technische universität dortmund

Microcontroller Systems

WCET Analysis of Parallel Benchmarks using On-Demand Coherent Cache

Arthur Pyka, Lillian Tadros, Sascha Uhrig

University of Dortmund

Hugues Cassé, Haluk Ozaktas and Christine Rochange

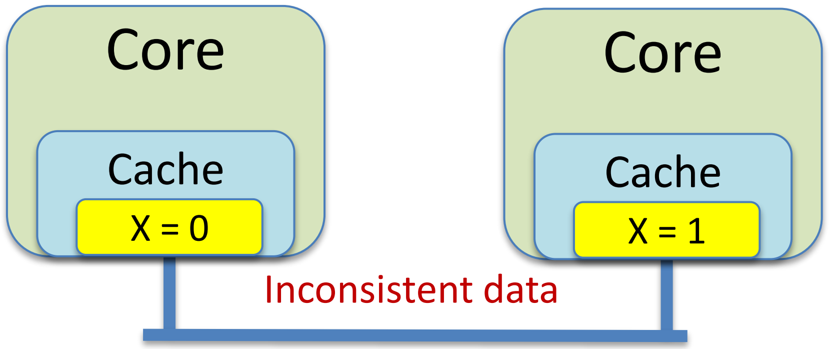
Université Paul Sabatier, Toulouse

Outline

- Cache Coherence in Todays Architectures
- Cache Coherence and Static WCET-Analysis
- In Short: The On-Demand Coherent Cache
- Static WCET-Analysis with ODC²
- Evaluation
- Conclusion

Cache Coherence in Todays Architectures

- Caches are part of almost every multi/many-core architecture.
- Accesses on shared data induce inconsistence of data in caches.
- Coherent accesses are preserved by cache coherence protocols.
- Coherence protocols affect the timing of data accesses.
- Coherence protocols affect the state and content of the caches.
- Coherence protocols affect a static worst-case execution time analysis.

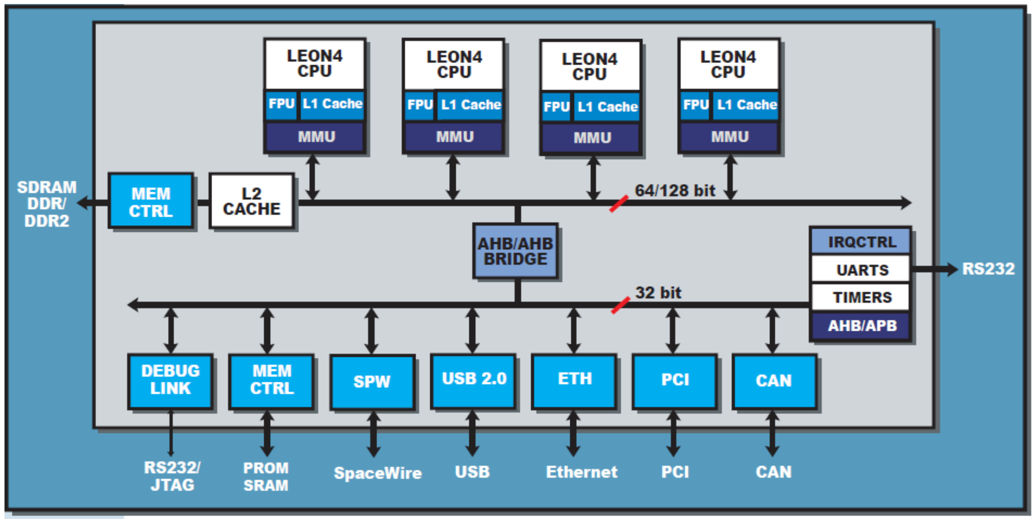


technische universität dortmund

Microcontroller Systems

Example: Quad-Core LEON4-FT GR712RC

- Bus interconnect
- Private L1-Caches
- Shared L2-Cache
- Write-Through Policy



Quad Core LEON4 Block Diagram [1]

- Cache Coherence is ensured using a **Snooping-based Write-Invalidate** protocol.

Snooping:	Any core listen to data accesses on the bus.
Write-Invalidate:	Invalidation of caches data by write accesses of other cores.

