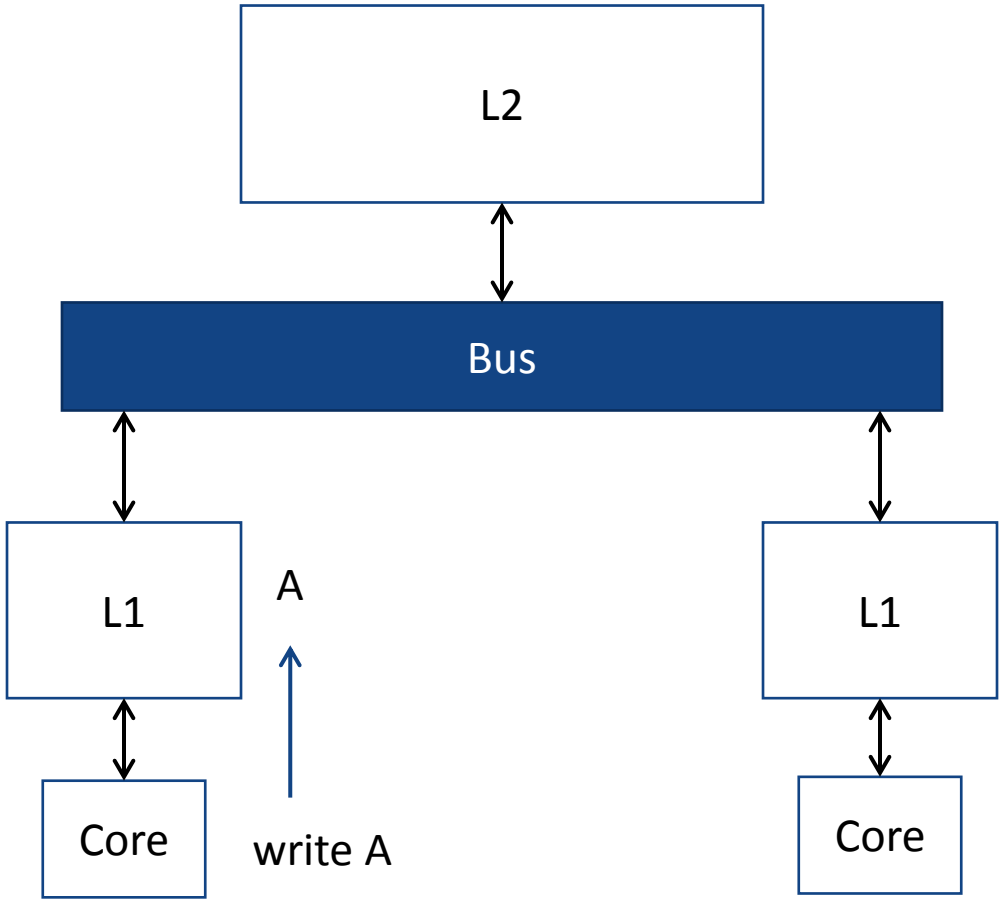
Write-though: summary

1. Stores to bus can create contention and affect guaranteed performance
2. More accesses to bus and L2 increase energy consumption
3. Only requires parity in L1
4. Simple coherence protocol

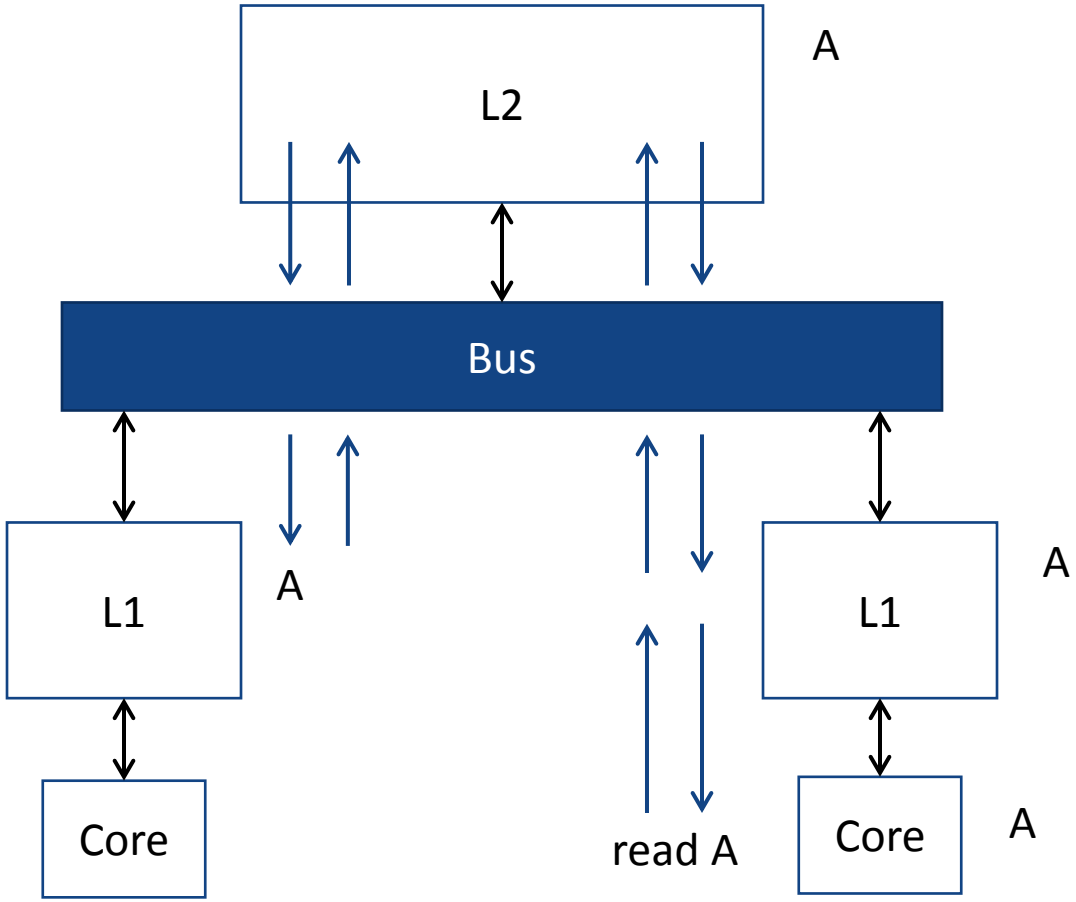


(higher is better)

