7.a WCET analysis techniques

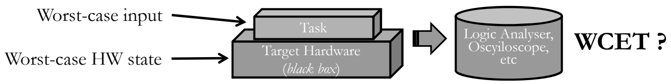
Credits to Enrico Mezzetti
(emezzett@math.unipd.it)

Worst-case execution time (WCET)

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

Computing the WCET – 1

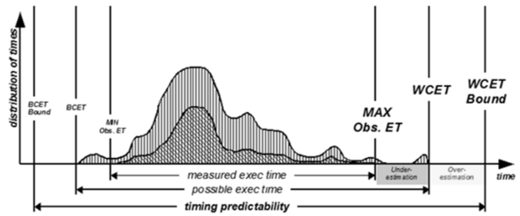
- Why not measure the WCET of a task on its real hardware?



- Triggering the WCET by test is very difficult
  - Worst-case input covering all executions of a real program is intractable in practice
  - Worst-case initial state is difficult to determine with modern HW
    - Complex pipelines (out-of-order execution)
    - Caches
    - Branch predictors and speculative execution

Computing the WCET – 2

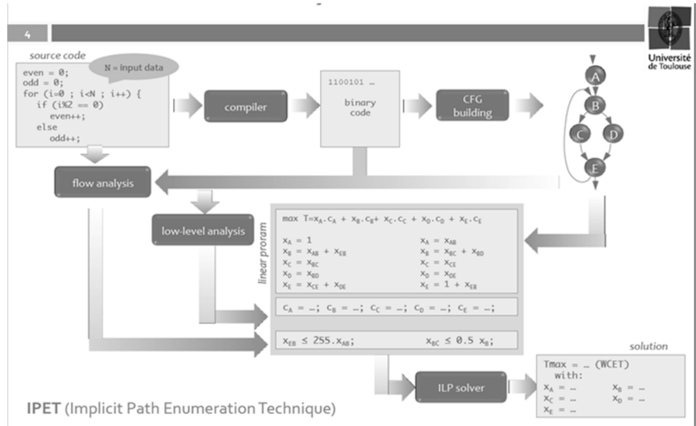
- Exact WCET not generally computable (~ the *halting problem*)
- A WCET estimate or *bound* are key to predictability
  - Must be *safe* to be an upper bound to all possible executions
  - Must be *tight* to avoid costly over-dimensioning



Static WCET analysis – 1

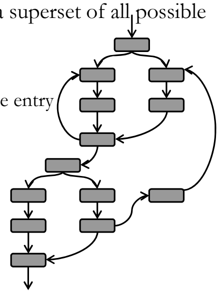
- Analyze a program without executing it
  - Needs an *abstract model* of the target HW
  - And the actual executable
- Execution time depends on execution path and HW
  - *High-level analysis* addresses the program behavior
    - Path analysis
  - *Low-level analysis* determines the timing behavior of individual instructions
    - Not constant for modern HW
    - Must be aware of the HW inner workings (pipeline, caches, etc.)

Static WCET analysis – 2



Static WCET analysis – 3

- ***High-level analysis***
  - Must analyze all possible execution paths of the program
    - Builds the Control-flow Graph (CFG) as a superset of all possible execution paths
    - *Basic block* is the unit of analysis
      - Longest sequence of instructions with single entry and single exit (no branches, no loops)
  - Challenges with path analysis
    - *Input-data dependency*
    - *Infeasible paths*
    - *Loop bounds* (and recursion depth)
    - *Dynamic calls* (through pointers)



Static WCET analysis – 4

- ***High-level analysis*** (cont'd)
  - Several techniques are used
    - Control-flow analysis to compute execution paths (CFG)
      - CFG unit: basic block
    - Data-flow analysis to find loop bounds
    - Value analysis to resolve memory accesses
  - Information automatically gathered is not exhaustive
    - 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 – 5

- ***Low-level analysis***
  - Requires abstract modeling of all HW features
    - Processor, memory subsystem, buses, peripherals, …
    - It is *conservative* : it must never underestimate actual timing
    - All possible HW states should be accounted for
  - Challenges with HW modeling
    - *Precise modeling* of complex hardware is difficult
      - Inherent complexity (e.g., out-of-order pipelines)
      - Lack of comprehensive information (copyrights, patents, …)
      - Differences between specification and implementation (!)
    - *Representation* of all HW states is computationally infeasible