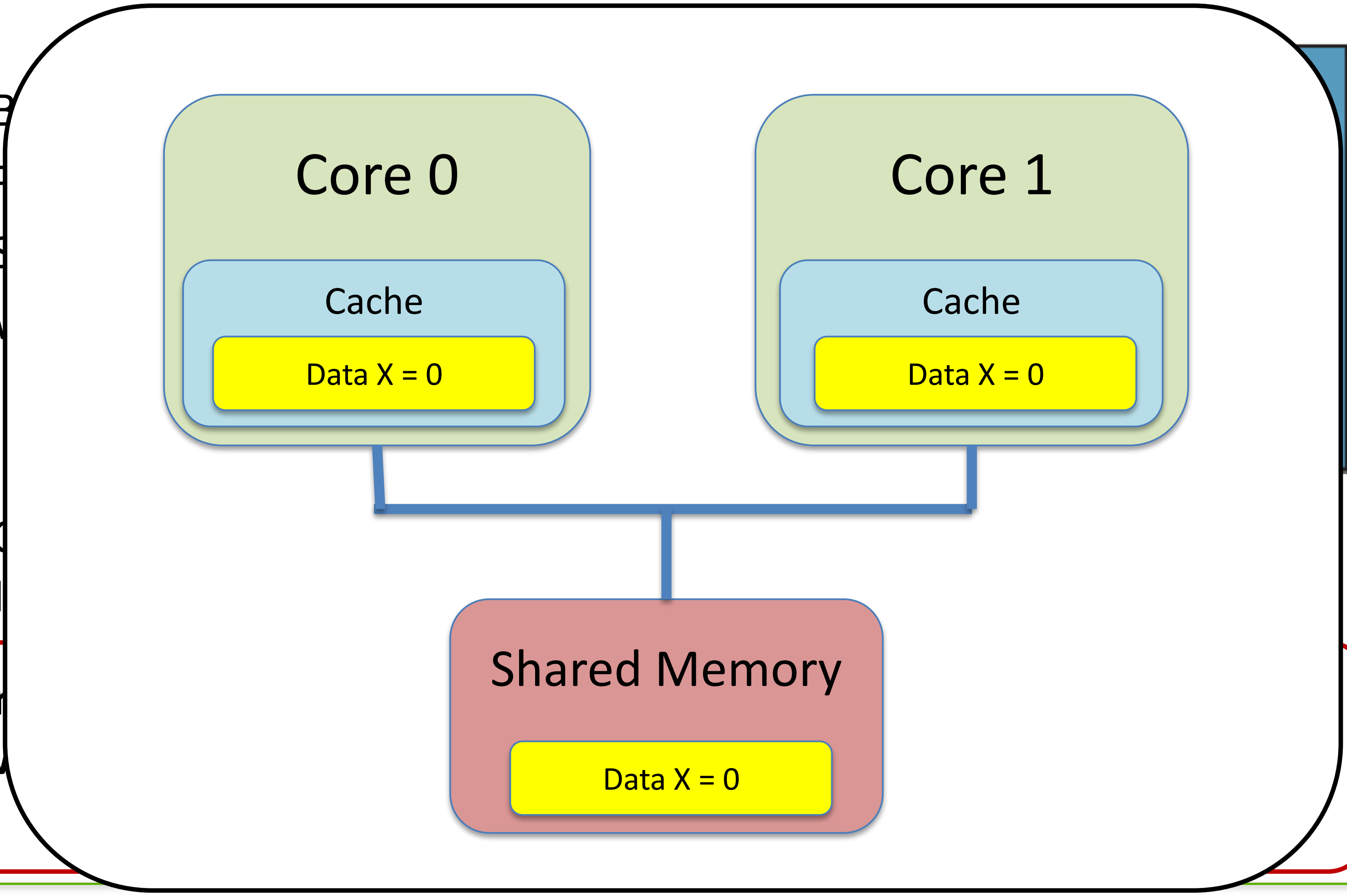
[1] Aeroflex Gaisler LEON4 Product Sheet

