

6.b WCET analysis

Credits to Enrico Mezzetti, PhD
(enrico.mezzetti@bsc.es)

Computing the WCET /1

- Why not measure the WCET of a task on its real HW?



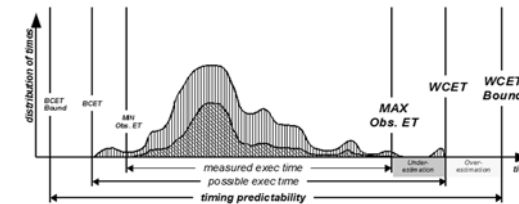
- Triggering the WCET by test is very difficult
 - Supplying input data that cover all possible program executions is intractable in practice
 - Worst-case initial state on modern HW is very difficult to determine
 - Complex pipelines (out-of-order execution)
 - Caches
 - Branch predictors and speculative execution

Worst-case execution time (WCET)

- For any input data and all initial logical states
 - So that all *execution paths* of the program are covered
- For any hardware state
 - So that the worst-case *execution conditions* are in effect
- Measurement-based WCET analysis
 - On either the real HW or a cycle-accurate simulator
 - The *high-watermark* value can be \ll WCET
- Static WCET analysis
 - Uses an abstract model of the HW and of the program

Computing the WCET /2

- Exact WCET not generally computable (\sim the *halting problem*)
- Yet, WCET *bounds* are essential to feasibility analysis
 - Which must be *safe* to upper bound all possible executions
 - Which be *tight* to avoid costly over-dimensioning



Static WCET analysis / 1

- To analyze a program without executing it
 - Needs an *abstract model* of the target HW
 - As well as the binary executable of the program
- Execution time depends on the program's control flow and the HW behavior
 - **High-level analysis** addresses the program behavior
 - *Control flow analysis* builds a control flow graph (CFG)
 - **Low-level analysis** determines the timing cost of individual instructions on the abstract model of the HW
 - Not constant for modern HW
 - Must be aware of the HW inner workings (pipeline, caches, etc.)

Implicit path enumeration technique

- The program's CFG is augmented with flow graph constraints
- The WCET is computed with integer linear programming or constraint programming
- $WCET = \max\{\sum_i x_i \times t_i\}$
 - x_i is the *execution frequency* of CFG edge i
 - t_i the *execution time* of CFG edge i

CFG	Flow graph constraints
	$x_1 = 1$
	$x_1 + x_8 = x_2$
	$x_2 = x_3 + x_4$
	$x_3 = x_5$
	$x_4 = x_6$
	$x_5 + x_6 = x_7$
	$x_7 = x_8 + x_9$
	$x_2 \leq LB * x_1$

Static WCET analysis / 2

Linear program

$$\max T = x_1 \cdot C_1 + x_2 \cdot C_2 + x_3 \cdot C_3 + x_4 \cdot C_4 + x_5 \cdot C_5 + x_6 \cdot C_6$$

$$x_1 = 1 \quad x_2 = x_3 \quad x_3 = x_4 \quad x_4 = x_5 \quad x_5 = x_6$$

$$x_6 = x_7 \quad x_7 = x_8 \quad x_8 = x_9 \quad x_9 = 1 + x_{10}$$

$$C_1 = -; C_2 = -; C_3 = -; C_4 = -; C_5 = -; C_6 = -;$$

$$x_{10} \leq 255 \cdot x_{11}; \quad x_{11} \leq 0.5 \cdot x_{12};$$

ILP solver

solution

$$T_{max} = \dots (WCET)$$

with:

$$x_1 = \dots \quad x_2 = \dots$$

$$x_3 = \dots \quad x_4 = \dots$$


$$x_5 = \dots \quad x_6 = \dots$$

Static WCET analysis / 3

- **High-level analysis / 1**
 - Must analyze all possible execution paths of the program
 - Builds the CFG as a superset of all possible execution paths
 - The unit of that analysis is the *basic block*
 - The longest sequence of program instructions with single entry and single exit (no branches, no loops)
 - Path analysis faces multiple challenges
 - *Input-data dependency*
 - *Infeasible paths*
 - *Loop bounds* and recursion depth
 - *Dynamic calls* through pointers

Static WCET analysis /4

■ *High-level analysis* /2

- Several techniques are employed to allow using IPET
 - *Control-flow analysis* to construct the CFG
 - First finding the basic blocks and then building the graph among them
 - *Data-flow analysis* to find loop bounds
 - *Value analysis* to resolve memory accesses
- Automated information extraction is insufficient 
 - User annotation of *flow facts* is needed
 - To facilitate detection of infeasible paths
 - To refine loop bounds
 - To define frequency relations between basic blocks
 - To specify the target of dynamic calls and referenced memory addresses

Static WCET analysis /6

■ *Low-level analysis* /2

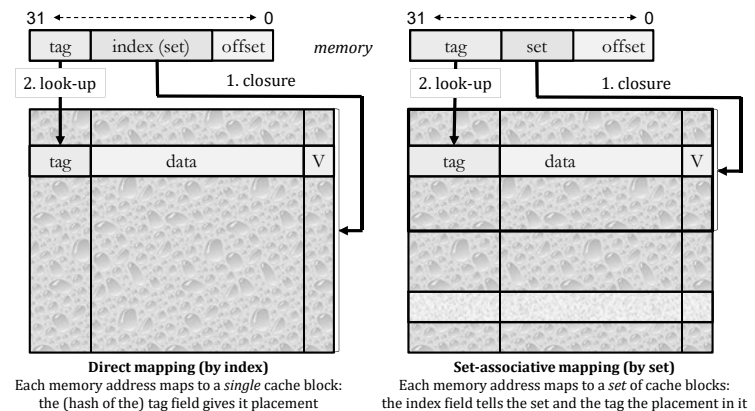
- Concrete HW states
 - Determined by the history of execution
 - Cannot compute all HW states for all possible executions
 - Invariant HW states are grouped into execution contexts
 - *Conservative overestimations* are made to reduce the research space
- *Abstract interpretation*
 - Computes abstract states and specific operators in the abstract domain
 - *Update function* to keep the abstract state current along the exec path
 - *Join function* to merge control flows after a branch
- Some techniques are specific to each HW feature

