WCET Analysis for Multi-Core Processors with Shared Buses and Event-Driven Bus Arbitration

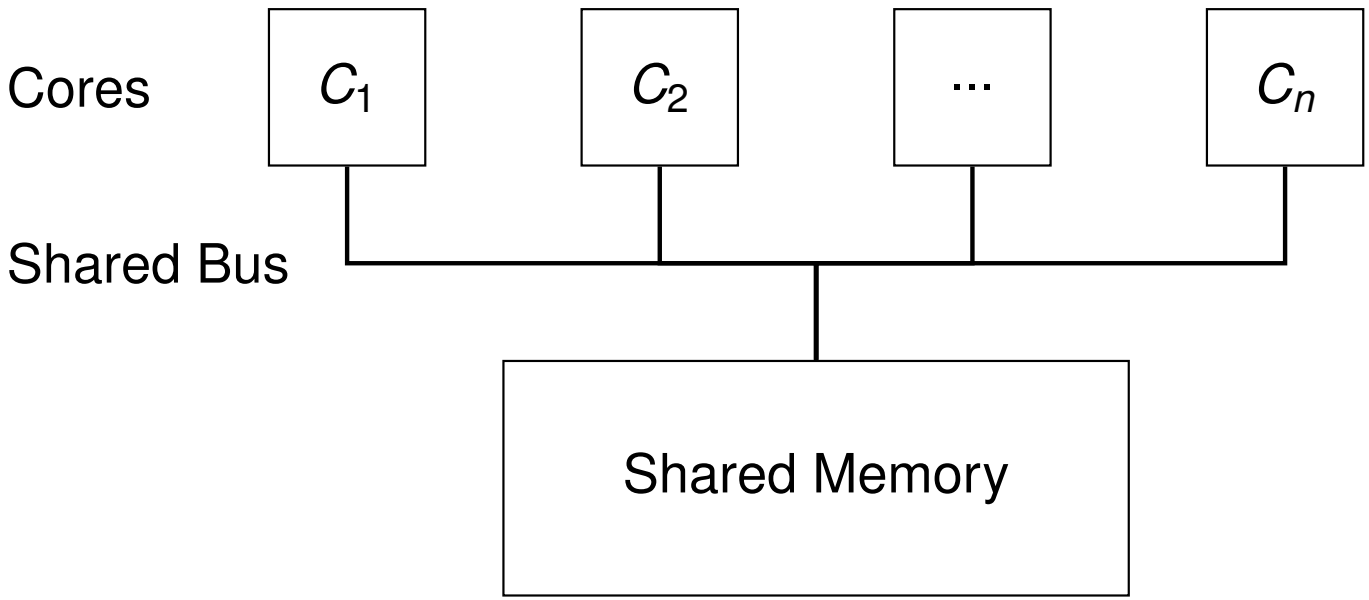
Michael Jacobs, Sebastian Hahn, Sebastian Hack

Department of Computer Science
Saarland University

November 16, 2015

Considered HW Platform

- Multi-core processor with n cores
- Shared bus
  - Connecting the cores to the memory
  - Event-driven bus arbitration
  - Running example: round-robin



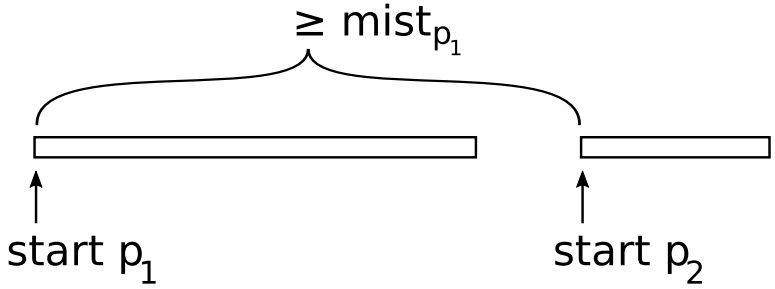
Considered Execution Model

Set of programs:

- $Progs = \{p_1, \ldots, p_{|Progs|}\}$

Per program $p_i \in Progs$:

- Minimum inter-start time ($mist_{p_i}$)
  - Optional
  - Zero if not specified



Scheduling:

- Partitioned
- Non-preemptive

WCET Analysis for Multi-Core Processors

- **Calculate WCET bound** for a program executed on a core
  - Must consider shared-resource interference!
  - E.g. cycles blocked at shared bus

Two kinds of WCET bounds:

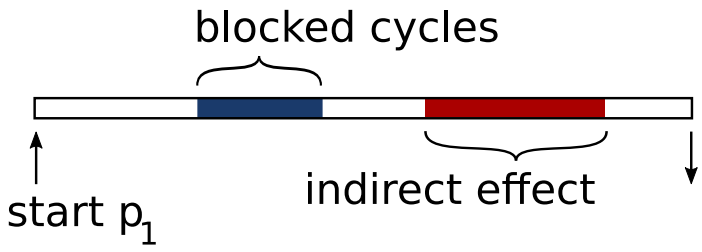
- **Co-runner-insensitive**
  - Independent of co-running programs
  - Only depend on the HW platform
  - Implicitly assume worst co-runners
- **Co-runner-sensitive**
  - Take into account co-running programs
  - Consider (limited) scheduling knowledge
  - Potentially more precise

We propose approaches for both!

