WCET Analysis of Java Bytecode Featuring Common Execution Environments

Kasper Søe Luckow
Joint work with
Christian Frost, Casper Svenning Jensen, and Bent Thomsen

Department of Computer Science,
Aalborg University

JTRES, September 2011
The Worst Case Execution Time (WCET) forms an integral component in schedulability analysis and, thus, in determining temporal correctness.

For hard real-time systems, the estimated WCETs must be safe, that is, must be at least as high as the actual WCET.

Safe WCET (and schedulability) analysis is currently possible on hardware implementations of the JVM using WCA\(^1\) for JOP (and SARTS\(^2\) for schedulability analysis).

Portable WCET analysis has been proposed by XRTJ, but relies on a measurement-based technique for low-level WCET analysis.

We want to extend WCET analysis of hard real-time Java systems to execution environments with software implementations of the JVM and common embedded processors using a static approach.

\(^1\)http://www.jopdesign.com
\(^2\)http://sarts.boegholm.dk/
TetaJ at a Glance

- Capable of conducting WCET analysis of Java bytecode taking into account a software implementation of the JVM, and common embedded hardware

**Design goals**

- **Flexibility**
  - To address Java’s portability goal

- **Safe and precise WCET estimates**
  - Safety: necessitated by hard real-time systems
  - Precision: to make fewest computational resources suffice while still being temporally correct

- **Applicable for iterative development**
  - Analyses on method level
  - Analysis time and memory consumption are reasonably low
TetaJ employs a static analysis approach for WCET estimation.

The program analysis problem of determining WCET is viewed as a model checking problem.

Java bytecode program, JVM, and hardware are modeled as a Network of Timed Automata (NTA) (UPPAAL$^3$).

Program and execution environment are viewed as three independent layers of models:
- Interact through pre-defined interfaces
- Achieves high flexibility

---

3 http://www.uppaal.com
Architecture of TetaJ

- TetaJ Control Flow Graphs (TCFGs)
  - Achieves reusability
- Extendible support:
  - AVR $\rightarrow$ TCFG
  - Java Bytecode $\rightarrow$ TCFG
- CFG analyses for e.g.
  - Loop detection
  - Condition optimisation
- Transform TCFG to NTA
- Combine models into one
- Interact with UPPAAL to obtain WCET
UPPAAL Models

Initialisation Model

Program Model
UPPAAL Models Cont’d

JVM Model (excerpt)

C. Frost, C. S. Jensen, K. S. Luckow, and B. Thomsen
UPPAAL Models Cont’d

Java Bytecode Implementation

Hardware Models (From METAMOC)
Evaluation of Optimisations

<table>
<thead>
<tr>
<th>Optimisation</th>
<th>Analysis time</th>
<th>States explored</th>
<th>Memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>No optimisations</td>
<td>14h 51m 17s</td>
<td>41854143</td>
<td>3,905 MB</td>
</tr>
<tr>
<td>Only state space reduction</td>
<td>13h 33m 21s</td>
<td>41854143</td>
<td>2,426 MB</td>
</tr>
<tr>
<td>Only condition optimisation</td>
<td>1m 16s</td>
<td>53732</td>
<td>294 MB</td>
</tr>
<tr>
<td>Only template reduction</td>
<td>4h 46m 41s</td>
<td>41854143</td>
<td>3,851 MB</td>
</tr>
<tr>
<td>All optimisations</td>
<td>19s</td>
<td>57553</td>
<td>144 MB</td>
</tr>
</tbody>
</table>

- Based on a trivial program containing, among others, a loop and variable assignments
- Optimisations decrease resources needed substantially
  - Analysis time decreases from 15 hours to 19 seconds
  - Memory consumption decreases from 3.9GB to 144MB
- Template reduction and condition optimisations are the prime contributors to this decrease
Evaluation of Safety and Precision

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Meas. WCET</th>
<th>TetaJ WCET</th>
<th>Pessimism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterative Fibonacci</td>
<td>46,642</td>
<td>46,933</td>
<td>0.6%</td>
</tr>
<tr>
<td>Factorial</td>
<td>39,726</td>
<td>40,939</td>
<td>3.1%</td>
</tr>
<tr>
<td>Reverse Ordering</td>
<td>64,436</td>
<td>81,919</td>
<td>27.1%</td>
</tr>
<tr>
<td>Bubble Sort</td>
<td>907,103</td>
<td>2,270,401</td>
<td>150.3%</td>
</tr>
<tr>
<td>Binary Search</td>
<td>54,430</td>
<td>99,301</td>
<td>82.4%</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>849,353</td>
<td>3,740,769</td>
<td>440.4%</td>
</tr>
</tbody>
</table>

