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 $\leq$ 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

2012/13 UniPD / T. Vardanega

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 overestimation 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 \( i \)

\[
\begin{align*}
\text{CFG} & \quad \text{Flow constraints} \\
\begin{array}{c|c}
\hline
\text{CFG} & \text{Flow constraints} \\
\hline
x_1 & = 1 \\
x_1 + x_8 = x_2 \\
x_2 & = x_3 + x_4 \\
x_3 & = x_5 \\
x_4 & = x_6 \\
x_5 + x_6 = x_7 \\
x_7 & = x_8 + x_9 \\
x_2 & \leq \text{LB} \times x_1 \\
\end{array}
\end{align*}
\]

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

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 affects 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

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

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

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

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
Advanced utilization tests

- Hyperbolic bound improves Liu & Layland utilization test
  - For systems with periodic tasks under FPS and DMPO

\[ \sum_{i=1}^{N} U_i = \frac{N(2^{2/N} - 1)}{N} \]

\[ \sum_{i=1}^{N} U_i \leq 1 \]

\[ \prod_{i=1}^{N} (U_i + 1) \leq 2 \]

Fine-grained response time analysis

\[ R_i = B_i + CS_i + C_i + \sum_{j \in \text{jobs}(i)} \left( \frac{R_j^E + J_j^E}{T_j} \right) \]

Blocking time (resource access protocol or kernel)

- "In" context switch
- "Out" context switch
- Interference from the clock
- Interference from interrupts

\[ R_i = R_i^E + J_i^E \]

"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)
    - \( t_2 \) cannot proceed until \( t_1 \) completes
  - Indirect priority relation
    - \( t_4 \) does not suffer interference from \( t_3 \) (under FPS and synchronous release of \( t_1 \) and \( t_4 \) for priorities increasing with values)
Example – 1

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

Example – 2

- T1 (Producer) [cyclic]
- T2 (Consumer) [sporadic]
- T3 (Callback) [sporadic]
- Q1
- Q2
- End-to-end deadline

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

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

  - 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

<table>
<thead>
<tr>
<th>Task</th>
<th>Interval (MIAT)</th>
<th>WCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁</td>
<td>1.750</td>
<td>0.500</td>
</tr>
<tr>
<td>T₂</td>
<td>2.000</td>
<td>0.500</td>
</tr>
<tr>
<td>T₃</td>
<td>4.000</td>
<td>0.500</td>
</tr>
</tbody>
</table>

Critical Instant for T₃

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

<table>
<thead>
<tr>
<th>Simple Operation</th>
<th>Composite Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Shared Resource</td>
<td>SO 1</td>
</tr>
<tr>
<td>List</td>
<td>SO 2</td>
</tr>
<tr>
<td></td>
<td>CO 1</td>
</tr>
<tr>
<td></td>
<td>EO 1</td>
</tr>
<tr>
<td>BCET</td>
<td>SO 1</td>
</tr>
<tr>
<td>ACET</td>
<td>SO 2</td>
</tr>
<tr>
<td>WCET</td>
<td>CO 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enclosing Operation</th>
<th>Message Transmission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>BCET</td>
<td>Best Message Size</td>
</tr>
<tr>
<td>ACET</td>
<td>Avg Message Size</td>
</tr>
<tr>
<td>WCET</td>
<td>Worst Message Size</td>
</tr>
<tr>
<td>SO 1</td>
<td>SO 2</td>
</tr>
<tr>
<td></td>
<td>SO 3</td>
</tr>
<tr>
<td></td>
<td>CO 1</td>
</tr>
<tr>
<td></td>
<td>EO 1</td>
</tr>
</tbody>
</table>

Example: Ravenscar callback

- T1 (Producer) deposits request
- T2 (Consumer) fetches request
- T3 (Callback) deposits result
- Q1
- Q2

Example: shared resources in MAST

- Simple operation
  - Put, Q1: WCET = 2
  - Get, Q1: WCET = 1

- Shared Resource
  - Q1
  - ICP
  - Ceiling = NA
Example: modeling tasks in MAST

<table>
<thead>
<tr>
<th>Simple operation</th>
<th>Enclosing Operation</th>
<th>Scheduling Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Producer SO</td>
<td>Producer EO</td>
<td>Producer SS</td>
</tr>
<tr>
<td>WCET = 8</td>
<td>WCET = D</td>
<td>WCET = D</td>
</tr>
</tbody>
</table>

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  $B_1=2$  $B_2=0$  $B_3=2$
Q2 Ceiling=5

Classic RTA

$R_1 = 17$  
$R_2 = 25$  
$R_3 = 7$

This misses out completely that $T_3$ is to be processed by $T_2$ and $T_1$ (§)

Example: introducing transactions

Producer TR
Consumer
Callback

$D = 40$

T=40

External event E1

Producer TR
Consumer
Callback

$D = 40$
**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) \( T_3 = 40 \) \( C_3 = 5 \) \( p_3 = 5 \)

\[ \begin{align*}
Q1 & \text{ Ceiling=} 4 \\
\text{B1} & = 2 \quad \text{B2} & = 0 \quad \text{B3} & = 2
\end{align*} \]

Classic RTA  Precedence and offset-based

\[ \begin{align*}
R_1 & = 17 \\
R_2 & = 25 \\
R_3 & = 7
\end{align*} \]

\[ R_1 \text{ (To)} = 12 \quad R_2 \text{ (To)} = 20 \quad R_3 \text{ (To)} = 27 \]

**Summary**

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