Write-Back

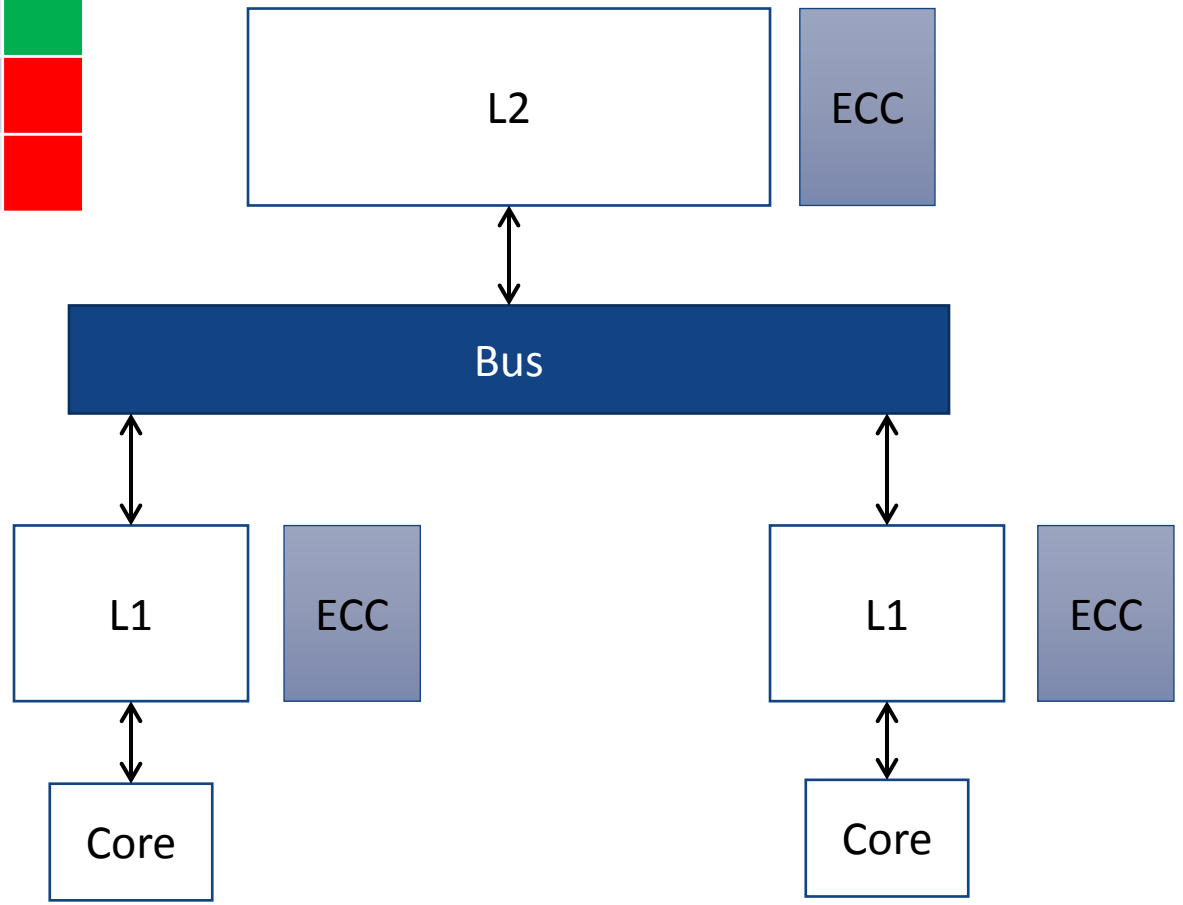


Write-Back



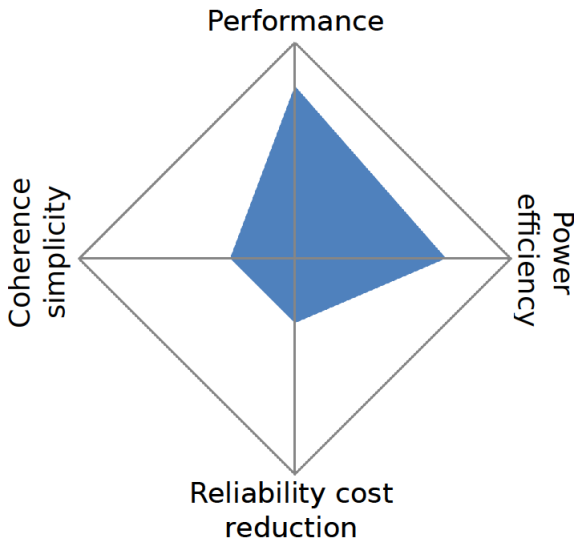
Write-Back

Metric	
Performance	
Energy	
Coherence simplicity	
Reliability cost	



Write-back: summary

- Reduced pressure on bus improves guaranteed performance and energy consumption
- ECC (SECDED) is required for private caches
  - There can be dirty data in L1
- Increase in coherence protocol complexity
  - Due to private dirty lines tracking