Existing Approaches

- Compositionality [Schranzhofer et al., 2011]
  - WCET analysis ignores bus blocking
  - Bound on blocked cycles is added
  - Ignores indirect effects

  $\Rightarrow$ **Unsound** for many HW platforms, e.g.
  - In-order pipelines with unblocked stores
  - Out-of-order pipelines

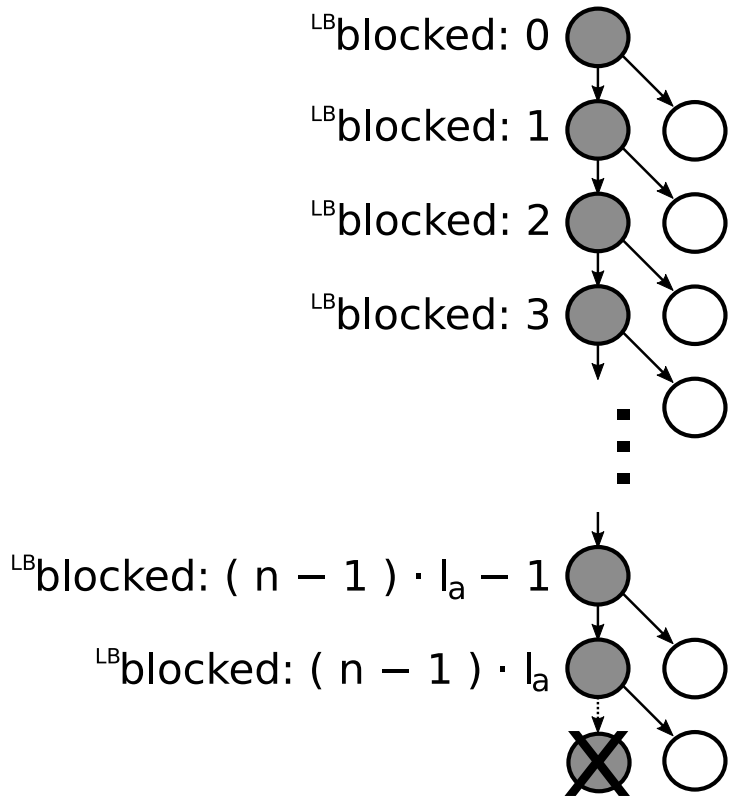


- Enumerate possible interleavings of accesses by the cores [Kelter and Marwedel, 2014]
  - High computational complexity
  - Strong synchronicity assumptions

Co-Runner-Insensitive Analysis

Modeling Shared-Bus Interference

- By non-determinism
  - A pending access request can be:
    - ★ granted immediately or
    - ★ blocked for another cycle
  - Splits in micro-architectural analysis
- Bounding the non-determinism
  - Worst-case per access request
  - E.g. for round-robin arbitration
    - ★ Each concurrent core is granted a complete access first:



- Path analysis
  - Find longest path through graph
  - Modeled as integer linear program (ILP)
  - Classical implicit path enumeration [Li and Malik, 1995]

Experimental Evaluation

- Hardware configuration
  - In-order execution
  - local instruction scratchpad (fitting whole program)
  - local data cache (misses served via bus)
  - Round-robin bus arbitration
- 31 benchmarks
  - Mälardalen
  - Generated from SCADE models
- Results normalized to analysis ignoring bus interference
- Geometric mean over normalized results

Poor Scalability

- Non-determinism increases with number of cores

	2-Core	*4-Core*
analysis runtime	8.878	38.840
peak memory cons.	1.581	3.616

Exploiting Pipeline Convergence

- Pipeline states often converge
  - After a few cycles blocked at the bus
  - State unchanged until access finished
  - **Converged chain**

Exploiting Pipeline Convergence

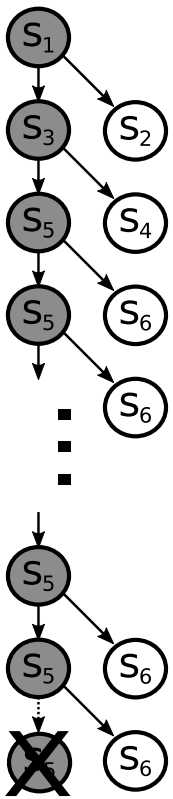
- Pipeline states often converge
  - After a few cycles blocked at the bus
  - State unchanged until access finished
  - **Converged chain**, e.g. for s_5