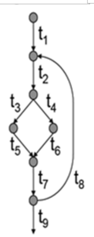
Static WCET analysis – 6

- ***Low-level analysis*** (cont'd)
  - 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

Implicit path enumeration

- The program structure is mapped into flow graph constraints
  - WCET computed with integer linear programming or constraint-solving techniques
- $WCET = \sum_i x_i \times t_i$
  - Where x_i is the execution frequency of CFG edge i
  - And t_i the execution time of CFG edge i

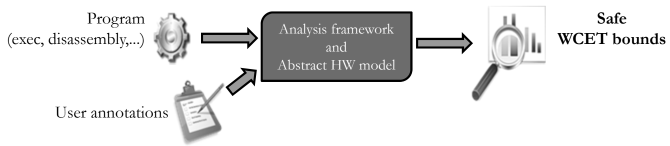
CFG



Flow constraints

x1 = 1
x1 + x8 = x2
x2 = x3 + x4
x3 = x5
x4 = x6
x5 + x6 = x7
x7 = x8 + x9
x2 <= LB * x1

Static WCET analysis: the big picture



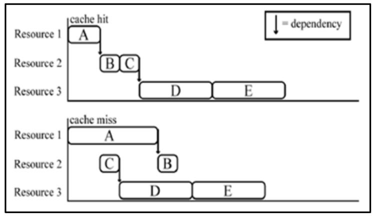
- Open problems
  - Can we always trust HW modeling?
  - How much overestimation do we incur?
    - Inclusion of infeasible paths
    - Overestimation intrinsic in abstract state computation
  - Weaknesses of user annotations
    - Labor intensive and error prone

Static WCET analysis – 7

- Safeness is at risk
  - When *local* worst case does not always lead to *global* worst case
  - When *timing anomalies* occur
    - Complex hardware architectures (e.g., out-of-order pipelines)
    - Even improper design choices (e.g., 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

Scheduling anomaly: example

- Some dependence between instructions
- Shared resources (e.g. pipeline stages) and opportunistic scheduling



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

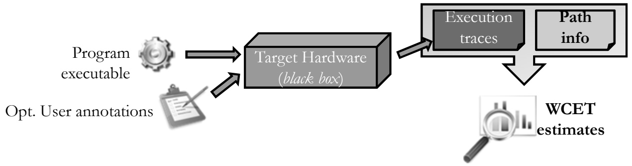
Hybrid analysis (measurement based) – 1

- To obtain *realistic* (less pessimistic) WCET estimates
  - On the real target processor
  - On the final executable
  - 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

Hybrid analysis (measurement based) – 2

- Approaches to collect timing information
  - *Software instrumentation*
    - The program is augmented with instrumentation code
    - Instrumentation effects the timing behavior of the program
      - A.k.a.: *probe effect*
      - Cannot be simply removed at end of analysis
  - *Hardware instrumentation*
    - Depends on specialized HW features (e.g., debug interface)
- Confidence in the results contingent on the coverage of the executions
  - Exposed to the same problems as static analysis and measurement

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
    - May cause serious cost impact and inherent difficulty of exactness

Summary

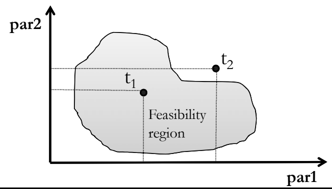
- The challenge of computing the WCET
- Static analysis
  - High-level analysis
  - Low-level analysis
- Hybrid analysis (measurement-based)

7.b Schedulability analysis techniques

Credits to Marco Panunzio
(panunzio@math.unipd.it)

Feasibility region

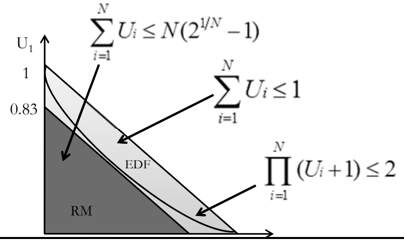
- The topological space that represents the set of feasible systems with respect to the workload model parameters
  - N-dimensional space with N-parameter analysis
  - Function of the timing parameters
  - Specific to the scheduling policy in force



t_1 is feasible
t_2 is not feasible

Advanced utilization tests

- *Hyperbolic bound* improves Liu & Layland utilization test
  - For systems with periodic tasks under FPS and DMPO
  - E. Bini and G. Buttazzo: "*A Hyperbolic Bound for the Rate Monotonic Algorithm*". Proceedings of the 13th ECRTS, 2001