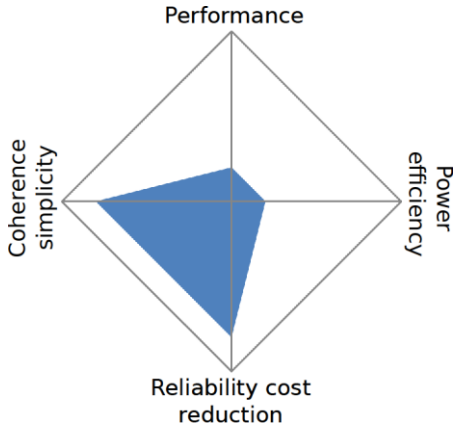


Write Policies in Commercial Architectures

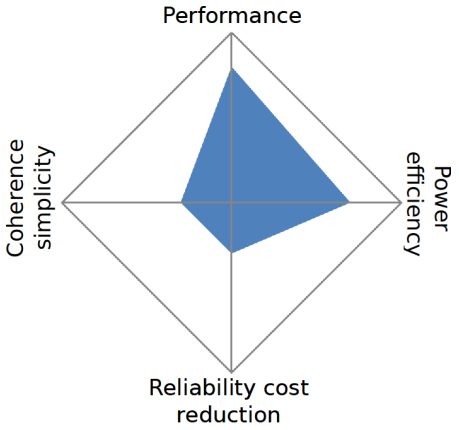
Processor	Cores	Frequency	L1 WT?	L1 WB?
ARM Cortex R5	1-2	160MHz	Yes, ECC/parity	Yes, ECC/parity
ARM Cortex M7	1-2	200MHz	Yes, ECC	Yes, ECC
Freescale PowerQUICC	1	250MHz	Yes, ECC	Yes, parity
Freescale P4080	8	1,5GHz	No	Yes, ECC
Cobham LEON 3	2	100MHz	Yes, parity	No
Cobham LEON 4	4	150MHz	Yes, parity	No

- There is a mixture of WT/WB implementations
  - No obvious solution
- Both solutions can be appropriate depending on the requirements

