Comparing Implicit Path Enumeration and Model Checking - based WCET Analysis

Benedikt Huber, Martin Schöberl
Vienna University of Technology

benedikt@vmars.tuwien.ac.at
mschoebe@mail.tuwien.ac.at

Why consider Model Checking ?

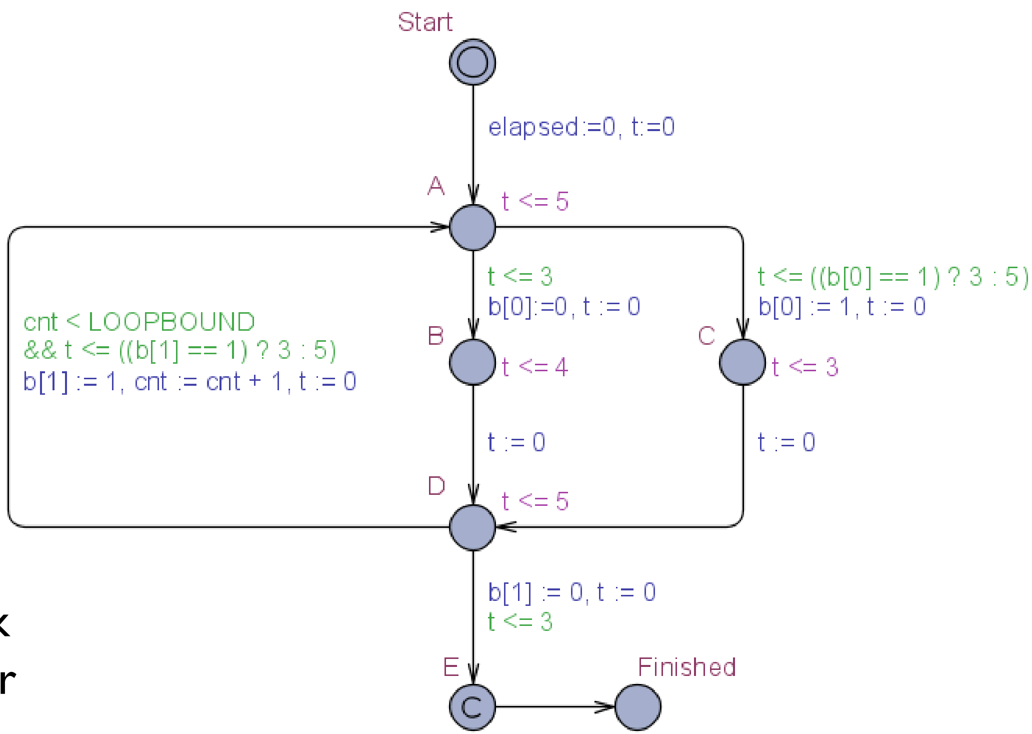
- Instruction timing depends on execution history
- ILP – based WCET calculation
  - Expressive constraints, efficient solvers
  - Needs good abstractions and/or graph duplication to take execution history into account
- Model Checking
  - Use a model checker to *calculate* a WCET bound
  - States generated on the fly, provide execution context
  - No need to enumerate all paths
  - Easy to model hardware
  - Reports worst-case path

Determining the WCET using UPPAAL

- Control flow graph → Timed Automaton
- Clocks represent elapsed time (global, basic block)
- Bounded integer variables
  - Loop Counters
  - Hardware State
- Guards on clocks and variables
  - Model instruction timing
  - Exclude infeasible paths
- Verify whether the task always finishes within T time units
  - Binary search with known upper bound

Example: Loop with Branch Prediction

Invariants
Timing Guards
Loop Guards
Clock Reset
Variable Update



elapsed .. total elapsed time
t .. time spend in basic block
b .. state of branch predictor
cnt .. loop counter

- *Verify:* `A[](Task.E imply elapsed ≤ WCET)`
- Find path: `E<>(Task.E && elapsed == WCET)`
- UPPAAL reports worst-case path: ACDABDACDABD…E

Implementation Context

- New version of our WCET analysis tool for Java processors
  - Target: The Java Optimized Processor (JOP)
  - But the approach also works for other platforms
- Analysis of Java byte code
  - Close to target platform, but much easier than assembler
  - Analysis: Call graph, Dynamic Dispatch, Loop Bounds
- Common Tool infrastructure
  - CFG construction & analysis
  - Report generation
  - Microcode Analysis

Evaluation: IPET and Model Checking

- Target: JOP + *variable block method cache (FIFO replacement)*
- IPET
  - Static cache approximation
  - We use this property: If during the execution of some method, the cache is guaranteed not to overflow, each method is loaded at most once.
- Model Checking: Cache simulation
  - Cache is an array of bounded integer variables
  - Update on access, wait on miss
- Questions we wanted to answer
  - Is model checking in principle capable of handling our applications ?
  - Comparison of static cache approximation with cache simulation

Benchmark Results

JOP Apps	Methods	Calc. WCET	IPET (s)	Verify (s)
MatrixMult	3	1088497	0.01	0.23
CRC	6	191825	0.01	0.52
Lift	13	8355	0.01	0.18
UdpIp	28	129638	0.04	1.78
Kfl (8 blocks)	46	37963	0.13	31.77
Kfl (1 block)				0.57
Kfl (16 blocks)				Timeout

- Method Cache: Simulation and Static Approximation
  - Simulation does not scale well
  - On evaluation platform, approximation is good enough
    - +3% - +7% compared to simulation
    - Took much longer to develop

Conclusion and Discussion

- IPET as 'the standard method' is a good idea
- Model Checking ?
  - Use model checking for important code fragments
  - Combine with Implicit Path Enumeration
  - Well suited to distinguish tractable number of hardware states
- UPPAAL has a nice abstraction for time
  - But only simple integer variables for hardware components
  - Binary search could be eliminated
- Future Work
  - Apply model checking to JOP multiprocessor
  - Work on other processors

Thank you.