Worst-Case Execution Time Analysis from UML-based RT/E Applications

Chokri Mraidha, Sébastien Gérard, François Terrier, David Lugato

Chokri.Mraidha@cea.fr
Introduction

PhD student in the Software for Process Safety Laboratory in the French Atomic Energy Agency (CEA)

- Director: Judith Benzakki
- Advisor: Sébastien Gérard

PhD topic

- Optimized software synthesis in model driven development of real-time embedded systems
- Quantitative analysis of models
  - Execution time
Plan of the presentation

- Context
- Related works on WCET analysis methods
- A hybrid method for WCET estimation
- An analysis-based method for WCET estimation
- Conclusions and perspectives
Context

**Domain:** Distributed Real-time and Embedded Systems (DRES)

**Accord/UML**

- An MDD approach for DRES
  - Based on the UML
  - RTE profile specializing the SPT profile
  - Set of modeling rules for RT applications development
- Accord/UML tool support
  - Implementation of the Accord/UML profile
  - Accord framework
    - Kernel and virtual machine (UNIX, Linux, VxWorks)
Highlight: an execution model based on RTOs

RTO: An autonomous computing entity!

- **Message processing & attribute access control**
- **Operation calls**
  - External interface
    - operation 1
    - operation 2
    - ...
  - Method code
    - Attributes

Tasks:
- method 1
- method 2
- method 1
Communication mechanisms and task model

- **Two main schemes of communication**
  - Synchronous or asynchronous

- **Accord/UML tasking model underlying Real-Time Object concept**
  - General computational model:
    - One task per operation call
    - Real-time features
      - TimeRef, Deadline, ready time, period and period number
  - Scheduling policy
    - EDF
Schedulability analysis on UML models

Task model calculus & schedulability analysis

1. Accord/UML-models with RT specification
2. Adapted application models + all required RT specification for schedulability analysis
3. Real-time symbolic execution tree of the application
4. Formal behavioral analyser tool (AGATHA)

If not schedulable system:
- Model correction
- Real-time constraints adaptation

Mapping between UML models and AGATHA’ formal language with clear modelling of the scheduling policy

Need the Worst Case Execution Time of each operation (~ task)
Plan of the presentation

- Context
- Related works on WCET analysis methods
  - A hybrid method for WCET estimation
  - An analysis-based method for WCET estimation
- Conclusions and perspectives
Analysis-based methods (static)

- No execution of the program
- A two-phases process
  - High-level analysis: computes all execution paths
  - Low-level analysis: estimates execution time of these paths
- Requires
  - Task specification
  - Timing model of the microprocessor
- Result
  - An upper bound of the WCET: safe
- Main issues
  - Avoid overestimation of the WCET
  - Requires an elaborated timing model of the microprocessor
  - Not easily retargetable (timing model)
Measurement-based methods (dynamic)

- Measurement of the execution time at runtime
  - Find input values having the longest execution time
- A two-phases process
  - Find input values covering all execution paths
  - Measure execution time for every input value at runtime
- Requires
  - A set of input values
  - A binary of the task to measure its execution time
- Result
  - A lower-bound of the WCET
- Main issues
  - Find all input values for measurement
    - Impossible for infinite domains
<table>
<thead>
<tr>
<th></th>
<th>Analysis-based approaches</th>
<th>Measurement-based approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>High-level analysis:</td>
<td>Execution time</td>
</tr>
<tr>
<td></td>
<td>execution path computing</td>
<td>measurement for given</td>
</tr>
<tr>
<td></td>
<td>exhaustive</td>
<td>input values</td>
</tr>
<tr>
<td>--</td>
<td>Low-level analysis:</td>
<td>Find input values for</td>
</tr>
<tr>
<td></td>
<td>execution time estimation</td>
<td>measurement</td>
</tr>
<tr>
<td></td>
<td>overestimations</td>
<td>covering all execution</td>
</tr>
</tbody>
</table>
Plan of the presentation

- Context
- Related works on WCET analysis methods
- A hybrid method for WCET estimation
- An analysis-based method for WCET estimation
- Conclusions and perspectives
A hybrid method for WCET estimation

- **A two-phases approach**

  - **A static analysis phase**
    - High-level analysis of static analysis methods
    - Automatic test cases generation
      - Tests covering all possible execution paths
      - Analysis techniques: model checking, constraint-based techniques, symbolic execution…

  - **A measurement-based phase**
    - Runtime measurement
      - Measurement using tests of the previous phase
Hybrid approach overview

Set of automata communicating by rendezvous

Labeled transitions system

Model transformation

Code generation with timing measurement routines

Path Conditions for every execution path (equivalence class of tests for each path)

Measurement of execution time

One test for every path (one representant of the equivalence class)

Constraints solver

J1 = (Vit=120, dist=150)
J2 = (Reg=true, Vit=55)

...
Advantages and limitations of the approach

- **Advantages**
  - Automatic process
    - No code or model annotation required from the user
    - Integrated into a development toolchain
  - No timing model required
    - Easily retargetable to new sw/hw components
  - Good scalability with symbolic execution