^{UB}time: 15 — ^{LB}blocked: 0

^{UB}time: 16 — ^{LB}blocked: 1

^{UB}time: 17 — ^{LB}blocked: 2

^{UB}time: 18 — ^{LB}blocked: 3



^{UB}time: $15 + (n - 1) \cdot l_a - 1$ — ^{LB}blocked: $(n - 1) \cdot l_a - 1$

^{UB}time: $15 + (n - 1) \cdot l_a$ — ^{LB}blocked: $(n - 1) \cdot l_a$

Exploiting Pipeline Convergence

- Pipeline states often converge
  - After a few cycles blocked at the bus
  - State unchanged until access finished
  - **Converged chain**
- ^{UB}time dominated by last state in chain
  - Safely replace chain by last state in it

^{UB}time: 15

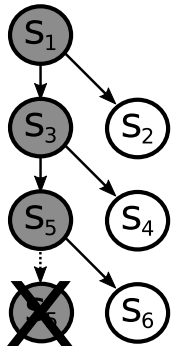
^{UB}time: 16

^{UB}time: $15 + (n - 1) \cdot l_a$

^{LB}blocked: 0

^{LB}blocked: 1

^{LB}blocked: $(n - 1) \cdot l_a$



- **Fast-forwarding** of converged chains

Improved Scalability

- Fast-forwarding improves scalability
- In-order execution

	instr. scratchpad data cache		instr. cache data cache	
	2-Core	*4-Core*	*2-Core*	*4-Core*
WCET bound	1.604	2.803	1.678	3.028
analysis runtime	1.685	1.670	5.905	5.903
peak memory cons.	1.056	1.056	1.430	1.423

- Runtime and memory consumption **independent** of n

Improved Scalability

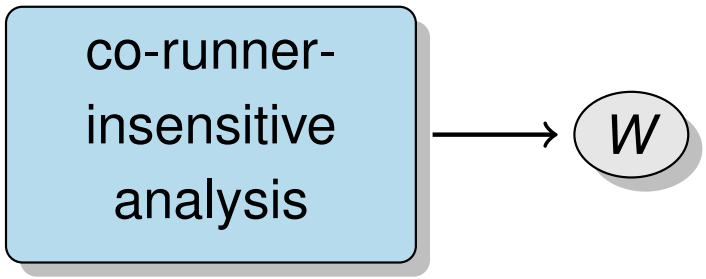
- Fast-forwarding improves scalability
- Out-of-order execution

	instr. scratchpad data cache		instr. cache data cache	
	2-Core	*4-Core*	*2-Core*	*4-Core*
WCET bound	1.657	2.965	1.726	3.175
analysis runtime	3.339	3.473	39.170	47.271
peak memory cons.	1.165	1.187	6.303	7.591

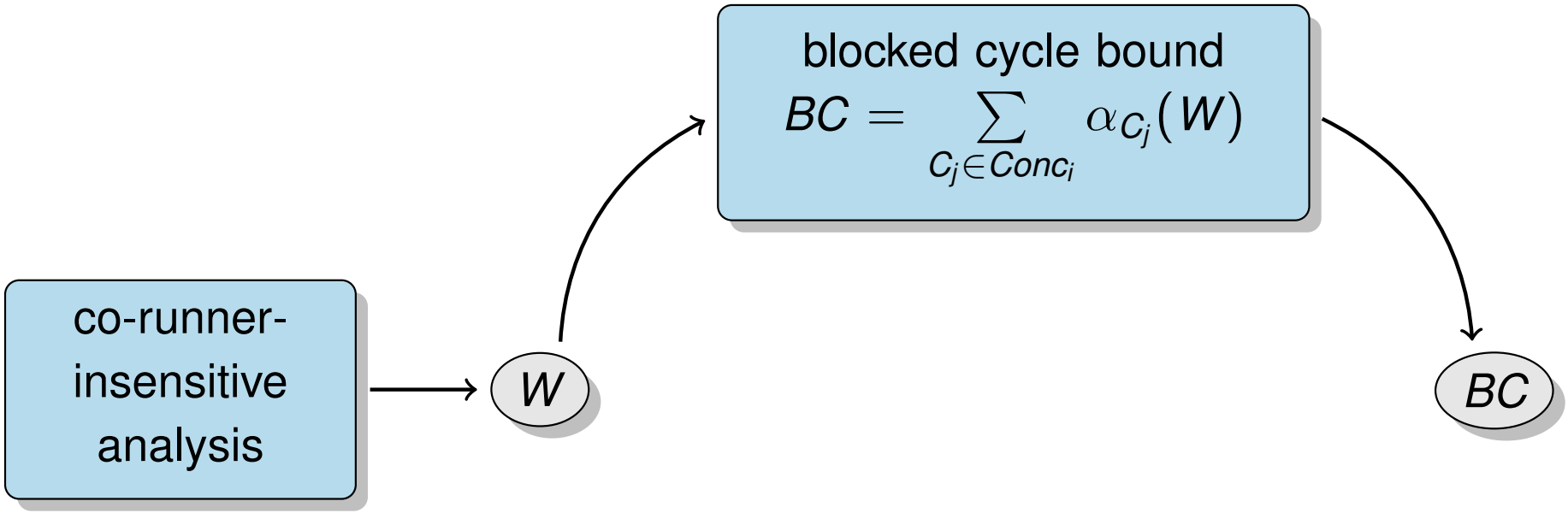
- **Moderate growth** of runtime and memory consumption w.r.t. *n*