Example: Quad-Core LEON4-FT GR712RC

- B
- F
- S
- V

- C
  I

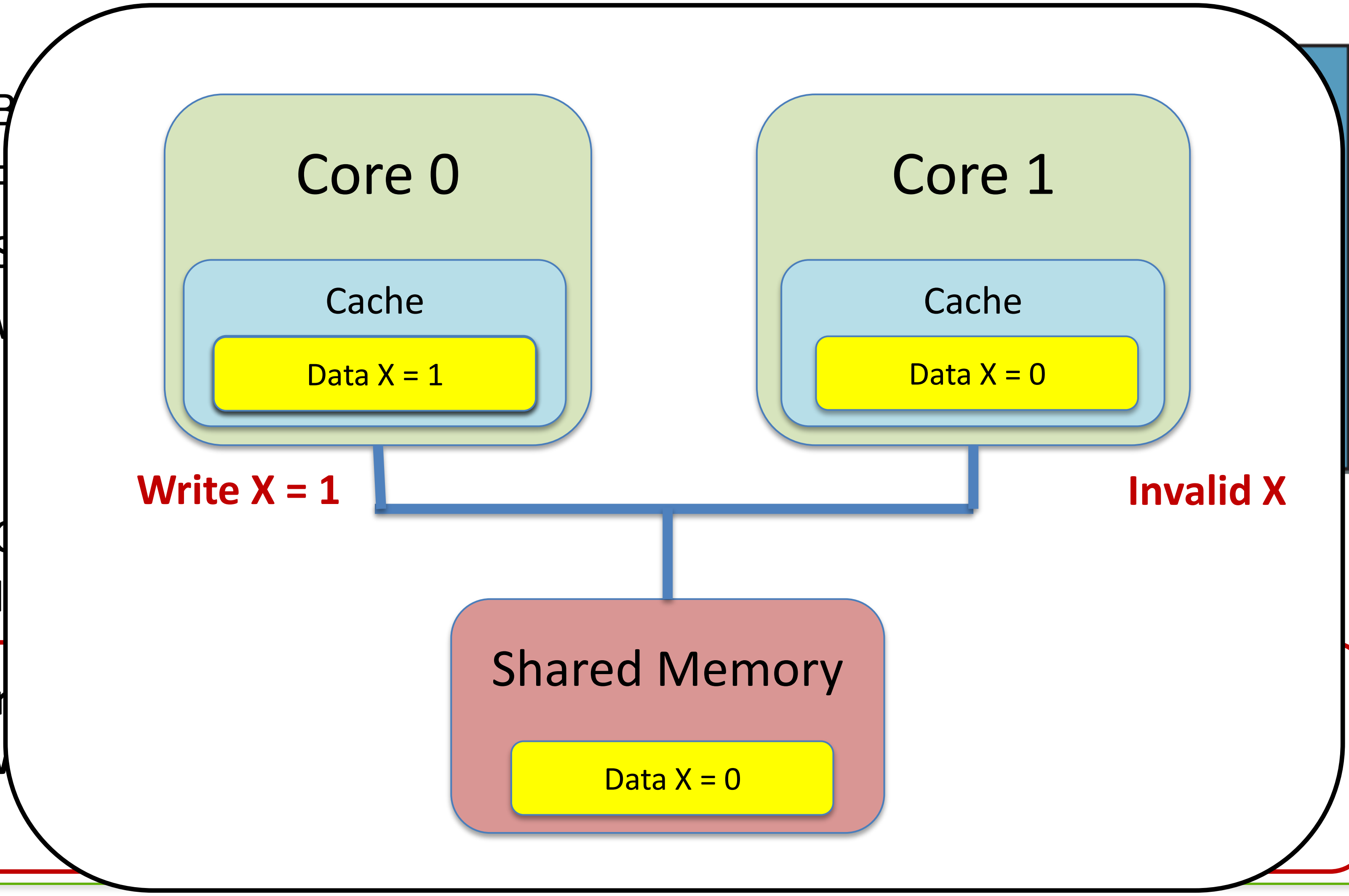
Sr
W



Example: Quad-Core LEON4-FT GR712RC

- B
- F
- S
- V
- C
  I

Sr
W

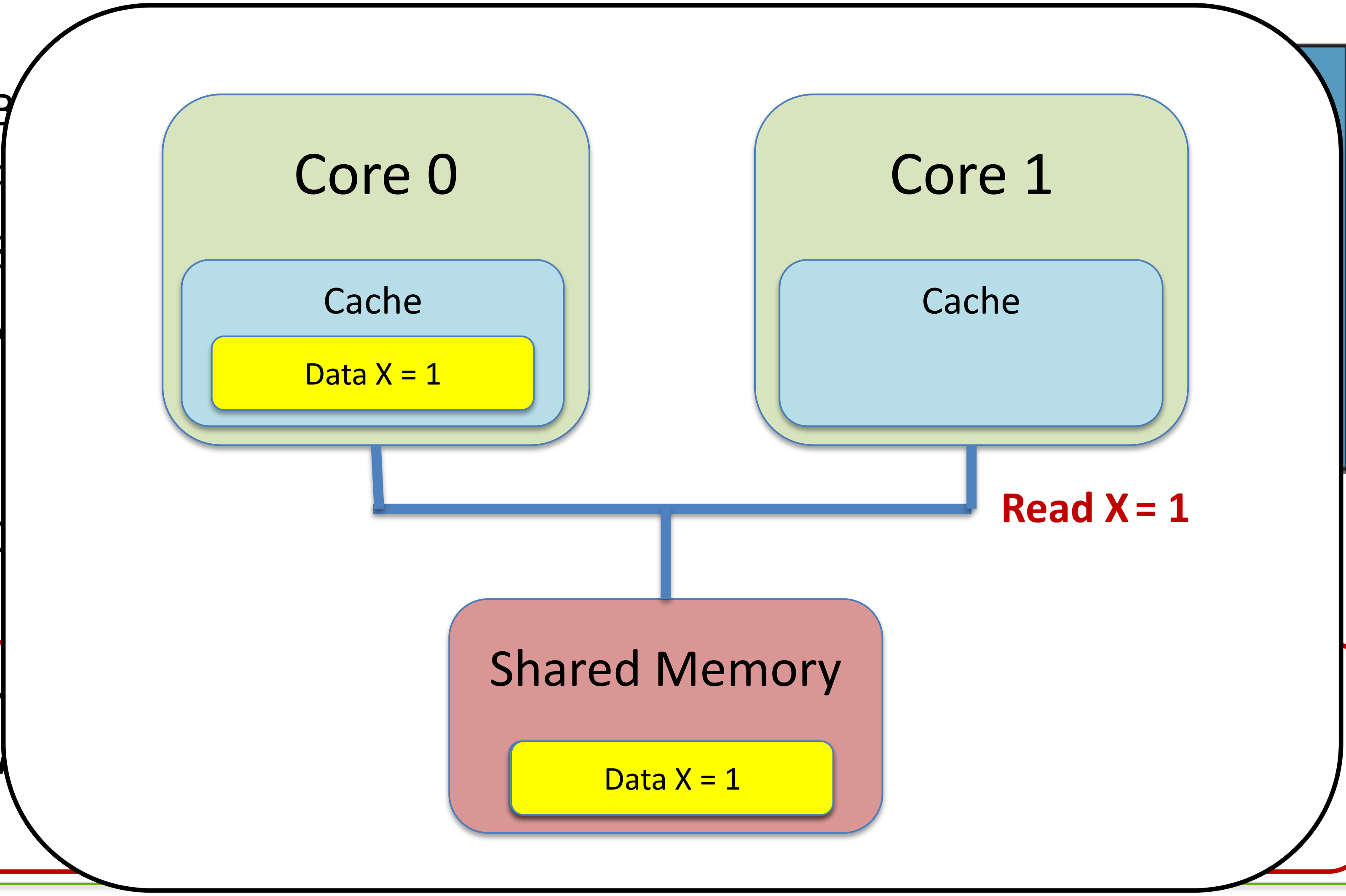


Example: Quad-Core LEON4-FT GR712RC

- B
- F
- S
- V

- C
  I

Sr
W

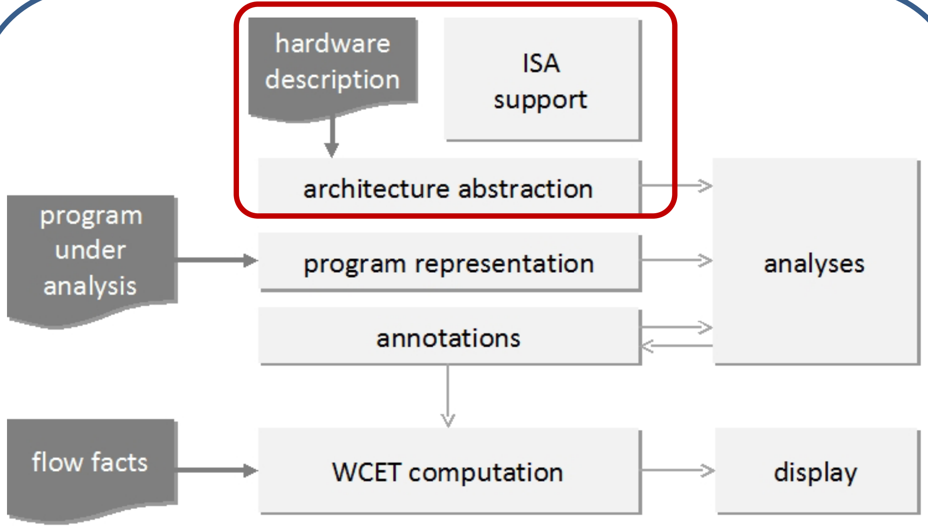


technische universität dortmund

Microcontroller Systems

Cache Coherence and Static WCET-Analysis

- Estimation of the WCET using a static analysis tool
