WCET Analysis for Multi-Core Processors

Michael Jacobs, Sebastian Hack,
Jan Reineke, Reinhard Wilhelm

Department of Computer Science
Saarland University

February 28, 2013

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Outline

Need for WCET Analysis

- Embedded systems
- Safety-critical applications
  - E.g. in automotive or medical industry
- Strict timing requirements
  - Dictated by the physical environment
- Sound execution time bounds for programs needed

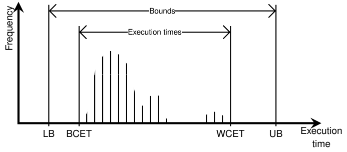
$\Rightarrow$ Worst-Case Execution Time (WCET) analysis

Execution Time of a Computer Program

- Execution time
  - Number of processor cycles
  - Needed to execute a given program
  - On a given hardware platform

⇒ Bounds are specific to a hardware platform

- Execution time depends on
  - *Program input*
    Which path through the program is taken?
  - *Initial system state*
    E.g. load of a cached memory block faster



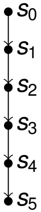
⇒ Sound bounds must hold for all possible combinations

Exact Behavior of a Computer System

A Formalization

- Exact behavior of a Core under Consideration (CuC)
  - Set of actual system states: S
  - Transitions under cycles of the CuC: $\mathit{Transitions} \subseteq S \times S$
  - A trace describes one execution behavior
- Not suitable for timing analysis
  - Realistic systems are complex
  - Large space of initial system states and program inputs
    - ★ Exhaustive simulation is no option
  - Many details irrelevant to timing

$\Rightarrow$ Abstract timing models needed



WCET Analysis Based on Abstract Timing Models

- Abstract timing model of the CuC
  - Set of abstract system states: $\widehat{S}$
  - Abstract cycle semantics of the CuC: $\widehat{\mathit{Transitions}} \subseteq \widehat{S} \times \widehat{S}$
- An abstract state may describe several concrete states

$$\begin{tikzcd} \widehat{s_0} \arrow[d, mapsto, "\gamma"] & \widehat{s_1} \arrow[d, mapsto, "\gamma"] \\ s_0, s_2 & s_1, s_3 \end{tikzcd}$$

- $\widehat{\mathit{Transitions}}$ subsumes *Transitions*
- An abstract trace may describe several concrete traces

Infeasible Traces

- Abstract models over-estimate the concrete execution behavior
- Example
  - Concrete system

    $s_0 \rightarrow s_1\,,\, s_2 \rightarrow s_3$
  - Abstraction

    $$\begin{tikzcd} \widehat{s_0} \arrow[d, mapsto, "\gamma"] & \widehat{s_1} \arrow[d, mapsto, "\gamma"] \\ s_0\,,\, s_2 & s_1\,,\, s_3 \end{tikzcd}$$
  - Abstract model

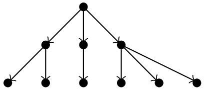
    $\widehat{s_0} \rightarrow \widehat{s_1}$
  - Described concrete traces

    $s_0 \rightarrow s_1\,,\, s_2 \rightarrow s_3\,,\, s_0 \rightarrow s_3\,,\, s_2 \rightarrow s_1$
- Abstraction has introduced *infeasible traces*

WCET Analysis for Single-Core Processors

- Sound and precise analyses exist
- Already high analysis complexity
  - Uncertainty about successor states
    - Non-determinism introduced by abstraction
    - Many case splits needed
  - State space explosion

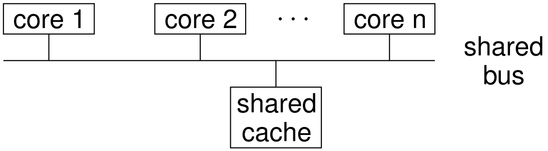
$$\begin{tikzcd}
 & \widehat{s_0} \arrow[dl, "\text{cache miss}"'] \arrow[dr, "\text{cache hit}"] & \\
\widehat{s_1} & & \widehat{s_2}
\end{tikzcd}$$



Outline

Transition to Multi-Core Processors

- Motivation
  - Reduce price, weight and energy consumption
    - Compared to multiple single-core processors
  - Use processors from the mass market
    - Further price reduction
- Several cores share common resources
  - Buses
  - Caches