Co-Runner-Sensitive Analysis

Iterative Co-Runner-Sensitive Analysis



Iterative Co-Runner-Sensitive Analysis

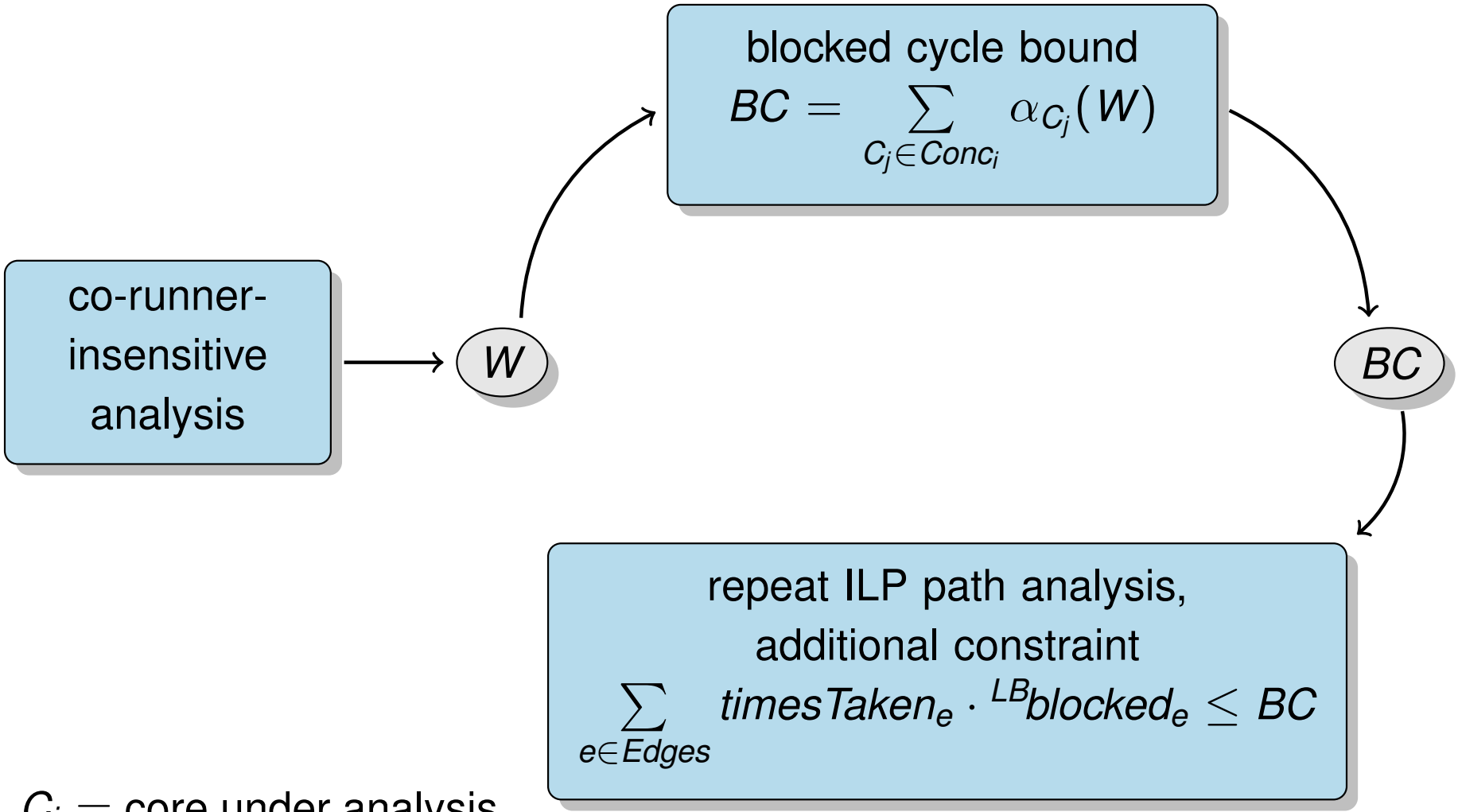


C_i = core under analysis

$Conc_i = Cores \setminus \{C_i\}$

$\alpha_{C_j}(W)$ = upper bound on number of access cycles of core C_j in W cycles