WT and WB comparison



Write-through



Write-back

- Each policy has pros and cons
- We want to get the best of each policy

↓

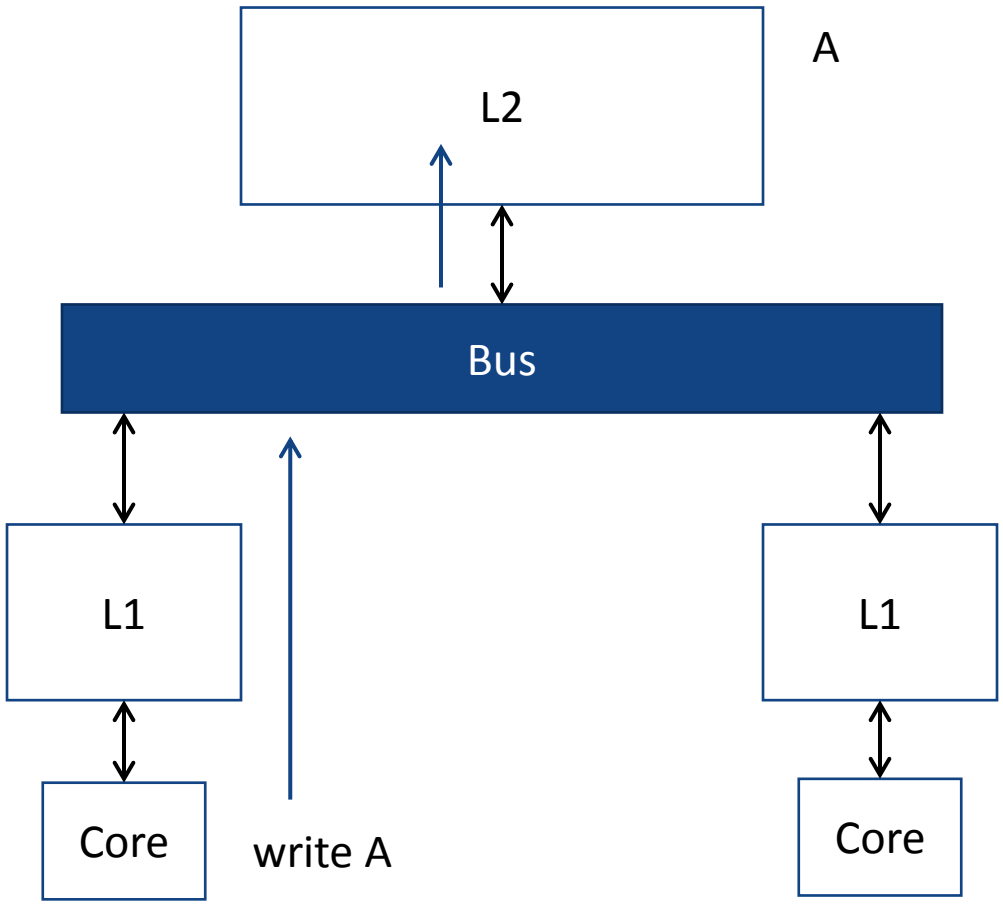
HWP

Hybrid Write Policy: main idea

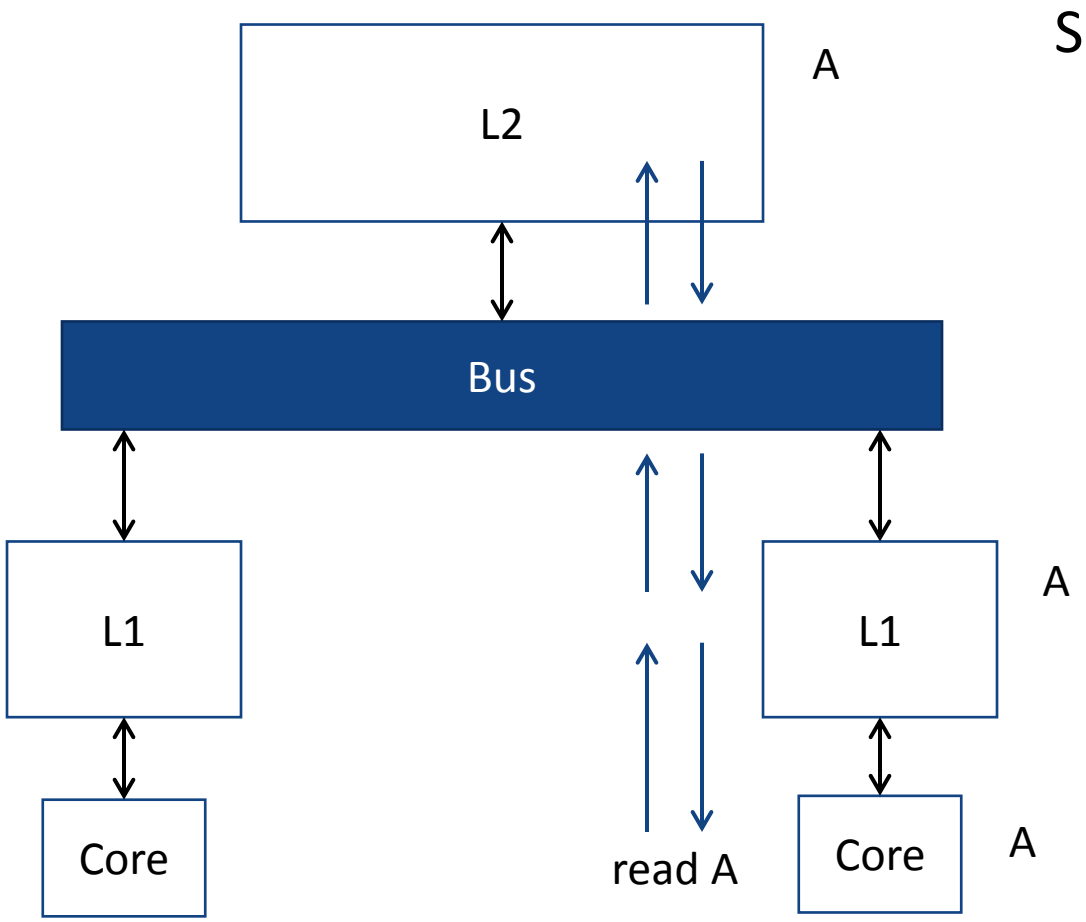
- Observations:
  - Coherence complex with WB because shared cache lines accessed may be dirty in local L1 caches
  - Private data is unaffected by cache coherence
  - A significant percentage of data is only accessed by one processor (even in parallel applications), so no coherence management is needed
- Based on these observations, we propose HWP:
  - Shared data is managed like in WT cache
  - Private data is managed like in WB caches
- Elements to consider:
  - Classify data as private/shared
  - Implementation (cost, complexity…)