- **Limitations**
  - First order logic models (Presburger arithmetic)
    - Limitation due to the Agatha tool
    - Ongoing work to compute more complex models
  - Possible scalability improvements
    - Explore the longest path first ➔ execution tree pruning
    - Test selection criteria
    - Apply the method to sub models ➔ iterative process
  - Gives only a lower bound of the WCET (measurement)!
Plan of the presentation

- Context
- Related works on WCET analysis methods
- A hybrid method for WCET estimation
- An analysis-based method for WCET estimation
- Conclusions and perspectives
Model-based WCET static analysis overview

- Path analysis with symbolic execution (optimization phase)
  - Infeasible paths removal → execution tree pruning
- Model transformation (computation phase)
  - Assign symbolic WCET values to primitive actions of the action language
    → WCET_ReadAction; WCET_WriteAction; WCET_AddAction...
  - Compute symbolic WCET for every execution path
- WCET is obtained by replacing symbolic values by numerical values
Illustration with an example

Model transformation to add symbolic timing values of elementary actions

\[ x := a \]
\[ x := x + a \]
\[ x := b \]

\[ x := a \]
\[ [x \leq 10] \quad [x > 10] \]

\[ WCET := WCET_{\text{ReadAction}} + WCET_{\text{WriteAction}} \]
\[ WCET := WCET_{\text{ReadAction}} + WCET_{\text{WriteAction}} + WCET_{\text{TestAction}} \]

\[ WCET := WCET_{\text{AddAction}} + 2 \times WCET_{\text{ReadAction}} + 2 \times WCET_{\text{WriteAction}} + WCET_{\text{TestAction}} \]
Results

■ Advantages
  ➔ Good results for systems using simple microprocessors
    ✓ Sequential execution: no pipeline
    ✓ No cache
    ✓ Systems where architectural mechanisms harming time predictability are switched off (critical systems)

■ Drawback
  ➔ Overestimated WCET for systems using these architectural features (increasingly frequent)

→ Need to take into account architectural features to reduce overestimation
  ➔ Pipeline
  ➔ Cache
  ➔ Branch predictors
  ➔ Out of order execution
Perspectives: improvements

How to take into account architectural features?

- Timing models of microprocessors
  - Implement this model in a static analysis tool (Agatha in our case)
    - Not interesting because not flexible, not portable…
  - Extract architectural elements timing properties in a dedicated model
    - Platform Real-time Description Model

- What should this model contain?
  - Timing properties of architectural elements
    - pipeline, cache…
  - Accuracy of results increases with number of modeled elements
  - Stay at a some level of abstraction in order to be usable by tools
Perspectives: Model-driven WCET analysis

- Apply Model Driven Engineering tenets for WCET analysis
  - Task model
  - Platform RT Description Model
  - Model transformations to map the platform model on the task model → WCET Analysis Model

- Benefits
  - Portability
  - Reusability
  - Maintainability
  - Model-level automated process
    - Model transformations
    - We can analyse WCET for several platforms and choose the best suited one before implementation
Example: a simple pipelined architecture

- **No memory, no data cache**
  - All operands are in registers or in the instruction itself (constants)
- **No branch prediction mechanisms**
- **No out of order execution**
- **No short-circuit mechanisms**
  - A data dependency costs a pipeline stall of 2 cycles
  - Only RAW (Read After Write) data dependencies are possible
    - Occurs when \( \text{instr}_{n+1} \) needs to read data processed by \( \text{instr}_n \)
Example: a simple pipelined architecture (cont.)

- WCET
  - The cycle number of WR unit of the last instruction
- Model transformation rules
  - Computes cycle of WR for every instruction
  - $WR_n$: cycle of WR of instruction $n$
  - $WR_n = WR_{n-1} + 1$
  - $WR_n = WR_{n-1} + 1 + \Delta RAW$ if $RAW(n-1, n)$
Without taking into account the pipeline: sequential execution

\[ WCET = 4 \times 5 = 20 \text{ cycles} \]

A saving of 50%

\[ WCET = 4 \times 5 = 20 \text{ cycles} \]

Taking into account the architecture improves accuracy!

\[ WCET = 9 \]

For 4 instructions only!

For a very simple architecture
Plan of the presentation

- Context
- Worst-Case Execution Time methods
- A hybrid method for WCET estimation
- An analysis-based method for WCET estimation
- Conclusions and perspectives
Conclusions

- **Estimation of WCET from UML-based RT models**
  - A hybrid approach
    - ✓ Provides lower-bound of WCET
    - ✓ Precise results useful for soft real-time systems
  - An analysis approach
    - ✓ Safe bounds
    - ✓ Model-driven analysis technique
      - Task model + Platform RT Description model ➔ WCET analysis model
      - » Flexible, portable technique
      - Model Transformation process
      - The architecture becomes a parameter of the analysis (SoC)

- **Perspectives**
  - Take into account more architectural features
    - ✓ Refine the model transformation rules by taking into account the data cache