Iterative Co-Runner-Sensitive Analysis

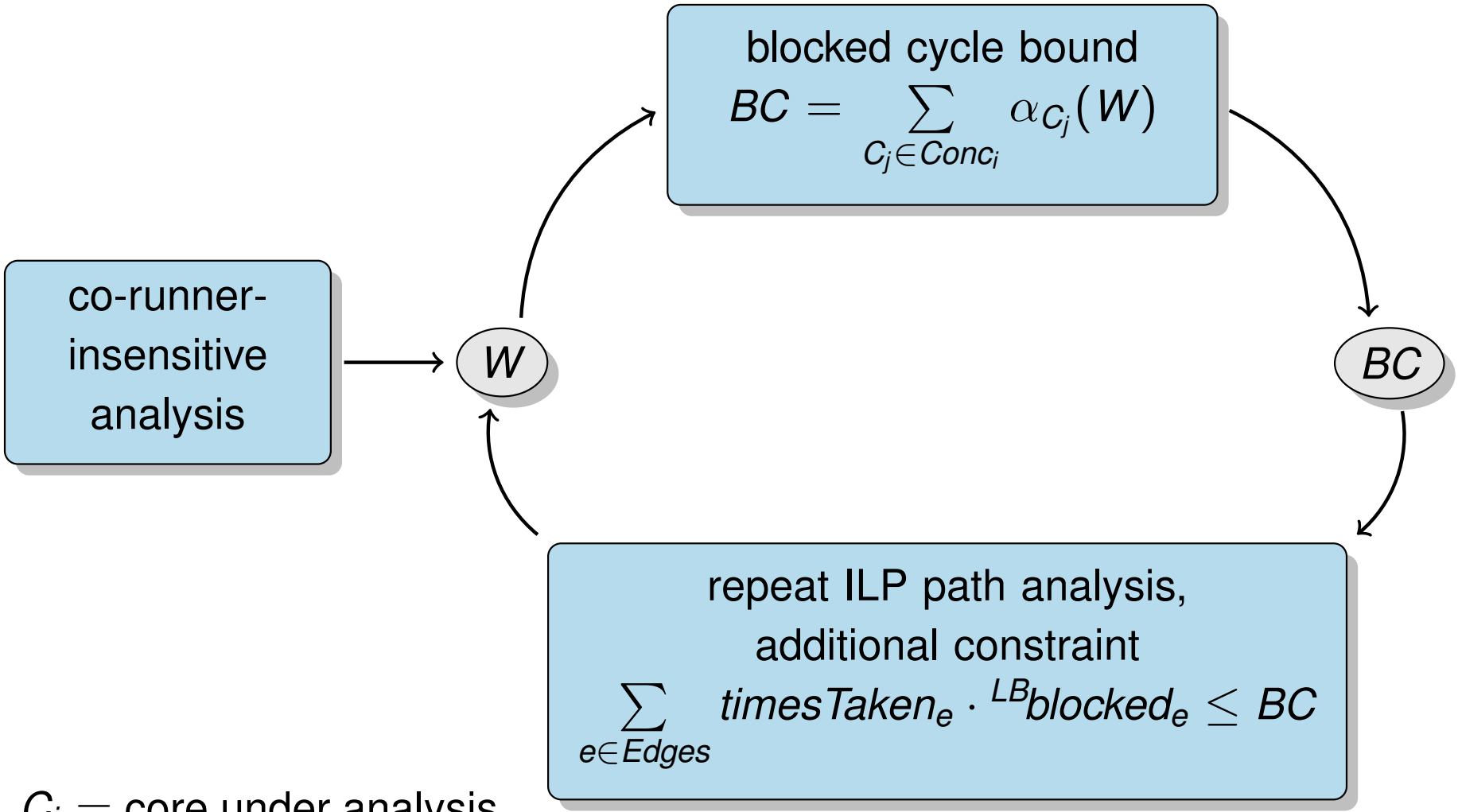


C_i = core under analysis

$Conc_i = Cores \setminus \{C_i\}$

$\alpha_{C_j}(W)$ = upper bound on number of access cycles of core C_j in W cycles

