Worst-Case Execution Time Analysis using Model Checking and Static Analysis

Martin Toft
Ph.D. student
mt@cs.aau.dk

Department of Computer Science
Aalborg University

September 15, 2009
Real-Time Systems

- Systems that need to respond to real-life events in a timely manner
- Found in planes, rockets, robots, medical equipment, etc.
- Contains processes with associated worst-case execution times (WCETs) and deadlines
- The WCETs must be determined
Worst-Case Execution Times

- Underestimation entails unsafe systems
- Too much overestimation entails inefficient systems
The Problem

- Taking modern processor features into account
  - Caching
  - Pipelining
- Ensuring modularity
  - Easy reuse of models for common hardware parts
  - The user is able to experiment
  - Different trade-offs between accuracy and performance

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Instr. 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Instr. 2</td>
<td>Instr. 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Instr. 3</td>
<td>Instr. 2</td>
<td>Instr. 1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Instr. 4</td>
<td>Instr. 3</td>
<td>Instr. 2</td>
<td>Instr. 1</td>
</tr>
<tr>
<td>5</td>
<td>Instr. 5</td>
<td>Instr. 4</td>
<td>Instr. 3</td>
<td>Instr. 2</td>
</tr>
</tbody>
</table>
The General Method: WCET Analysis using Model Checking and Static Analysis

Abstract process model and value analysis

Abstract hardware model with caching and pipelining

Timed automata models for hardware components and process functions:

WCET 42 cycles
Prototype Implementation

- For the ARM920T processor
- Contains the ARM9TDMI core (also in ARM922T and ARM940T)
- Five stage pipeline
- Separate instruction and data caches
- Does not suffer from timing anomalies
  \[\Rightarrow\] OK to assume that local worst-case implies global worst-case
Prototype Implementation (cont.)

- Annotated Executable
  - disassemble (objdump, Dissy)
    - ARM assembly
      - value analysis (WALi)
  - Pipeline (UPPAAL model)
    - ARM-to-UPPAAL
      - Control Flow Graph (UPPAAL model)
      - combine
        - Complete model (UPPAAL model)
        - model check (UPPAAL)
          - WCET
- Main Memory (UPPAAL model)
- Cache specs.
  - generate (cache-gen)
    - Caches (UPPAAL model)
Prototype Implementation (cont.)

- **Annotated Executable**
  - disassemble (objdump, Dissy)
  - ARM assembly
  - value analysis (WALi)

- **Pipeline (UPPAAL model)**
  - ARM-to-UPPAAL
  - Control Flow Graph (UPPAAL model)

- **Main Memory (UPPAAL model)**

- **Cache specs.**
  - generate (cache-gen)
  - Caches (UPPAAL model)

- **Combine**
  - Complete model (UPPAAL model)

- **Model check (UPPAAL)**
  - WCET
Prototype Implementation (cont.)

- Annotated Executable
  - disassemble (objdump, Dissy)
    - ARM assembly
      - value analysis (WALi)
  - Pipeline (UPPAAL model)
  - Main Memory (UPPAAL model)
  - Cache specs.
    - generate (cache-gen)
      - Caches (UPPAAL model)
      - Combine
        - Complete model (UPPAAL model)
      - model check (UPPAAL)
        - WCET

- Prototype Implementation (cont.)
Prototype Implementation (cont.)

1. Annotated Executable
   - disassemble (objdump, Dissy)
     - ARM assembly
       - value analysis (WALi)
   - ARM-to-UPPAAL
     - Control Flow Graph (UPPAAL model)
   - Pipeline (UPPAAL model)
   - Main Memory (UPPAAL model)
2. Cache specs.
   - generate (cache-gen)
     - Caches (UPPAAL model)
   - Combine
     - Complete model (UPPAAL model)
3. Model check (UPPAAL)
   - WCET
Prototype Implementation (cont.)

- Annotated Executable
  - disassemble (objdump, Dissy)
  - ARM assembly
  - value analysis (WALi)

- Pipeline (UPPAAL model)
  - ARM-to-UPPAAL
  - Control Flow Graph (UPPAAL model)

- Main Memory (UPPAAL model)

- Cache specs.
  - generate (cache-gen)
  - Caches (UPPAAL model)

- Complete model (UPPAAL model)

- model check (UPPAAL)
  - WCET
Prototype Implementation (cont.)

1. **Annotated Executable**
   - **disassemble** (objdump, Dissy)
   - **ARM assembly**
   - **value analysis** (WALi)

2. **Pipeline** (UPPAAL model)
   - **ARM-to-UPPAAL**
   - **Control Flow Graph** (UPPAAL model)

3. **Main Memory** (UPPAAL model)
   - **combine**
   - **Complete model** (UPPAAL model)

4. **Cache specs.**
   - **generate** (cache-gen)
   - **Caches** (UPPAAL model)
   - **model check** (UPPAAL)

5. **WCET**
Prototype Implementation (cont.)

Annotated Executable

Pipeline (UPPAAL model)

Main Memory (UPPAAL model)

Cache specs.

disassemble (objdump, Dissy)

ARM-to-UPPAAL

ARM assembly

Control Flow Graph (UPPAAL model)

combine

Complete model (UPPAAL model)

model check (UPPAAL)

WCET

value analysis (WALi)

generate (cache-gen)

Caches (UPPAAL model)
The Combined Model

function: main

function: fib

Pipeline

Fetch stage

Decode stage

Execute stage

Memory stage

Writeback stage

Caches

Instruction Cache

Data Cache

RAM

Main Memory

We ask UPPAAL for the WCET:
sup: cyclecounter
We ask UPPAAL for the WCET:

\texttt{sup: cyclecounter}
Experiments on the Prototype Implementation

- Examine three qualities:
  - Size and complexity of processes
  - How much sharper WCETs get when caching is taken into account
  - Resource usage
- No toggling of the pipeline
- No reference WCETs available
- Benchmark programs from Mälardalen Real-Time Research Center
  - Used to benchmark WCET analysis tools
  - Wide selection of computation tasks
Experiments on the Prototype Implementation (cont.)

- Taking the instruction cache into account:
  - WCETs that are up to 97% sharper
  - 78% on average at -02

- Data cache:
  - Up to 68% sharper
  - 31% on average at -02

- Prototype is able to analyse 14 of the 25 non-floating point benchmarks

- Some benchmarks fail due to
  - State space explosion
  - Write to program counter
  - Recursion
  - Value analysis problems
Experiments on the Prototype Implementation (cont.)

- Taking the instruction cache into account:
  - WCETs that are up to 97% sharper
  - 78% on average at -02

- Data cache:
  - Up to 68% sharper
  - 31% on average at -02

- Prototype is able to analyse 14 of the 25 non-floating point benchmarks

- Some benchmarks fail due to
  - State space explosion
  - Write to program counter
  - Recursion
  - Value analysis problems

The details: [http://metamoc.martintoft.dk](http://metamoc.martintoft.dk)
Graphical User Interface for Experimentation

The WCET Analysis using Model Checking and Static Analysis.
Thank you for your attention

Questions?

These slides are available at http://martintoft.dk/slides/mt-lab-advisory-board-meeting-2009.pdf