- Abstraction of the hardware architecture incl. caches
- Cache Analysis to calculate data access latencies



Overview of the OTAWA framework [2]

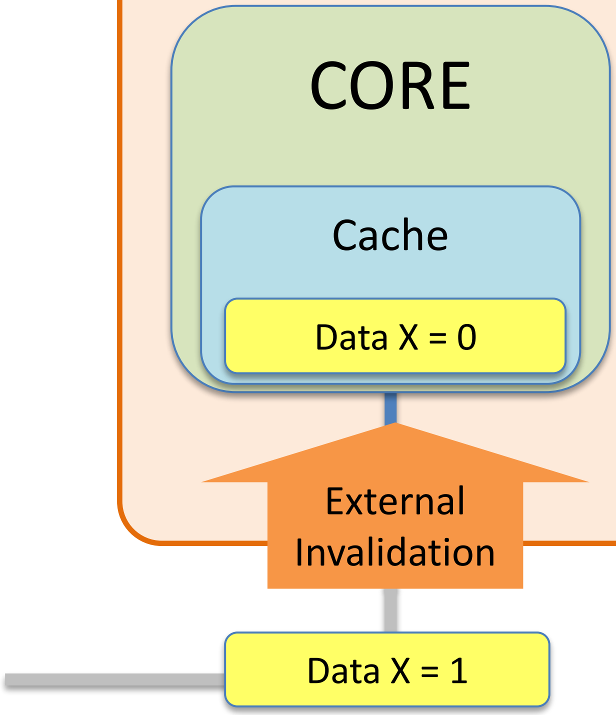
Cache Analysis

- Prediction of cache-hits and cache-misses
- Abstraction of possible cache states (Must/May Analysis)
- Modification of cache state by data accesses in program
- Analysis of private cache in isolation from other cores