Iterative Co-Runner-Sensitive Analysis

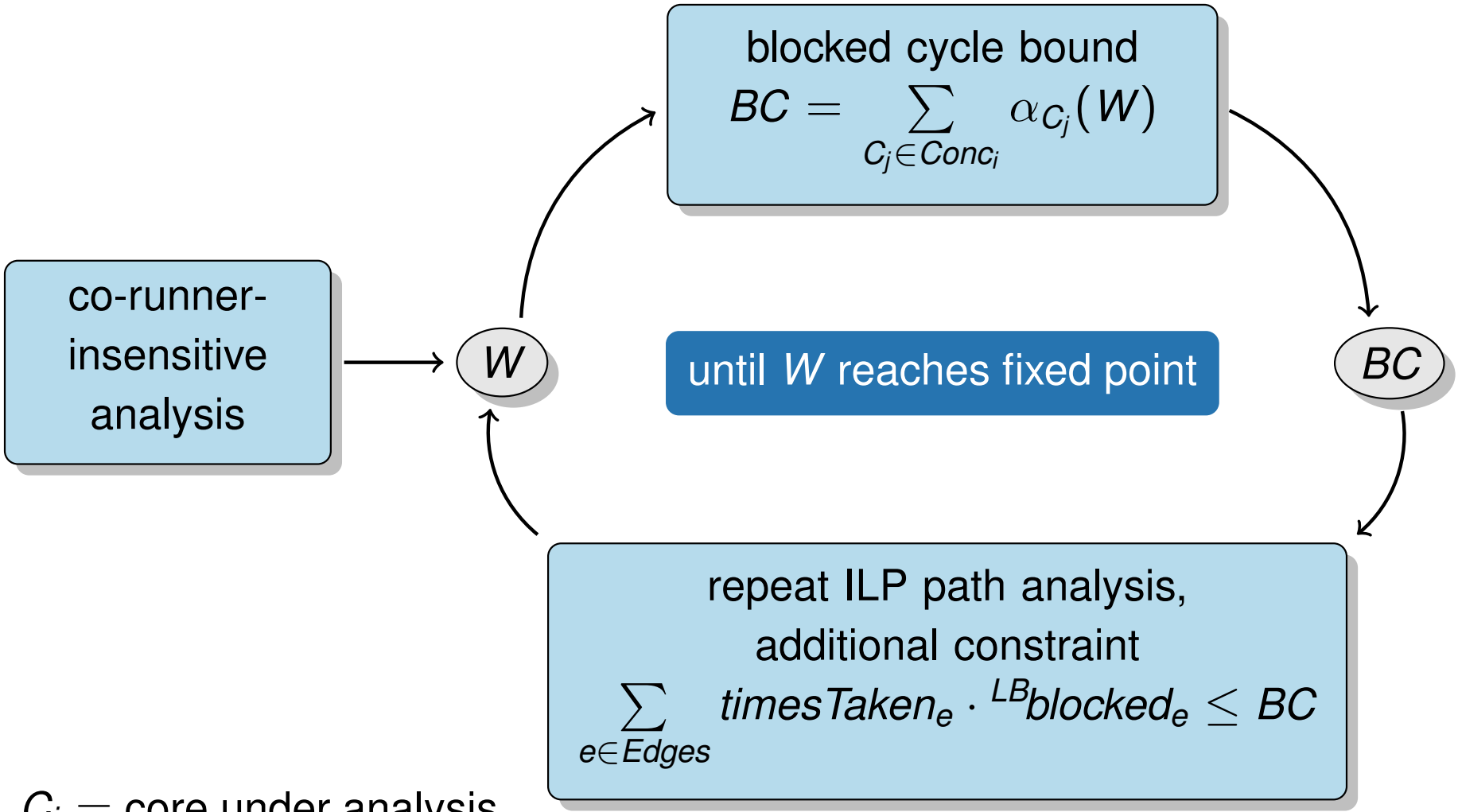


C_i = core under analysis

$Conc_i = Cores \setminus \{C_i\}$

$\alpha_{C_j}(W)$ = upper bound on number of access cycles of core C_j in W cycles

Iterative Co-Runner-Sensitive Analysis



C_i = core under analysis

$Conc_i = Cores \setminus \{C_i\}$

$\alpha_{C_j}(W)$ = upper bound on number of access cycles of core C_j in W cycles

Upper-Bounding Concurrent Access Cycles

Meaning of $\alpha_{C_j}(W)$

How many access cycles can core C_j perform at most in any interval of W time units?

- Our approach
  - Micro-architectural analysis of program(s) executed on C_j
  - **Generalized** implicit path enumeration
  - Exploit minimum inter-start time for precision
- Why generalize?
  - Implicitly enumerate all paths $\leq W$
  - Path may start / end at any program point
  - Path may span across multiple program runs
  - Path may span across different programs

Experimental Evaluation

Hardware configuration:

- Dual-core processor
- Out-of-order execution
- Instruction cache
- Data cache
- Round-robin bus arbitration

Setup for experiments:

- 19 programs of our benchmark suite
  - Those for which the co-runner-insensitive analysis needed $\leq$ 5 minutes
- Co-runner-sensitive analysis for all 19^2 possible pairs
  - 361 experiments
- In each experiment
  - One program per core
  - Minimum inter-start time of co-runner identical to its WCET bound

Iteration Example

- Benchmark `bsort100.c`
- Co-runner `janne_complex.c`

Iteration	WCET	reduction	runtime	peak mem.
0	5,197,213	0.000%	6m 13s	546M
1	4,869,654	6.303%	6m 16s	667M
2	4,750,724	8.591%	6m 21s	741M
3	4,708,728	9.399%	6m 43s	901M
4	4,695,003	9.663%	7m 6s	901M
5	4,687,438	9.809%	7m 29s	901M
6	4,686,534	9.826%	7m 32s	901M
7	4,686,534	9.826%	7m 33s	901M

