7.a WCET analysis techniques

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

Computing the WCET – 1

- Why not measure the WCET of a task on its real hardware?
  - Worst-case input
  - Worst-case HW state

- 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

Static 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 analysis – 2

- High-level analysis
  - Must analyze all possible execution paths of the program
    - Builds the Control-flow Graph (CFG)
    - Superset of all possible execution paths
    - Basic block is the unit of analysis
      - Sequence of instructions with no branches/loops
    - Challenges with path analysis
      - Input-data dependency
      - Infeasible paths
      - Loop bounds (and recursion depth)
      - Dynamic calls (through pointers)

Static analysis – 3

- High-level analysis (cont’d)
  - Several techniques are used
    - Control-flow analysis to compute execution paths
    - Data-flow analysis to find loop bounds
    - Value analysis to resolve memory accesses
    - CFG unit: basic blocks
  - 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 analysis – 4

**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 analysis – 5

**Low-level analysis (cont’d)**
- Concrete HW states
  - Determined by the execution history
  - Cannot compute all HW states for all possible executions
  - Invariant HW states are grouped into execution contexts
  - Conservative overestimation to reduce the research space
- Applied techniques
  - Abstract interpretation
    - Computes abstract state and specific operators in the abstract domain
    - Update function to update the abstract state along the exec path
    - Join function to merge control flow after a branch
  - Some techniques are specific to each HW feature

Static 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 analysis – 7

**Safety 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)
- Faster execution of A leads to a worse 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
- Safety 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 region that represents the set of feasible systems with respect to given 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 RM priority assignment

\[
\begin{align*}
0.83 & \leq \sum_{i=1}^{N} U_i (2^{2i} - 1) \\
1 & \leq \sum_{i=1}^{N} U_i \\
0 & \leq \sum_{i=1}^{N} (U_i + 1) \leq 2
\end{align*}
\]
**Fine-grained response time analysis**

\[
R^+_{i} = B(i) + CSI(i) + C(i) + \sum_{j \in \text{jobs}(i)} R^+_{i,j} + \sum_{j \in \text{jobs}(i)} J^+_{i,j}
\]

\[
R^+_{i,j} = B(i) + CSI(i) + C(i)
\]

\[
R^+_{i,j} = R^+_{i} + J^+_{i}
\]

*Blocking time (resource access protocol or kernel)*

*‘In’ context switch ‘Activation’ jitter*

*‘Out’ context switch ‘Interference from interrupts’*

*Time to issue a request ‘Interference from the clock’*

*‘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)
    - \( \tau_2 \) cannot proceed until \( \tau_1 \) completes
  - Indirect priority relation
    - \( \tau_4 \) does not suffer interference from \( \tau_3 \) (under FPS and synchronous release of \( \tau_1 \) and \( \tau_4 \) for priorities increasing with values)

**Example – 1**

- A “callback pattern” to permit in out interactions between tasks in Ravenscar systems

**Example – 2**

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

**Sensitivity analysis – 1**

- Position of the system in the feasibility region
- \( C_1 \)
- \( C_2 \)
- \( C_3 \)
- \( \Delta C^\text{min}_1 \)
- \( \Delta C^\text{max}_1 \)
- Maximum feasible variation for the WCET of \( \tau_1 \) (negative in the example)
- \( \Delta C^\text{min}_2 \)
- \( \Delta C^\text{max}_2 \)
- Maximum feasible variation for the WCET of \( \tau_2 \) (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

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

Critical Instant for \( T_3 \)

Level 3 busy period

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
**Example: 1 – Ravenscar callback**

- **T1 (Producer):** cyclic
- **T2 (Consumer):** sporadic
- **Q1:** fetches request
- **Q2:** deposits request
- **T3 (Callback):** sporadic
- **Q2:** fetches result
- **Q2:** deposits result

**Example: 2 – shared resources in MAST**

- **Simple operation:** WCET = 2
- **Put_Q1:** BCET
- **ACET
- **WCET**
- **Shared Resource List**

**Example: 3 – modeling tasks in MAST**

- **Simple operation:** WCET = 8
- **Produce_SO:** BCET
- **ACET
- **WCET**
- **Shared Resource List**

**Example: 4 – timing attributes**

- **Producer [1]:** (C) T1=40, C1=10, p1=4
- **Consumer [2]:** (S) T1=40, C1=10, p2=2
- **Callback [3]:** (S) T1=40, C1=5, p3=5

- **Q1:** Ceiling=4
- **Q2:** Ceiling=5

**Example: 5 – classic RTA results**

- **Producer [1]:** (C) T1=40, C1=10, p1=4
- **Consumer [2]:** (S) T1=40, C1=10, p2=2
- **Callback [3]:** (S) T1=40, C1=5, p3=5

- **Q1:** Ceiling=4
- **Q2:** Ceiling=5

- **Classic RTA**

  - **B1 = 2**
  - **B2 = 0**
  - **B3 = 2**

  This misses out completely that T3 is to be preceded by T2 and T1.
Example: 6 – introducing transactions

**External event E1**

- **Producer_TR**
  - D = 40
  - Operation: Produce_EO
  - Activity: Producer_TR

- **Event Handler**

- **Scheduling Server Producer_SS**
  - D = 40

- **Operation**
  - T = 40

- **Consume_EO**
  - **Activity**

- **Event Handler**

- **Scheduling Server Consumer_SS**
  - D = 40

- **Operation**
  - T = 40

- **Callback_EO**
  - **Activity**

- **Event Handler**

- **Scheduling Server Callback_SS**
  - D = 40

---

Example: 7 - end-to-end analysis

**Precedence and offset-based**

- **R1 (Tr) = 12**
- **R2 (Tr) = 20**
- **R3 (Tr) = 27**

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

- **B1 = 2**
- **B2 = 0**
- **B3 = 2**

**Classic RTA**

- **R1 = 17**
- **R2 = 25**
- **R3 = 7**

**Example tool (MAST)**

---

**Summary**

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