Fine-grained response time analysis

$$R_i^{n+1} = B_i + CS1 + C_i + \sum_{j\in hp(i)} \left\lceil \frac{R_i^n + J_j^A}{T_j} \right\rceil (CS1 + C_j + TS + CS2) + I_{clock}^{R_i^n} + I_{extInt}^{R_i^n}$$

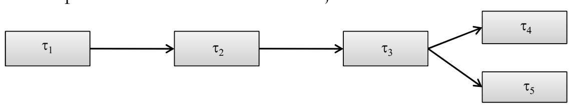
- *Blocking time* (resource access protocol or *kernel*)
- *"In" context switch*
- *"Activation" jitter*
- Time to issue a *suspension* call
- *"Out" context switch*
- Interference from the clock
- Interference from interrupts

$$R_1 = B_i + CS1 + C_i$$

$$R_i = R_i^n + J^W$$ ← *"Wake-up" jitter*

Transactions – 1

- Causal relations between activities
  - Consider information relevant to analysis that is not captured by classic workload models
    - Dependencies in the activation of jobs

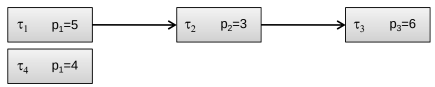


  - Originally introduced for the analysis of distributed systems
    - Also useful for the analysis of single-node "collaboration patterns"

Transactions – 2

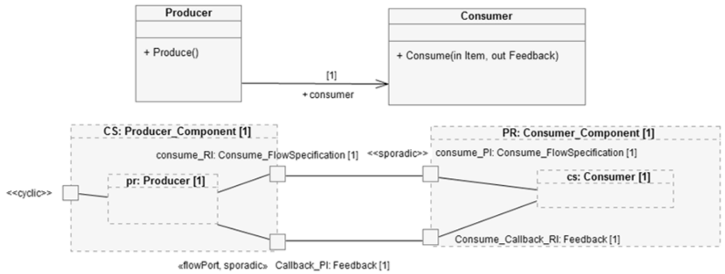
- Two main kinds of dependence
  - *Direct precedence* relation (e.g., producer-consumer)
    - τ_2 cannot proceed until τ_1 completes

  - *Indirect priority* relation
    - τ_4 does not suffer interference from τ_3 (under FPS and synchronous release of τ_1 and τ_4 for priorities increasing with values)

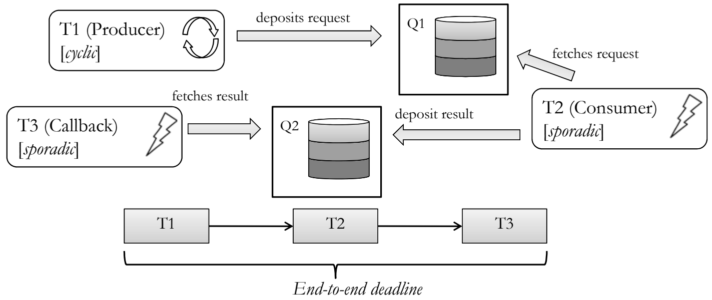


Example – 1

- A "*callback pattern*" to permit **in out** interactions between tasks in Ravenscar systems



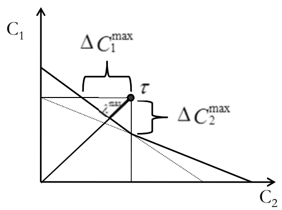
Example – 2



The feasibility of the *end-to-end response time* against this deadline is what matters (!)