Hybrid Write Policy

Shared data

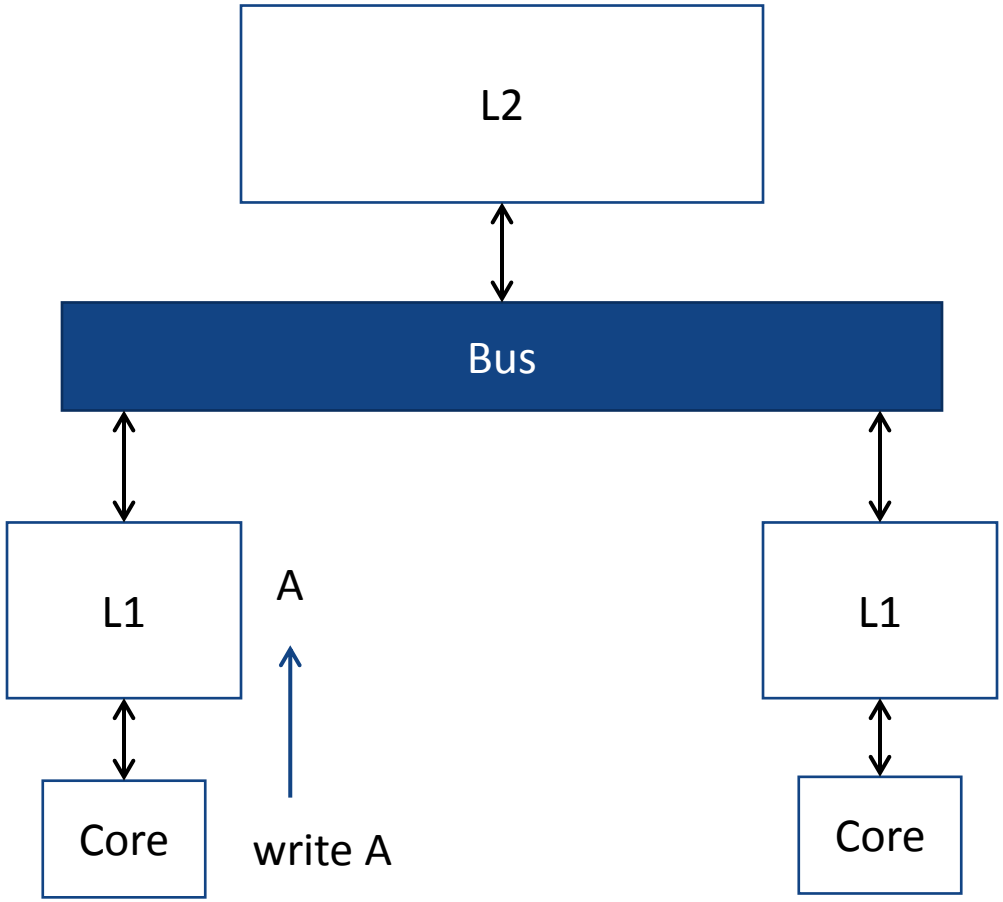


Hybrid Write Policy



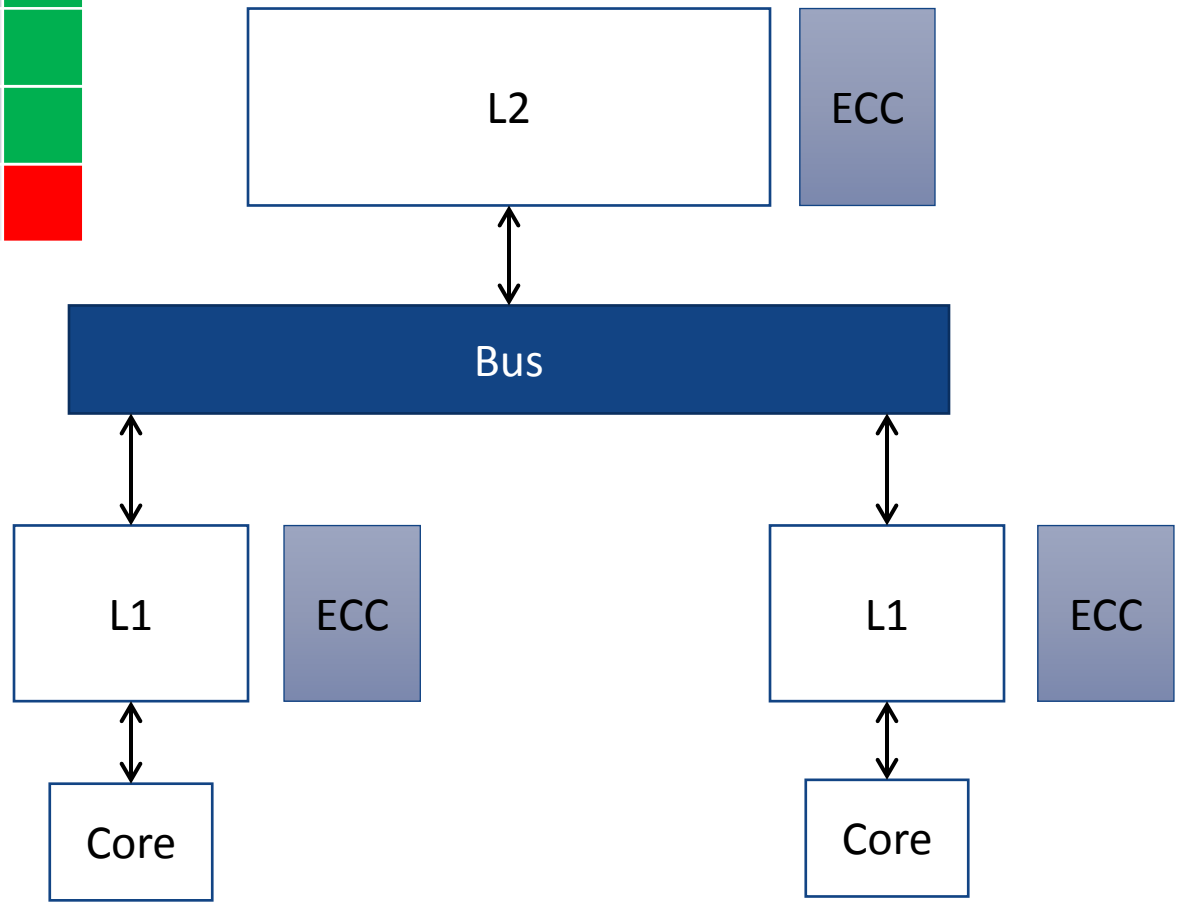
Hybrid Write Policy

Private data



Hybrid Write Policy

Metric	
Performance	
Energy	
Coherence simplicity	
Reliability cost	



Private/Shared data classification

- The hardware needs to know if data is shared or private

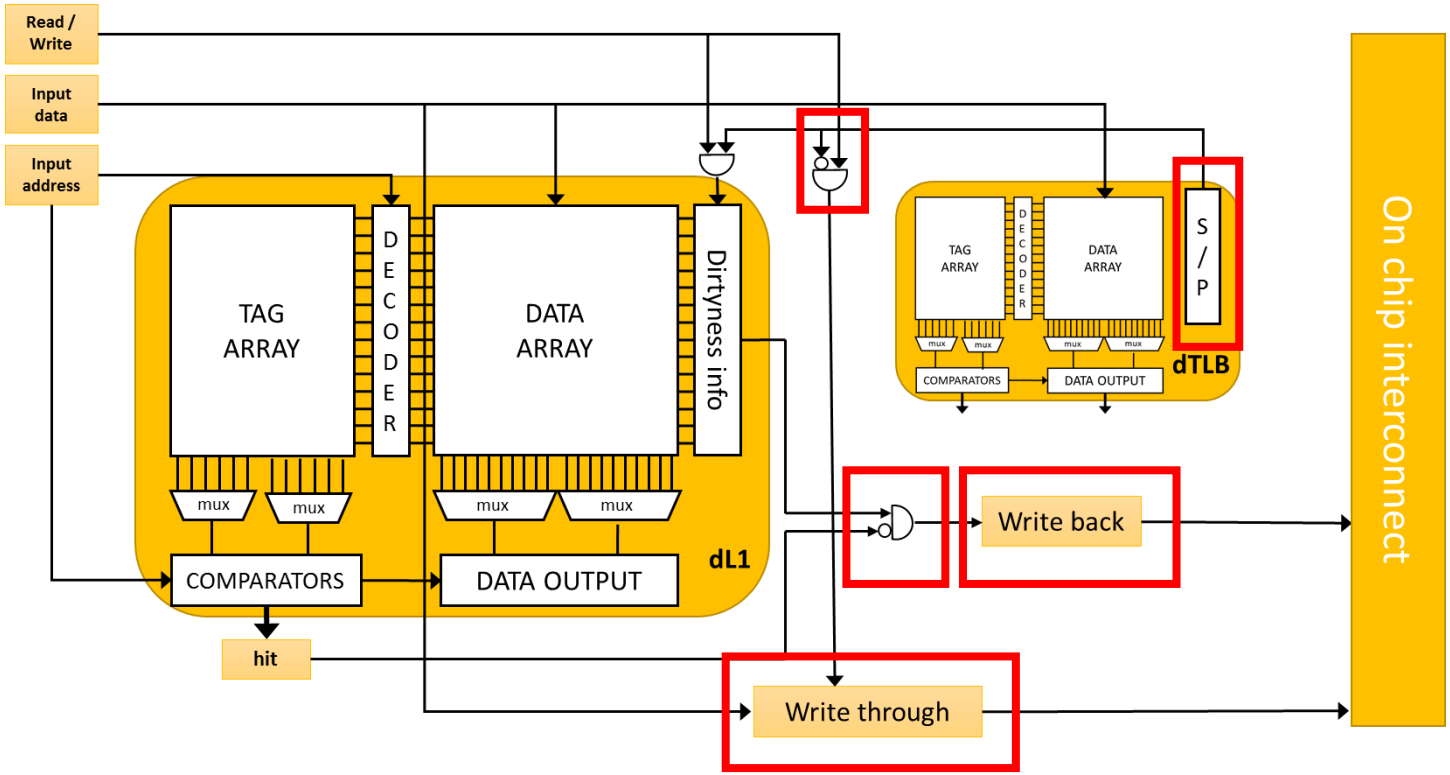
- Page granularity is optimal for OS
- If any data in a page is shared, the page is classified as shared