- Disadvantage: Interference on shared resources
  - Subject of WP4 in R2

Bus Arbitration

- Shared bus
- One core at a time allowed to access
- Bus arbitration avoids bus conflicts
  - Grants access to one core
  - Blocks other cores requesting access
- Different arbitration protocols
  - Round-robin
  - FCFS (First-Come-First-Serve)
  - TDMA (Time-Division-Multiple-Access)

Bus Interference

- *Bus interference*
  - Arbitration influences a core's behavior
  - Core behaves differently than with dedicated bus

Cache Sharing

- Shared cache
- Several cores use the same cache lines
- Core A evicts a block loaded by core B
  - Core B may suffer an additional cache miss
- Core A preloads a block for core B
  - Core B enjoys an additional cache hit

Cache Interference

- *Cache capacity interference*
  - These additional misses and hits influence a core's behavior
  - Core behaves differently than with dedicated cache lines
- *Cache access interference*
  - Similar to bus interference
  - Not considered for now
  - Assume cache access resolved through shared bus arbitration

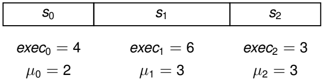
Challenges for WCET Analysis

- Traditional WCET analysis
  - Only considers one program executed on one core
- Analysis for multi-core processors has to consider interference
  - Bus interference
  - Cache interference
- Goal: Analyze program on one core
- However, programs on other cores may influence its behavior

$\Rightarrow$ Need for special timing analysis techniques

Existing Approaches

- Only applicable to compositional processor architectures
  - WCET analysis as if in isolation
  - Add penalties
    - ★ Cycles blocked at the bus
    - ★ Additional cycles on cache misses
  - Such processors are rarely available
- Only consider a very abstract model of computation
  - Based on superblocks



$exec_0 = 4$ $\mu_0 = 2$ $exec_1 = 6$ $\mu_1 = 3$ $exec_2 = 3$ $\mu_2 = 3$

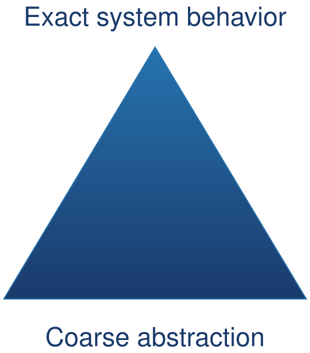
- Often only consider a single shared resource

$\Rightarrow$ Too restrictive

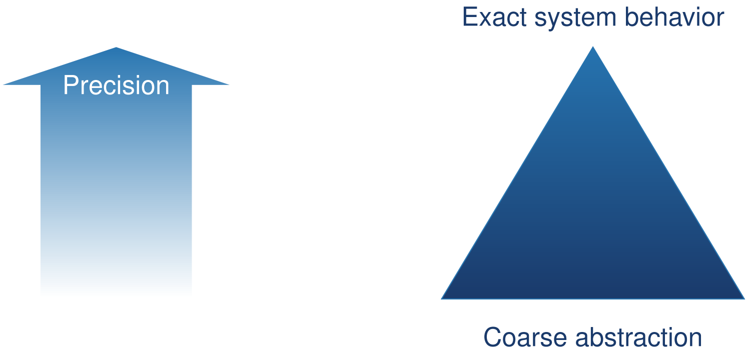
Considering Interference

- First approach
  - Analyzing the exact system behavior of all cores simultaneously
- Cores access shared resources
  - Different access interleavings exhibit different timing
- Must consider all interleavings
- State space explosion
  - Similar to verification of parallel programs
- Need for abstraction
  - Which is the right level of abstraction?