Experimental Results

Statistical Distribution

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- All 361 experiments:

	min.	low. qurt.	median	upp. qurt.	max.
runtime	1m 2s	10m 18s	19m 15s	35m 54s	118m 39s
peak mem.	285M	820M	1559M	2293M	7154M

- WCET bound reduced for 42 experiments (11.6%):

	min.	low. qurt.	median	upp. qurt.	max.
iterations	3	6	10	17	20
WCET reduction	0.068%	1.945%	3.657%	7.243%	12.456%

Summary

Summary

Four key contributions:

- Modeling shared-bus interference by non-determinism
- Fast-forwarding of converged chains
- Iterative calculation of co-runner-sensitive WCET bounds
- Generalized implicit path enumeration

Conclusion

- WCET analysis for relatively complex multi-core processors
  - Possible, but
  - Runtime and memory consumption are high
- Co-runner-insensitive analysis is **scalable**
  - Almost independent of number of cores
- Co-runner-sensitive analysis is **more precise**
  - Up to 12.5% of WCET bound reduction

Recently Published

- Paper at RTNS [Jacobs et al., 2015]
  - November 4-6, 2015
  - WCET Analysis for Multi-Core Processors with Shared Buses and Event-Driven Bus Arbitration
- Check paper for details
  - E.g. for generalized ILP

Tool Chain

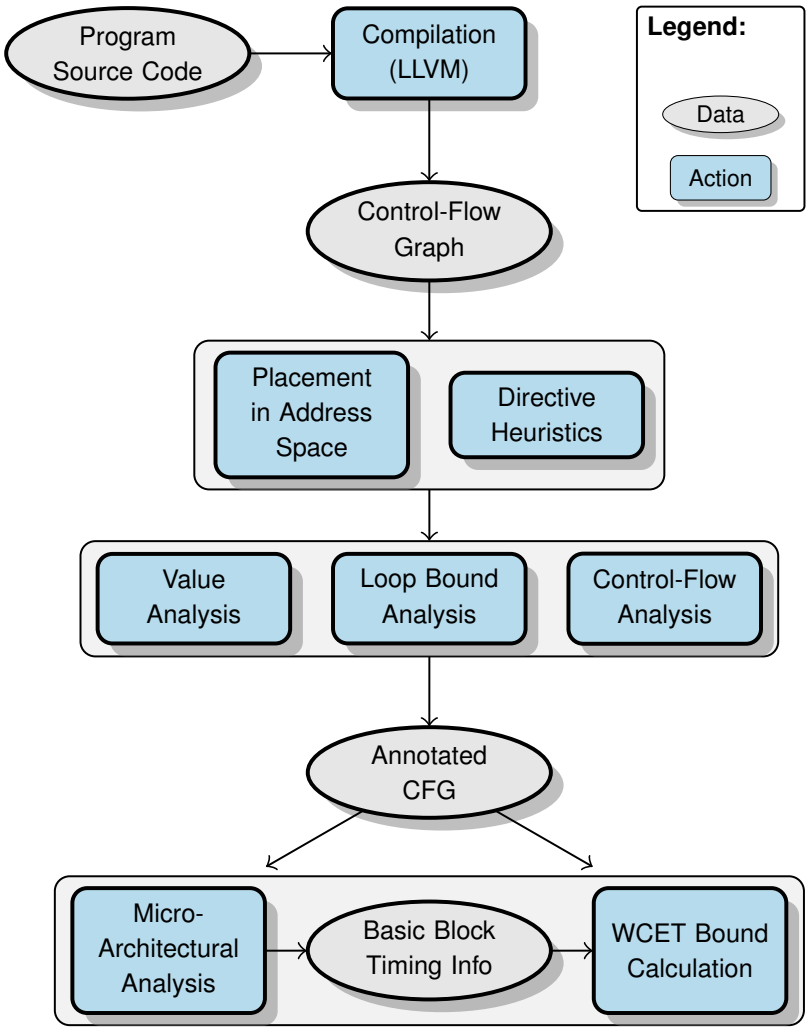
Tool Chain

Key facts:

- Own analysis framework
- Based on LLVM
  - Version 3.4
- Analysis on back-end IR
  - ARM back-end

Modular micro-architectural analysis:

- Pipeline execution
  - In-order
  - Out-of-order
- Different memory hierarchies
- Shared-bus interference



Analysis Paradigms

- Micro-architectural analysis
  - By **abstract interpretation**
  - [Thesing, 2004]
- State-sensitive execution graph
  - One edge per pair of in- and out-state of a basic block
  - "Prediction file/graph" in AbsInt[1] terminology
  - [Stein, 2010]
- Path analysis
  - By **implicit path enumeration** via ILP
  - Find longest path in execution graph
    - ★ for one program run [Li and Malik, 1995, Stein, 2010]
    - ★ generalized [Jacobs et al., 2015]
  - ILP solver CPLEX 12.4