- Techniques already exist in both OS (Linux) and real hardware platform (LEON3)

- Possible techniques:
  - Dynamic classification
    - Predictability issues in RTS
  - Software address partitioning
    - We assume this solution

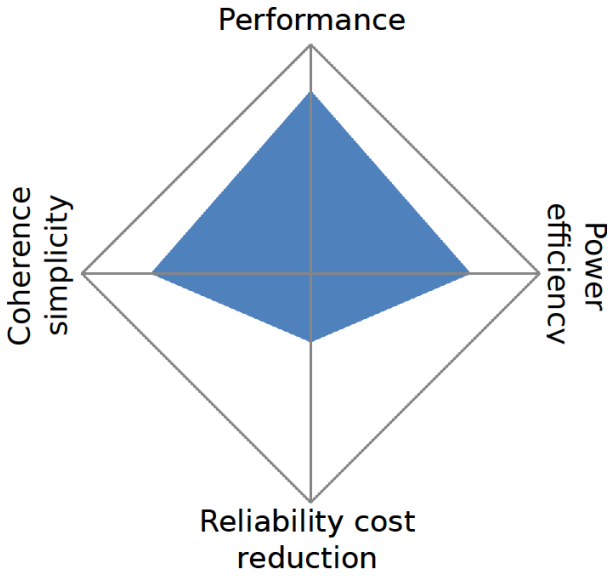
Implementation

- Small hardware modifications

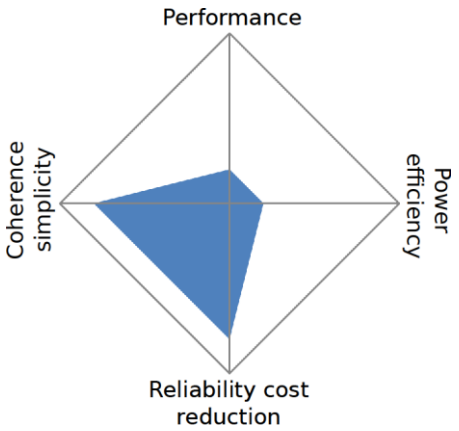


HWP: summary

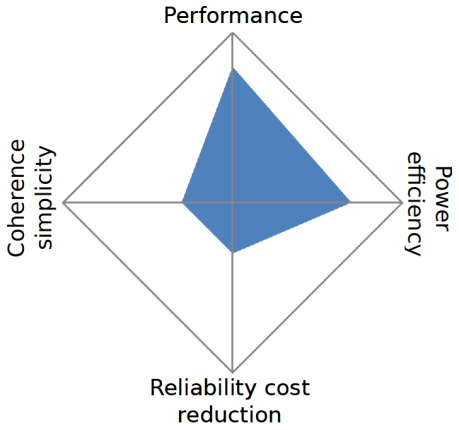
- Guaranteed performance
  - Accesses to bus are limited to shared data
- Energy consumption of bus and L2 also reduced
- Reliability
  - Sensitive data could be marked as shared so is always in L2
  - For critical applications, SECDED needed, private data can be in L1 and not in L2
- Coherence
  - Same coherence complexity as WT



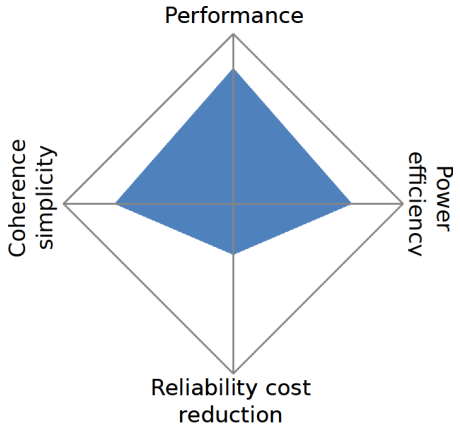
WT, WB and HWT comparison



Write-through



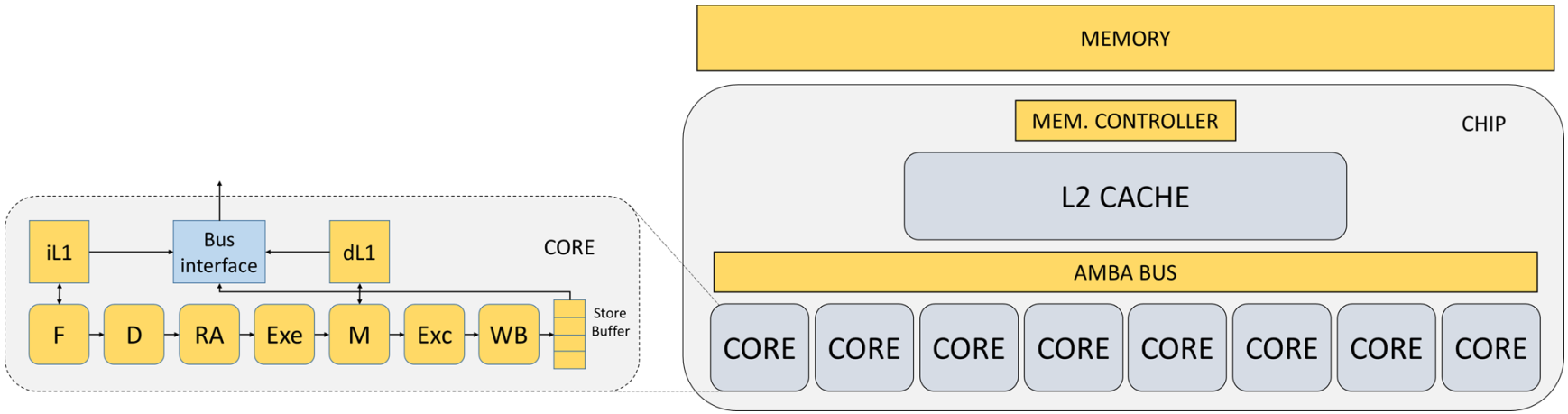
Write-back



Hybrid Write Policy

Evaluation: Setup

- SoCLib simulator for cycles
- CACTI for energy usage
- Architecture based on NGMP
  - With 8 cores instead of 4
- Private iL1 and dL1, shared L2
- Benchmarks:
  - EEMBC automotive, MediaBench