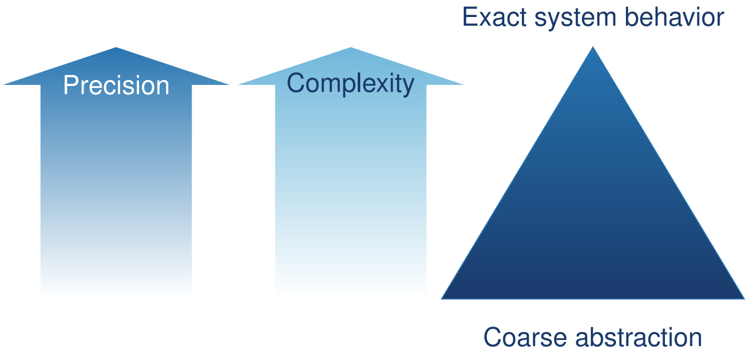
Precision versus Complexity



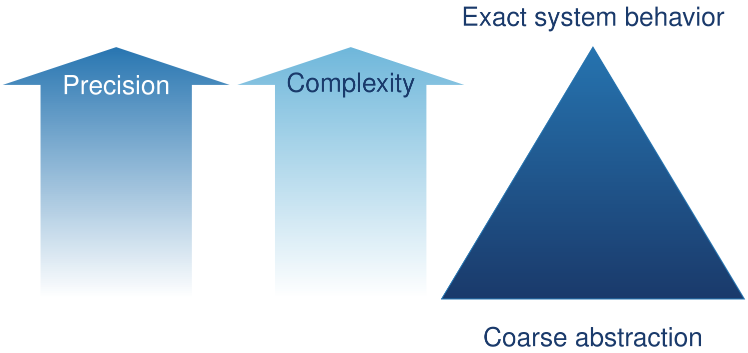
Precision versus Complexity



Precision versus Complexity



Precision versus Complexity



- Where is a good trade-off?

Coarse Abstraction + Interference Bounds

- Coarse abstraction as baseline
  - Consider only the analyzed core
  - Unknown state of the rest of the system
  - A lot of infeasible interference
- Improve by bounds on the interference
  - Precise enough
  - Efficiently obtained
- Exclude infeasible traces

Outline

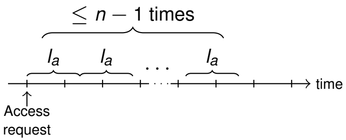
Bus Interference on a Concrete System

- A core executes a program part
- Execution is blocked for a number of cycles at the bus
- For many systems, this number can be bounded
- Bounds can be based on
  - The arbitration logic
  - The concurrent access behavior

Number of Blocked Cycles on a Concrete System

Bounds Based on the Arbitration Logic

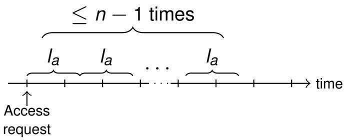
- Example: Round-robin arbitration
- Worst-case scenario [Pellizzoni and Caccamo, 2010]
  - All other cores are granted one access first
- An access may last at most l_a cycles
- Invariant for every feasible bus access
  - $\#blocked(access) \leq (n-1) * l_a$



Number of Blocked Cycles on a Concrete System

Bounds Based on the Arbitration Logic

- Example: Round-robin arbitration
- Worst-case scenario [Pellizzoni and Caccamo, 2010]
  - All other cores are granted one access first
- An access may last at most l_a cycles
- Invariant for every feasible bus access
  - $\#\mathit{blocked}(\mathit{access}) \leq (n-1) * l_a$



- Lift to an abstract trace
  - $\forall t \in \gamma(\widehat{\mathit{trace}})$:
    $\mathit{feasible}(t) \Rightarrow \#\mathit{blocked}(t) \leq \mathit{UBNumAccesses}(\widehat{\mathit{trace}}) * (n-1) * l_a$

Number of Blocked Cycles on a Concrete System

Bounds Based on the Concurrent Access Behavior

- Consider event-driven bus arbitration
  - Only blocked if another core has been granted access
- Invariant for every feasible concrete trace
  - It is not blocked longer than other cores access the bus
  - Corresponds to the constraint
    $\#\mathit{blocked}(\mathit{trace}) \leq \#\mathit{concurrentBusAccesses}(\mathit{trace})$

Number of Blocked Cycles on a Concrete System

Bounds Based on the Concurrent Access Behavior

- Consider event-driven bus arbitration
  - Only blocked if another core has been granted access
- Invariant for every feasible concrete trace
  - It is not blocked longer than other cores access the bus
  - Corresponds to the constraint $\#\mathit{blocked}(\mathit{trace}) \leq \#\mathit{concurrentBusAccesses}(\mathit{trace})$
- Lift to an abstract trace
  - Exact amount of concurrent bus access cycles not known
  - Pre-analyze the co-running tasks for an upper bound [Wandeler et al., 2006, Pellizzoni et al., 2010]
    - ★ Per number of execution cycles
    - ★ $\mathit{UBNumConcurrentBusAccesses} : \mathbb{N} \to \mathbb{N}$
  - $\forall t \in \gamma(\widehat{\mathit{trace}})$:
    $$\mathit{feasible}(t) \Rightarrow \#\mathit{blocked}(t) \leq \mathit{UBNumConcurrentBusAccesses}(\#\mathit{cycles}(\widehat{\mathit{trace}}))$$