[2] Ballabriga, C. et al.: OTAWA: an Open Toolbox for Adaptive WCET Analysis. In: *The 8th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*

Cache Coherence and Static WCET-Analysis

Cache Analysis



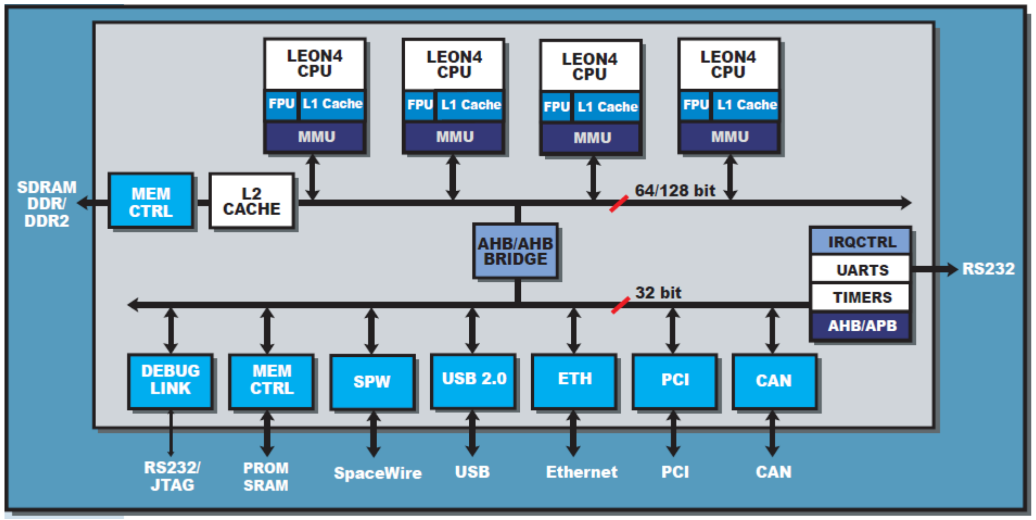
- External Invalidations turn the cache into a shared resource
- Target of invalidation is unknown
  -> Invalidation of any data must be assumed
- Time of invalidation is unknown
  -> Invalidation must be assumed at any possible time
- Prediction of cache-hits is nearly impossible