Methodology

- 4 different mixes from single thread benchmarks
- Suppose different percentages of shared data to evaluate the different scenarios
- Model for bus contention [1]
  - Uses PMC to count the type of the competing cores' accesses
  - With this model we obtain partially time composable WCET estimates
  - To summarize, the model takes into consideration the worst possible accesses the other cores DO make

$$\Delta_{b \to a}^{cont} = \sum_{t \in \mathcal{T}} min(n_a, n_b^t) \times lat^t$$

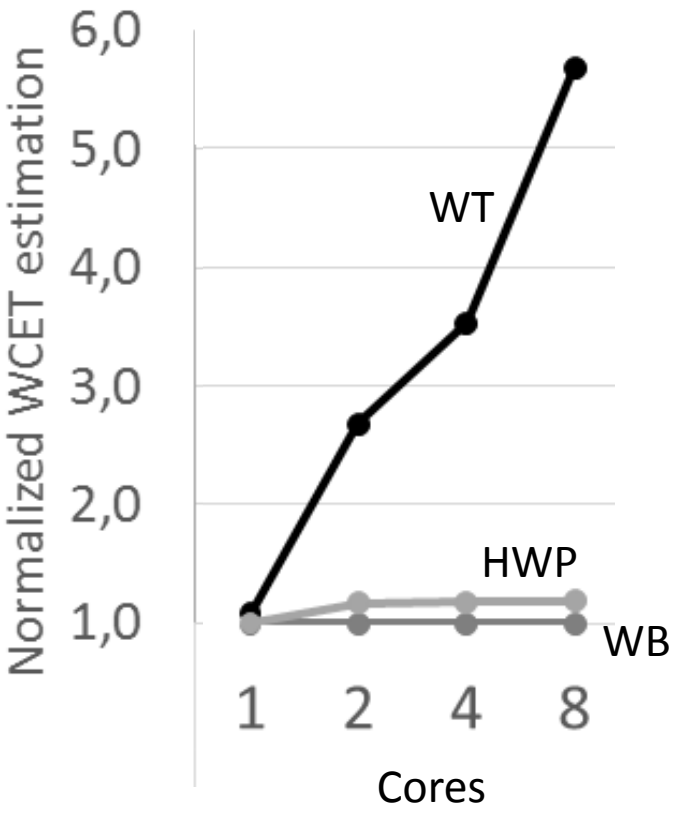
  - Task : 100 accesses to bus Other tasks: 50 accesses to bus
  - The model takes into account only the 50 potential interferences
  - More tight WCET estimates

[1] J. Jalle et al. Bounding resource contention interference in the next-generation microprocessor (NGMP)

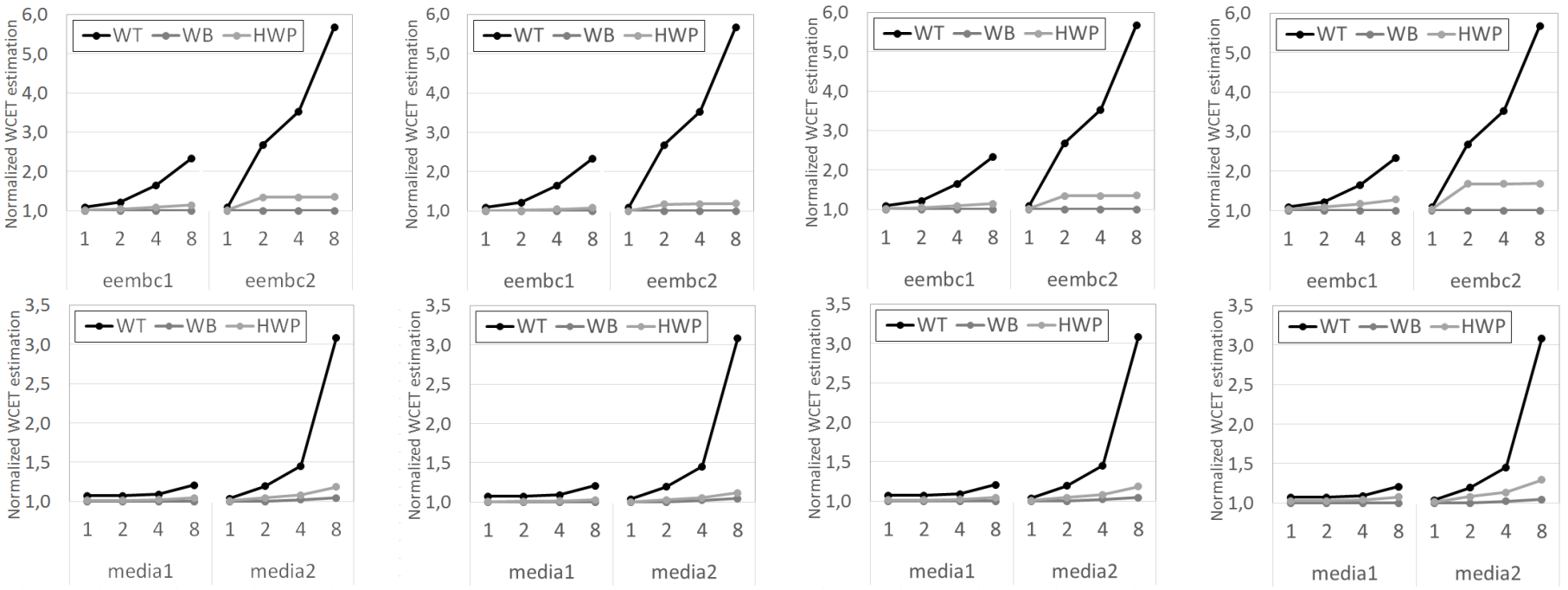
Guaranteed performance

- Normalized WCET bus contention
- 10% of data is shared

- WT does not scale well with the number of cores
- HWP scales similar to WB
- Some degradation due to shared accesses