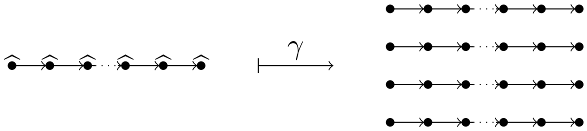
Bus Interference in an Abstract Model

- Consider an abstract trace

$$\begin{tikzcd}
\widehat{\bullet} \arrow[r] & \widehat{\bullet} \arrow[r] & \widehat{\bullet} \arrow[r, dotted] & \widehat{\bullet} \arrow[r] & \widehat{\bullet} \arrow[r] & \widehat{\bullet}
\end{tikzcd}$$

Bus Interference in an Abstract Model

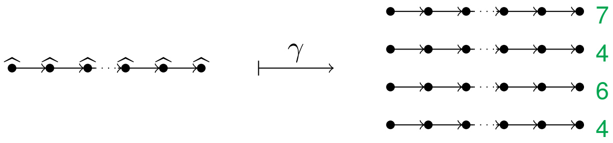
- Consider an abstract trace



- And the concrete traces it describes

Bus Interference in an Abstract Model

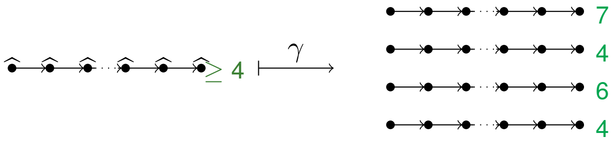
- Consider an abstract trace



- And the concrete traces it describes
- Number of blocked cycles known for each concrete trace

Bus Interference in an Abstract Model

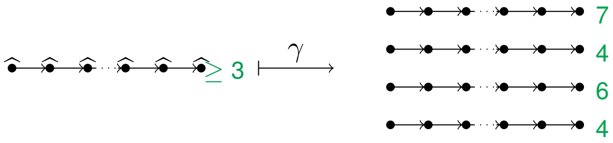
- Consider an abstract trace



- And the concrete traces it describes
- Number of blocked cycles known for each concrete trace
- Annotate abstract trace with a lower bound on them

Bus Interference in an Abstract Model

- Consider an abstract trace



- And the concrete traces it describes
- Number of blocked cycles known for each concrete trace
- Annotate abstract trace with a lower bound on them
- Approximate coarse lower bound

Bus Interference in an Abstract Model

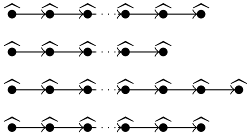
- Consider an abstract trace

$$\begin{tikzcd} \widehat{\bullet} \arrow[r] & \widehat{\bullet} \arrow[r] & \widehat{\bullet} \arrow[r, dotted] & \widehat{\bullet} \arrow[r] & \widehat{\bullet} \arrow[r] & \widehat{\bullet}\geq 3 \end{tikzcd}$$

- And the concrete traces it describes
- Number of blocked cycles known for each concrete trace
- Annotate abstract trace with a lower bound on them
- Approximate coarse lower bound
  - Without looking at the concrete traces

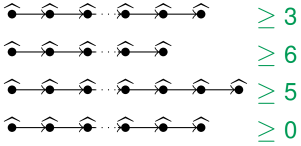
Detecting Infeasible Abstract Traces

- Set of abstract traces for a program part



Detecting Infeasible Abstract Traces

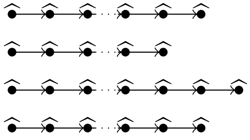
- Set of abstract traces for a program part