[1] http://www.absint.com

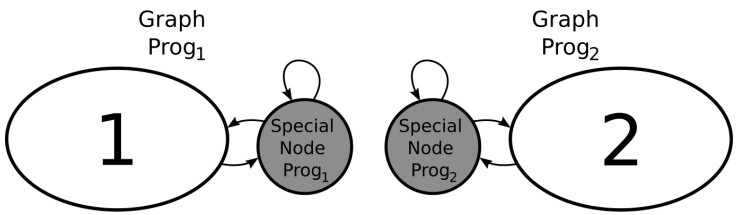
Setup for Co-Runner-Sensitive Analysis

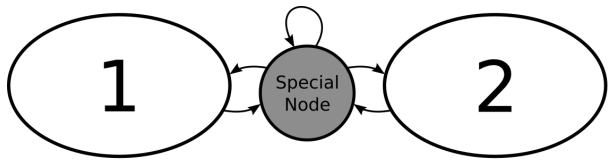
- One analysis tool instance per analyzed program
  - Potential for parallel execution
  - Reported runtime sequential
- Analysis instances exchange in each iteration
  - WCET bounds ($\Rightarrow$)
  - Upper bounds on access cycles ($\Leftarrow$)
- High runtime and memory consumption of generalized IPET
  - We use a time limit of 20 seconds per solver run
    - ★ Take best upper bound after limit exceeds
  - LP relaxation would also work
- Implementation restricted to one program per analyzed core
  - Upper bound on access cycles for a core calculated by one analysis instance
  - Each instance only argues about one program

Future Extension

Supporting multiple programs per core in co-runner-sensitive analysis

- Conceptual approach
  - Glue together execution graphs of multiple programs
  - Perform generalized IPET





- Planned implementation
  - Each tool instance dumps ILP formulations
  - Modularly combine ILP formulations
  - Actual iterations only call ILP solver

Supported Bus Arbitration Policies

Supported Bus Arbitration Policies

- **Requirement**:
  - Upper bound number of blocked cycles per access **independently of co-runners**
- Requirement holds for e.g.
  - Round-robin
  - First-come-first-serve
  - Time-division multiple access (though our approach pessimistic)
  - ...
- Thus, not yet supported
  - Priority-based arbitration
- **Additional requirement** for our co-runner-sensitive analysis:
  - Arbitration policy is work-conserving

Future Work: Time-Division Multiple Access

- Our approach implicitly assumes:
  - Each access might **just have missed** its slot
- Offset-based analysis can do much better!
  - Idea: track offsets w.r.t. the bus schedule per access
  - [Chattopadhyay et al., 2012]
- Our plan
  - Implement an offset-based analysis in our framework
  - Abstract offsets for scalability
    - ★ e.g. by intervals

Future Work: Priority-Based Arbitration

- Tweak micro-architectural analysis
  - Fast-forward to "∞" at convergence while blocked
  - If an access request does not converge until a threshold of blocked cycles
    $\Rightarrow$ Stop analysis, "potentially diverging"
- In path analysis, iterate from below
  - Start assuming no concurrent access cycles
  - Until least fixed point reached
- Make iterative analysis more precise
  - Priority-based arbitration is "more than" work conserving
  - At most one interfering access of lower priority per own access
- Other arbitration policies may also profit from least fixed point!

References I

Chattopadhyay, S., Kee, C., Roychoudhury, A., Kelter, T., Marwedel, P., and Falk, H. (2012).
A unified WCET analysis framework for multi-core platforms.
In *Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 99–108.

Jacobs, M., Hahn, S., and Hack, S. (2015).
Wcet analysis for multi-core processors with shared buses and event-driven bus arbitration.
In *Proceedings of the 23rd International Conference on Real-Time Networks and Systems*.

Kelter, T. and Marwedel, P. (2014).
Parallelism analysis: Precise WCET values for complex multi-core systems.
In Artho, C. and Ölveczky, P., editors, *Third International Workshop on Formal Techniques for Safety-Critical Systems*.

Li, Y.-T. S. and Malik, S. (1995).
Performance analysis of embedded software using implicit path enumeration.
In *Proceedings of the 32nd Annual ACM/IEEE Design Automation Conference*, pages 456–461.

Schranzhofer, A., Pellizzoni, R., Chen, J.-J., Thiele, L., and Caccamo, M. (2011).
Timing analysis for resource access interference on adaptive resource arbiters.
In *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 213–222.

Stein, I. J. (2010).
ILP-based path analysis on abstract pipeline state graphs.
PhD thesis.

References II

Thesing, S. (2004).
Safe and Precise WCET Determination by Abstract Interpretation of Pipeline Models.
PhD thesis.