External invalidations prohibit a feasible WCET estimation.

technische universität dortmund

Example: Quad-Core LEON4-FT GR712RC

- Bus interconnect
- Private L1-Caches
- Shared L2-Cache
- Write-Through Policy



Quad Core LEON4 Block Diagram [1]

- Alternative: Pure software-based cache coherence using the **Cache-Flush** instruction.

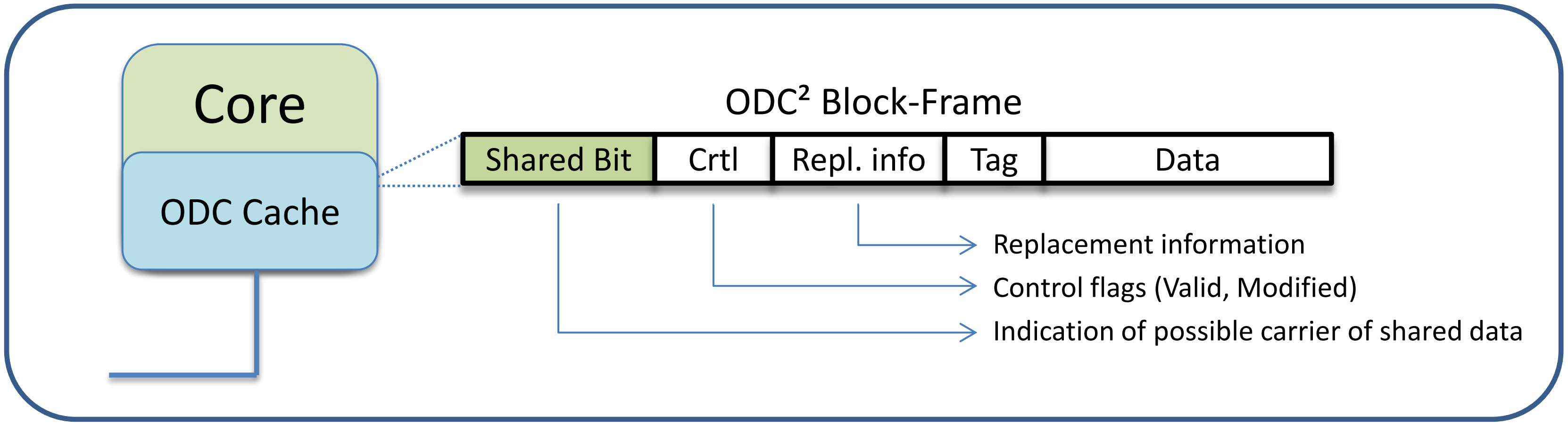
Cache-Flush: Invalidation of the complete cache triggered by a special instruction.

[1] Aeroflex Gaisler LEON4 Product Sheet

technische universität
dortmund

Microcontroller Systems

On-Demand Coherent Cache (ODC²)



- Software/Hardware co-approach
- Handling of coherence on demand
- Different handling of accesses to private and shared data
- Access to shared data only in protected code regions
- Invalidation of shared data after use
- No coherence transactions

On-Demand Coherent Cache (ODC²)

Abstract Program	ODC² Functionality

`…`
`- Accesses on private data -`
`…`
 	
`Mutex Lock / Barrier;`
	ODC² in Private-Mode
	• No coherence-operations
	• Functionality and timing equal to incoherent cache