- Annotate lower bounds on the number of blocked cycles

Detecting Infeasible Abstract Traces

- Set of abstract traces for a program part

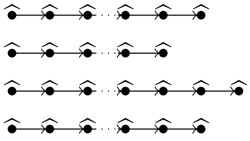


≥ 3	≤ 4
≥ 6	≤ 3
≥ 5	≤ 4
≥ 0	≤ 3

- Annotate lower bounds on the number of blocked cycles
- Annotate upper bounds on the number of blocked cycles
  - Derived for the concrete system and the arbitration protocol
  - Hold for all feasible concrete traces described

Detecting Infeasible Abstract Traces

- Set of abstract traces for a program part

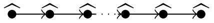
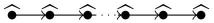


≥ 3	≤ 4	
≥ 6	≤ 3	↯
≥ 5	≤ 4	↯
≥ 0	≤ 3	

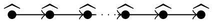
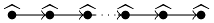
- Annotate lower bounds on the number of blocked cycles
- Annotate upper bounds on the number of blocked cycles
  - Derived for the concrete system and the arbitration protocol
  - Hold for all feasible concrete traces described
- Lower bound and upper bound contradict
  - All concrete traces described are infeasible
  - Then so is the abstract trace

Detecting Infeasible Abstract Traces

- Set of abstract traces for a program part

	≥ 3	≤ 4	
	≥ 6	≤ 3	↯
	≥ 5	≤ 4	↯
	≥ 0	≤ 3	

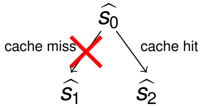
- Annotate lower bounds on the number of blocked cycles
- Annotate upper bounds on the number of blocked cycles
  - Derived for the concrete system and the arbitration protocol
  - Hold for all feasible concrete traces described
- Lower bound and upper bound contradict
  - All concrete traces described are infeasible
  - Then so is the abstract trace
- Remove infeasible abstract traces

	≥ 3	≤ 4
	≥ 0	≤ 3

Outline

Bounding Cache Interference

- Assume *only* cache capacity interference
  - Cache not accessed by two cores at the same time
- Coarse abstraction
  - Each access could *hit* or *miss* the cache
- Goal: Predict more cache hits
  - Exclude some case splits



Shared Cache Analysis

Independent of Co-Running Tasks

- Analysis as if on a single-core processor [Ferdinand and Wilhelm, 1997]
- Is an accessed block still a cache hit?
  - Consider maximum time since last access
  - How many changes to the cache could arbitrary programs on other cores maximally make in that time?
  - Is it enough to evict the cache block?
  - If not, still a *cache hit*
- Very conservative if co-running tasks do not make full use of the cache

Shared Cache Analysis

Depending on Co-Running Tasks

- Same idea as before
- Slight change to the classification
  - How many changes to the cache can the co-running tasks maximally make in that time?
- Use an upper bound on the concurrent cache access behavior
  - Per number of execution cycles
- Pre-analyze the concurrent cores for this bound

Outline

A Classification

Approaches to Interference Bounding

	Bus interference	Cache interference
Independent of co-running tasks	Based on arbitration protocol	Shared cache analysis independent of co-running tasks
Depending on co-running tasks	Based on concurrent access behavior	Shared cache analysis depending on co-running tasks

Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Summary

- WCET analysis for multi-core processors is important
- There is potential for improvements
- Find good trade-off
  - Precision
  - Complexity
- Considering co-running tasks may lead to more precise results

- Outlook
  - Implement a WCET analysis for a multi-core processor
  - Evaluate different levels of precision

References I

Ferdinand, C. and Wilhelm, R. (1997).
Fast and efficient cache behavior prediction.

Pellizzoni, R. and Caccamo, M. (2010).
Impact of peripheral-processor interference on WCET analysis of real-time embedded systems.
IEEE Transactions on Computers, 59:400–415.

Pellizzoni, R., Schranzhofer, A., Chen, J.-J., Caccamo, M., and Thiele, L. (2010).
Worst case delay analysis for memory interference in multicore systems.
In *Proceedings of the 13th Conference on Design, Automation and Test in Europe*, DATE '10, pages 741–746, 3001 Leuven, Belgium, Belgium. European Design and Automation Association.

Wandeler, E., Thiele, L., Verhoef, M., and Lieverse, P. (2006).
System architecture evaluation using modular performance analysis: a case study.
Int. J. Softw. Tools Technol. Transf., 8(6):649–667.