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

- Caches are part of almost every multi/many-core architecture.
- Accesses on shared data induce inconsistency 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.
Example: Quad-Core LEON4-FT GR712RC

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

- 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.

---

Example: Quad-Core
LEON4-FT GR712RC

- Bus interconnect
- Private L1-Caches
- Shared L2-Cache
- Write-Through Strategy
- Cache Coherence is ensured using a Snooping-based Write-Invalidate protocol.
Example: Quad-Core
LEON4-FT GR712RC

- Bus interconnect
- Private L1-Caches
- Shared L2-Cache
- Write-Through Strategy
- Cache Coherence is ensured using a Snooping-based Write-Invalidate protocol.

Core 0

Cache
Data X = 1

Core 1

Cache
Data X = 0

Write X = 1

Shared Memory
Data X = 0

Invalid X
Example: Quad-Core
LEON4-FT GR712RC

- Bus interconnect
- Private L1 Caches
- Shared L2 Cache
- Write-Through Strategy
- Cache Coherence is ensured using a Snooping-based Write-Invalidate protocol.
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

Cache Coherence and Static WCET-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.
Example: Quad-Core LEON4-FT GR712RC

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

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

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

Quad Core LEON4 Block Diagram [1]

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

```c
... - Accesses on private data - ...

Mutex Lock / Barrier;

enter_shared_mode();

... - Accesses on private/shared data - ...

exit_shared_mode();

Mutex Unlock / Barrier;

... - Accesses on private data - ...```

### ODC² Functionality

<table>
<thead>
<tr>
<th>Mode</th>
<th>Features</th>
</tr>
</thead>
</table>
| **ODC² in Private-Mode** | - No coherence-operations  
                          | - Functionality and timing equal to incoherent cache                     |
| **ODC² in Shared-Mode**  | - Newly loaded cache-lines are marked as `shared`  
                          | - Checking of target address     
                          | - Write-Through Policy         |
| **ODC² in Restore-Procedure** | - Invalidation of marked cache-lines                              |
| **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?