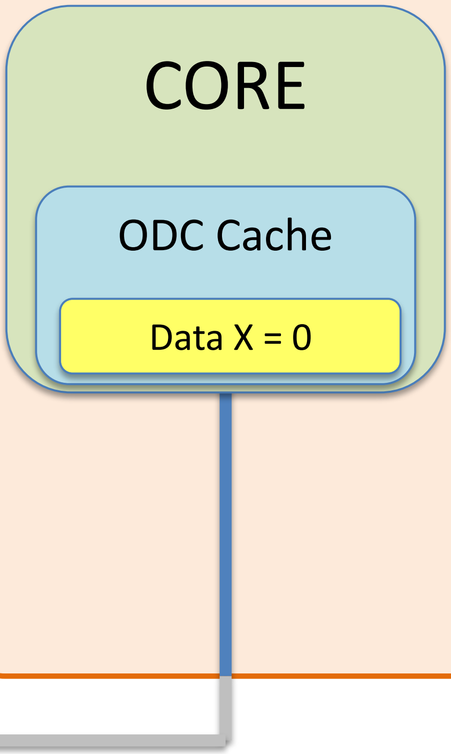
`enter_shared_mode();`
 	
`…`
`- Accesses on private/shared data -`
`…`
	ODC² in Shared-Mode
	• Newly loaded cache-lines are marked as *shared*
	• Checking of target address
	• Write-Through Policy

`exit_shared_mode();`	ODC² in Restore-Procedure • Invalidation of marked cache-lines

`Mutex Unlock / Barrier;`
 	
`…`
`- Accesses on private data -`
`…`
	ODC² in Private-Mode
	• No coherence-operations
	• Functionality and timing equal to incoherent cache

Static WCET-Analysis with ODC²

Cache Analysis



- Cache remains private resource
- Time and target of invalidation is known
- Invalidation increases cache-miss rate
- Invalidation improve efficiency
- Influence on timing is bounded and predictable
- Additional *Shared Analysis* to identify accesses to shared data
- New classification of accesses in Must/May Analysis: *Always Shared, Never Shared, Sometimes Shared*

Evaluation

System architecture:

- Tree-type interconnect
- PPC-based cores with private L1-Caches
- Shared memory

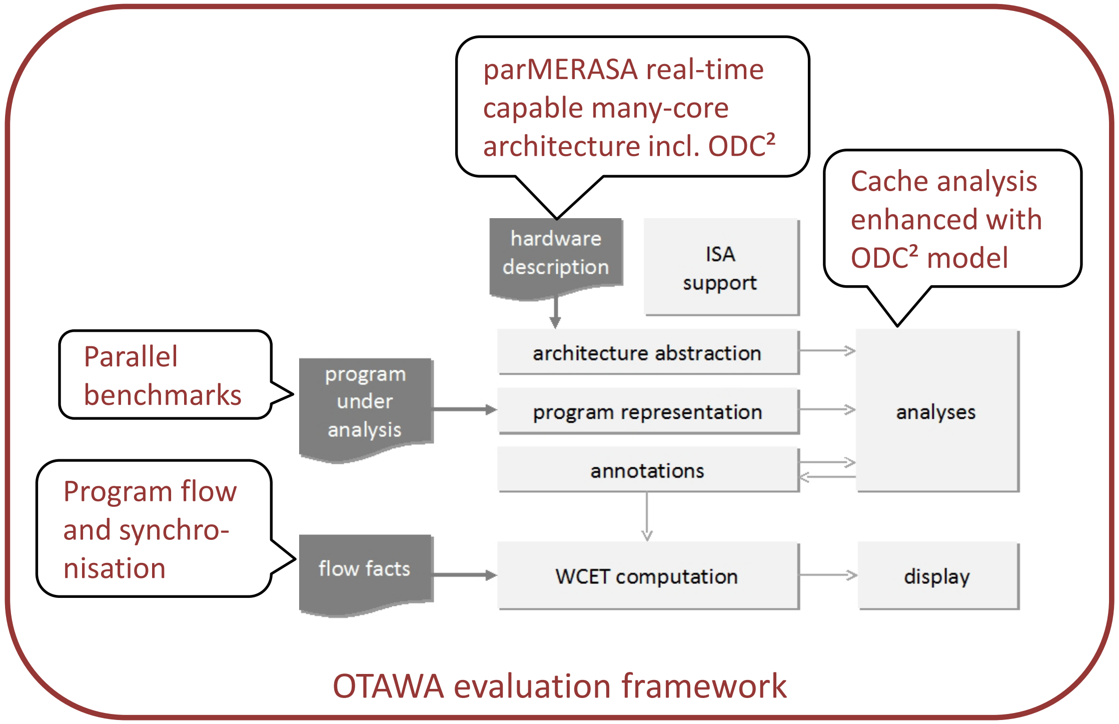
Parallel Benchmarks:

- Matrix Multiplication
- Fast Fourier Transformation
- Dijkstra Shortest Path

Evaluated Platforms:

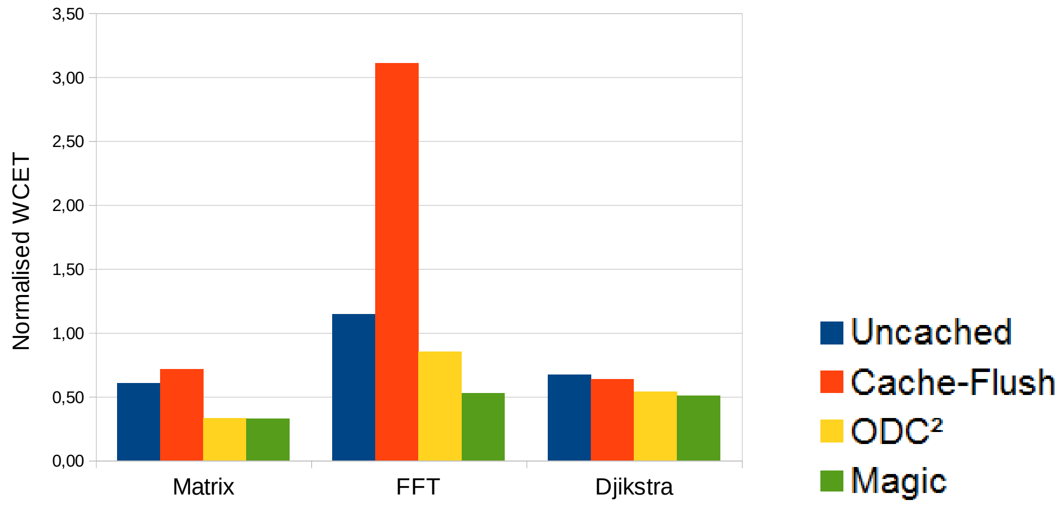
- ODC²
- Uncached shared data
- Software Cache-Flush
- Magic Coherence

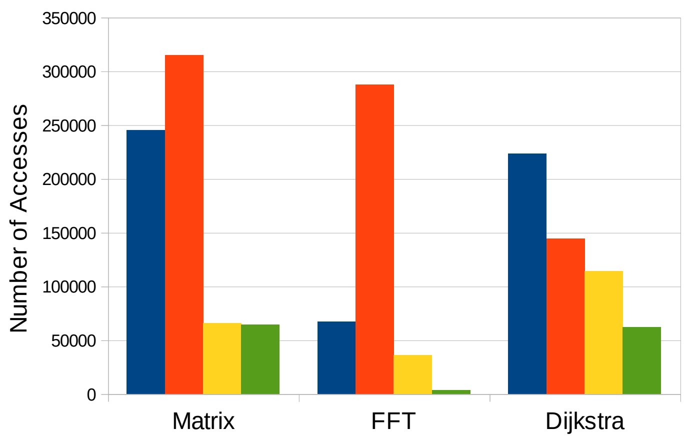
WCET analysis of ODC² and common alternatives using the OTAWA toolbox

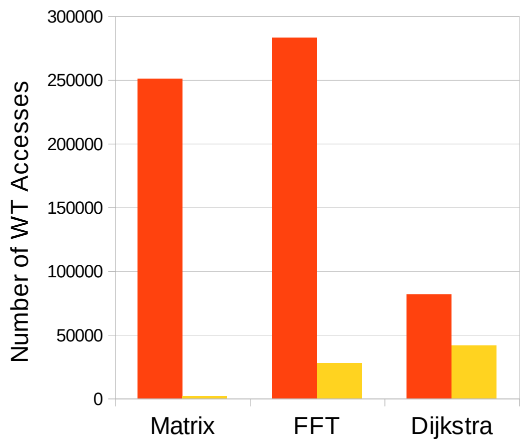


Evaluation

- WCET Analysis regarding parallel execution with 2-8 cores.
- Significantly reduced WCET compared to *Uncached* and *Cache-Flush*
- Reduced accesses to shared memory with ODC²
- Coherence overhead of ODC² depends on access pattern to shared memory







Conclusion

- Existing hardware cache coherence techniques are not suitable for a static WCET analysis, software coherence techniques lack in performance.
- ODC² permits time-predictable hardware/software co-approach.
- Static timing analysis using ODC² allows significant reduction of WCET compared to common alternatives.
- Promising approach for hard real-time multicore systems.

Thank you for your attention

Any questions?