Static WCET analysis /5

■ *Low-level analysis* /1

- Requires abstract modeling of all HW features
 - Processor, memory subsystem, buses, peripherals, ...
 - It is *conservative*: it must never underestimate actual costs
 - All possible HW states should be accounted for
- HW modeling faces multiple challenges
 - *Precise modeling* of complex hardware is difficult
 - Inherent complexity (e.g., out-of-order pipelines)
 - Lack of comprehensive information (intellectual property, patents, ...)
 - Differences between specification and implementation (!)
 - *Exhaustive representation* of all HW states is computationally infeasible

Understanding the cache



Cache Associativity

Just as bookshelves come in different shapes and sizes, caches can also take on a variety of forms and capacities. But no matter how large or small they are, caches fall into one of three categories: direct mapped, n-way set associative, and fully associative.

Direct Mapped

Tag	Index	Offset

A cache block can only go in one spot in the cache. It makes a cache block very easy to find, but it's not very flexible about where to put the blocks.

2-Way Set Associative

Tag	Index	Offset

This cache is made up of sets that can fit two blocks each. The index is now used to find the set, and the tag helps find the block within the set.

4-Way Set Associative

Tag	Index	Offset

Each set here fits four blocks, so there are fewer sets. As such, fewer index bits are needed.

Fully Associative

Tag	Offset

No index is needed, since a cache block can go anywhere in the cache. Every tag must be compared when finding a block in the cache, but block placement is very flexible!

They all look set associative to me.

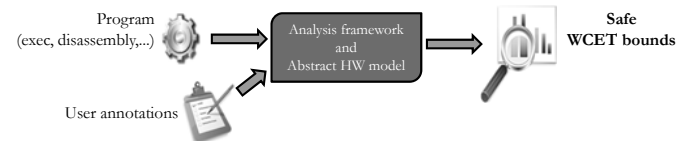
That's because they are! The direct mapped cache is just a 1-way set associative cache, and a fully associative cache of m blocks is an m -way set associative cache!

2017/18 UniPD - T. Vardanega Real-Time Systems 407 of 594

Static WCET analysis /7

- Safeness is at risk
 - When *local* worst case does not always lead to *global* worst case
 - Which is the case when **timing anomalies** occur
 - Complex hardware architectures (e.g., out-of-order pipelines)
 - Even improper design choices (e.g., inept cache replacement policies)
 - *Counter-intuitive* timing behavior
 - Faster execution of a single instruction causes *long-term* negative effects
 - Both are very difficult to account for in static analysis

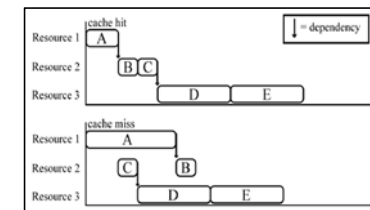
Static WCET analysis: the big picture



- Open problems
 - Can we always trust the abstract model of the HW?
 - How much overestimation do we incur?
 - Inclusion of infeasible paths
 - Overestimation is inevitable in abstract state computation
 - Intrinsic weakness of user annotations
 - Labor intensive and error prone

Timing anomaly: example

- Assume some dependence between instructions
- Shared resources (e.g. pipeline stages) and opportunistic scheduling of request servicing

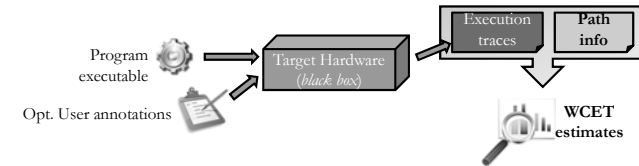


- Faster execution of A leads to a worse case overall execution owing to the order in which the instructions are executed

Hybrid analysis /1


- To obtain *realistic* (less pessimistic) WCET estimates
 - On the real target processor
 - On the final executable
 - Knowing that safeness not guaranteed (!)
- Hybrid approaches exploit
 - The measurement of *basic blocks* on the real HW
 - To avoid pessimism from abstract modeling
 - Static analysis techniques to combine the obtained measures
 - Knowledge of the program execution paths
- Newer approaches explore probabilistic properties (!)

Hybrid analysis: the big picture



- Open problems
 - Can we trust the resulting estimates?
 - Contingent on worst-case input and worst-case HW state
 - Consideration of infeasible paths
 - Needs the real execution environment or an identical copy of it
 - May cause serious cost impact and inherent difficulty of exactness

Hybrid analysis /2

- Approaches to collect timing information
 - *Software instrumentation*
 - The program is augmented with instrumentation code
 - Instrumentation effects the timing behavior of the program (aka the *probe effect*) and causes problems to deciding what's the final system
 - *Hardware instrumentation*
 - Depends on specialized HW features (e.g., debug interface)
- Confidence in the results contingent on the coverage of the executions and on the exploration of worst-case states
 - Exposed to the same problems as static analysis and measurement
 - *Worst-case state dependence is gone if HW response time is randomized* 



Summary

- The challenge of computing the WCET
- Static analysis
 - High-level analysis
 - Low-level analysis
- Hybrid analysis (measurement-based)

Selected readings

- R. Wilhelm et al. (2008)

The worst-case execution-time problem—overview of methods and survey of tools

DOI: 10.1145/1347375.1347389