- Safety and precision are evaluated by using a measurement-based method.
  - Formally not safe. Hence, this evaluation only provides indications.
- Generally: TetaJ WCET > Meas. WCET → indicates safety.
- Precision: as low as 0.6%
- Imprecise results, e.g. Bubble Sort (150%), are attributed lack of annotation possibilities for stating dependencies among loop bounds.
Mine Pump

- Classic text-book example of a real-time system
- A water pump is responsible for removing excess water to avoid endangering the lives of the mine workers

Requirements

- Pump starts whenever the water level reaches the high level marker
- Pump stops whenever the water level reaches the low level marker
- Pump must not run when the methane concentration is too high
- If the methane level is not critical, the mine pump must never be flooded

Task Period/Deadline

<table>
<thead>
<tr>
<th>Task</th>
<th>Period/Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methane</td>
<td>56ms</td>
</tr>
<tr>
<td>Water</td>
<td>40ms</td>
</tr>
</tbody>
</table>
The LEGO Mine Pump
The LEGO Mine Pump Cont’d

- Implemented in Java
- 429 lines of code
- 18 classes
- McCabe cyclomatic complexity of 6
- RTSJ, SCJ, PJ etc. have not been used
Execution Platform

- Based on a modified version of the Hardware near Virtual Machine (HVM)\(^4\) and the Atmel AVR ATmega2560 processor
- Representative execution platform for an embedded system using Java Bytecode

**HVM**

- Emphasises portability (currently has support for ATmega2560, CR16C, and x86)
- Supports systems with as low as 256 kB of flash, and 8 kB of RAM
- Employs iterative interpretation of Java Bytecode to machine code
- Originally not amenable to static WCET analysis

**Atmel AVR ATmega2560**

- Deterministic behaviour
- No caching, nor branch prediction
- Features a simple two-stage pipeline

\(^4\)http://www.icelab.dk
The Modified HVM

- Modifications of the HVM comprise e.g.:
  - Constant time analyse stage
  - Eliminating recursive solutions
  - Constant time type compatibility check

- The NTA of the modified HVM can be automatically constructed by the provision of its binary

- A timed automaton captures the conceptual model of iterative interpretation, that is, the fetch, analyse, and execute stages

- This automaton forms an NTA with the timed automata constructed from each of the Java Bytecode implementations
Applying TetaJ in the Case Study

<table>
<thead>
<tr>
<th>Task</th>
<th>Analysis time</th>
<th>Memory usage</th>
<th>TetaJ WCET</th>
<th>Meas. WCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methane 1</td>
<td>1m 19s</td>
<td>140 MB</td>
<td>41,644</td>
<td>8,901</td>
</tr>
<tr>
<td>Methane 2</td>
<td>1m 16s</td>
<td>105 MB</td>
<td>68,436</td>
<td>29,449</td>
</tr>
<tr>
<td>Methane 3</td>
<td>29s</td>
<td>75 MB</td>
<td>12,552</td>
<td>3,896</td>
</tr>
<tr>
<td>Water</td>
<td>6m 40s</td>
<td>271 MB</td>
<td>70,712</td>
<td>32,421</td>
</tr>
</tbody>
</table>

- Again TetaJ WCET > measured WCET
- Analysis times and memory consumptions are reasonably low
- A cyclic executive is constructed for scheduling the two tasks
  - Minor cycle is 8ms; major cycle is 280ms
  - A schedule can be constructed, thus we conclude that the system is schedulable (proof by construction)
Conclusion

- Shown that WCET analysis of Java Bytecode executed on common embedded hardware is feasible using model checking with UPPAAL
- From the case study, we have shown that TetaJ may be adopted in an iterative development method
  - Analysis time and resource consumption are reasonably low which can be attributed the optimisations
  - TetaJ provides analysis on method level by providing the fully qualified name of the method of interest
- We have indications that the WCET estimates obtained by TetaJ are safe
  - May therefore be appropriate for analysing hard real-time systems
- Precise WCETs can be obtained using TetaJ
Future Work

- Compare TetaJ with WCA
- Evaluate scalability of TetaJ
  - In general, apply TetaJ on more case studies
- Determine whether TetaJ still applies for systems that use e.g. the SCJ
- Merge TetaJ with SARTS