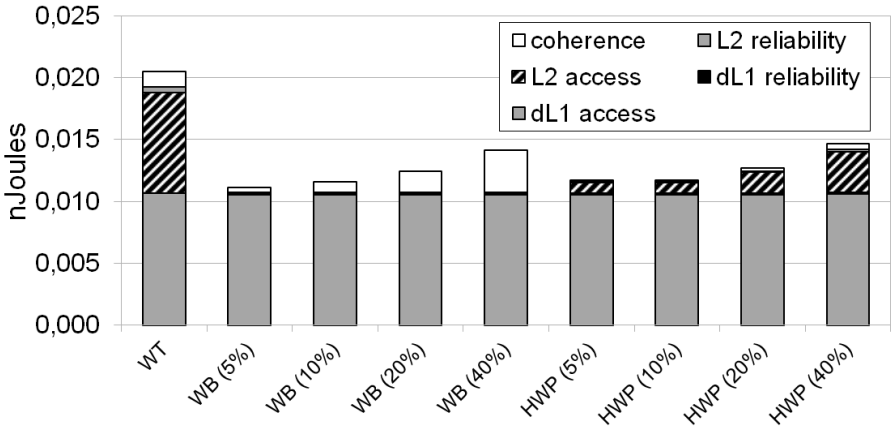
Guaranteed performance



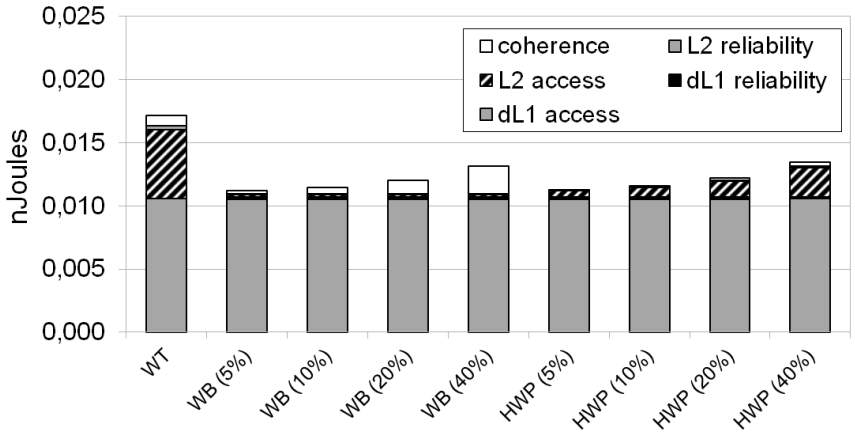
0% shared data | 10% shared data | 20% shared data | 40% shared data

- Each plot normalized to its own single-core
- Same trends we saw are seen across all setups

Energy



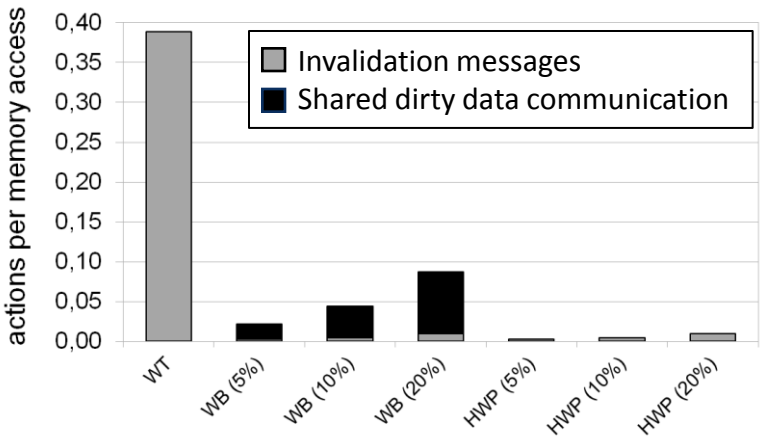
EEMBC



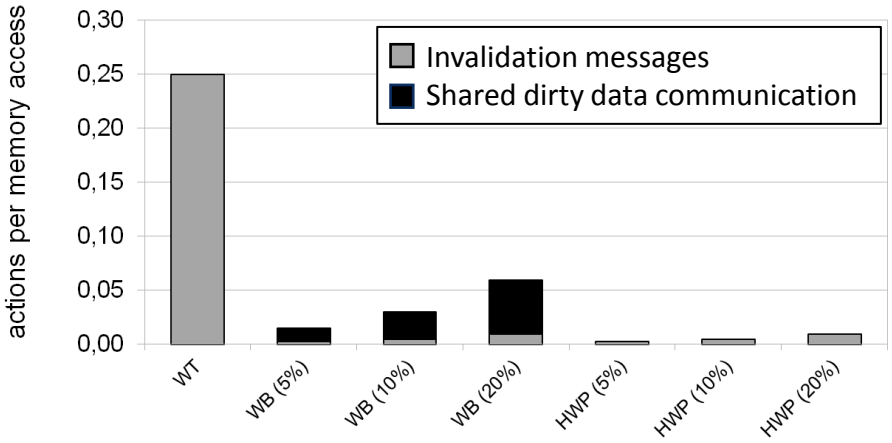
MediaBench

- Coherence is higher in WB policy
- Reliability has a small energy cost
- Main difference: L2 access energy

Coherence



EEMBC



MediaBench

- Invalidation messages
  - WT has a high number
  - WB and HWP only broadcast to shared data
- Shared dirty data communication
  - Significant impact in WB

Conclusions

- Both WT and WB offer tradeoffs in different metrics
  - No best policy, commercial architectures show this
- HWP tries to improve this
  - Not perfect, but improves overall
  - Guaranteed performance and energy similar to WB
  - Coherence complexity like WT

Thank you!
Any questions?

Pedro Benedicte, Carles Hernandez,
Jaume Abella, Francisco J. Cazorla