Sensitivity analysis – 1

- Investigates the changes in a given system that
  - Improve the fit of an already feasible system
  - Make feasible an infeasible system



τ	Position of the system in the feasibility region
ΔC_1^{max}	Maximum feasible variation for the WCET of t_1 (negative in the example)
ΔC_2^{max}	Maximum feasible variation for the WCET of t_2 (negative in the example)

Sensitivity analysis – 2

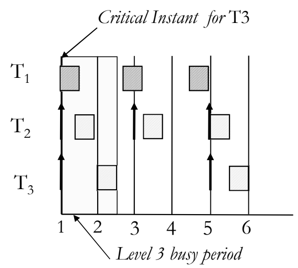
- Major computation complexity
- Theory still under development
  - Does not account for shared resources, multi-node systems, partitioned systems
- High potential
  - To explore solution space in the *dimensioning* phase of design
    - Presently only applicable to period/MIAT and WCET
  - To study the consequences of changes to timing parameters
    - To permit the inclusion of better functional value in the system
    - To renegotiate timing (or functional) parameters

MAST

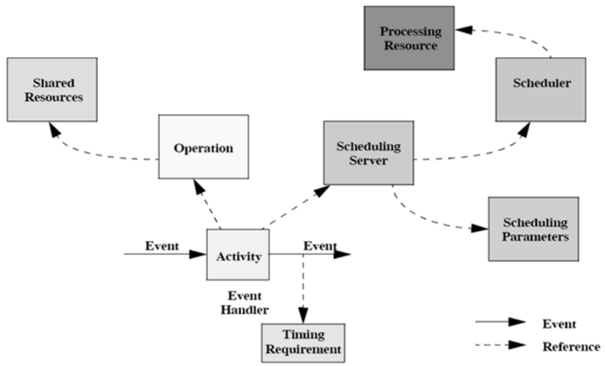
- Modeling and Analysis Suite for Real-Time Systems (MAST, http://mast.unican.es)
  - Developed at University of Cantabria, Spain
  - Open source
  - Implements several analysis techniques
    - For uni-processor and multi-processor systems
    - Under FPS or EDF

Classic workload model

$\mathbf{T_1}$	(Sporadic)	MIAT=1.750	WCET=0.500
$\mathbf{T_2}$	(Cyclic)	T=2.000	WCET=0.500
$\mathbf{T_3}$	(Cyclic)	T=4.000	WCET=0.500

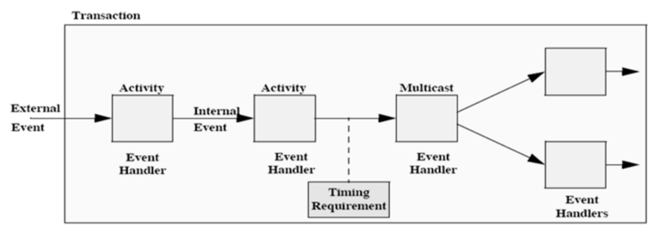


MAST – real-time model

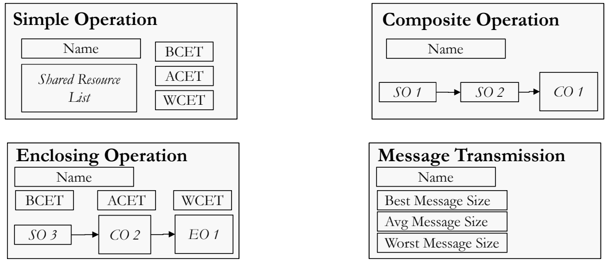


MAST – transaction

- To model causal relations between activities
  - Triggered by external events
    - Periodic, sporadic, aperiodic, etc…

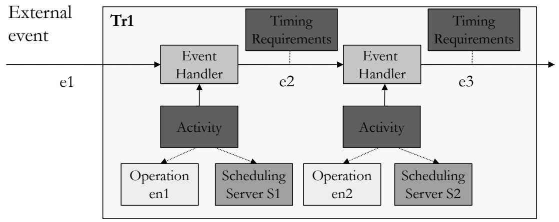


MAST – operations

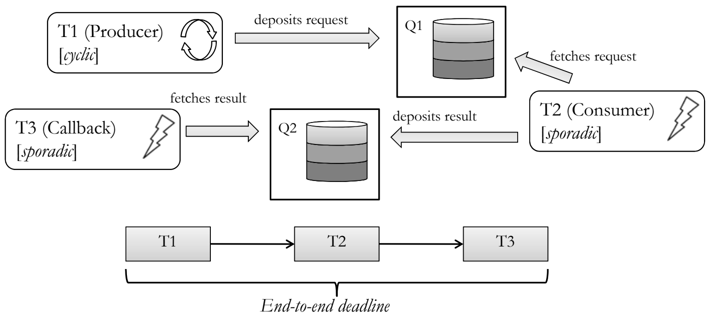


- The real-time model includes the description of all the operations in the system

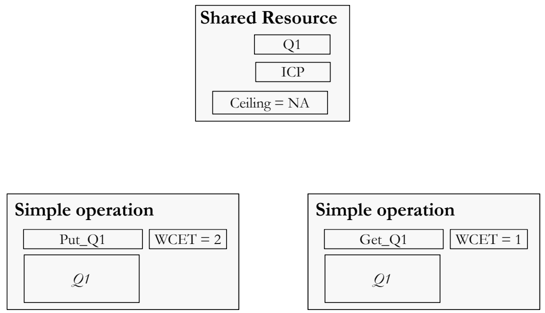
MAST – creation of a transaction



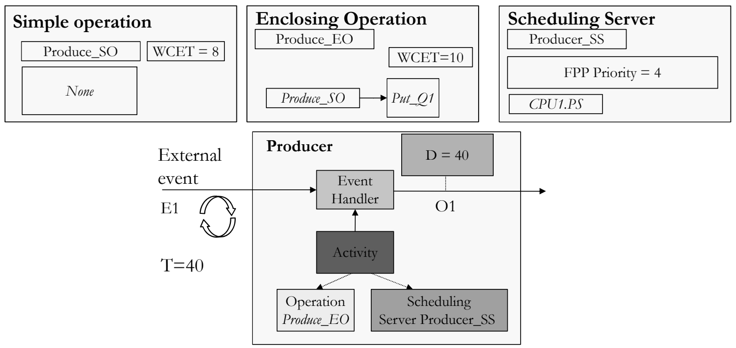
Example: Ravenscar callback



Example: shared resources in MAST



Example: modeling tasks in MAST



Example: timing attributes

Producer [1]	(C)	T_1=40	C_1=10	p_1=4
Consumer [2]	(S)	T_2=40	C_2=10	p_2=2
Callback [3]	(S)	T_3=40	C_3=5	p_3=5

Q1 Ceiling=4

Q2 Ceiling=5

Example: classic RTA results

Producer [1]	(C)	T_1=40	C_1=10	p_1=4
Consumer [2]	(S)	T_2=40	C_2=10	p_2=2
Callback [3]	(S)	T_3=40	C_3=5	p_3=5

Q1 Ceiling=4

Q2 Ceiling=5

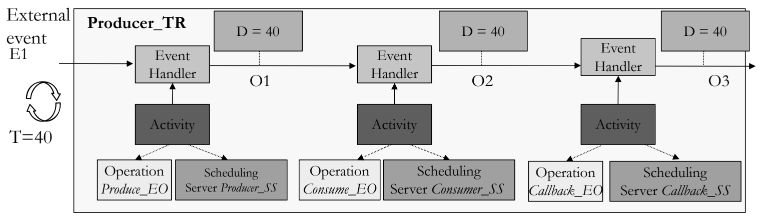
⇒ B_1=2 B_2=0 B_3=2

Classic RTA

R_1 = 17
R_2 = 25
R_3 = 7

} This misses out completely that T_3 is to be *preceded* by T_2 and T_1 (!)

Example: introducing transactions



Example: end-to-end analysis

Producer [1]	(C)	T_1=40	C_1=10	p_1=4
Consumer [2]	(S)	T_2=40	C_2=10	p_2=2
Callback [3]	(S)	T_3=40	C_3=5	p_3=5

Q1 Ceiling=4
Q2 Ceiling=5

⇒ B_1=2 B_2=0 B_3=2

Classic RTA	***Precedence and offset-based***
$R_1 = 17$	R_1 (Tr) = 12
$R_2 = 25$	R_2 (Tr) = 20
$R_3 = 7$	R_3 (Tr) = 27

⇐ ***Response time*** **relative to the beginning of the transaction!**

Summary

- Feasibility region
- Advanced utilization tests
- Fine-grained response time analysis
- Transactions
- Sensitivity analysis
- Example tool (MAST)