State-of-the-art of WCET (Worst-Case Execution Time) Estimation methods

Isabelle PUAUT

University de Rennes I / IRISA Rennes (CAPS)
September 2007
Outline

- Context: real-time systems
- Timing analysis methods
- Classes of WCET estimation techniques
  - Dynamic (measurement-based) methods
  - Static methods
- Static WCET estimation methods
  - Flow analysis
  - Computation
  - Hardware-level analysis
- Open issues and current research directions
Real-time systems

- **Definition**
  - Systems whose correct behavior depends not only on the value of the computation but also on the **time** when produced

- **Timing constraints**
  - Quantified limit between the occurrence of events (minimum and/or maximum)
  - Example: deadline

- **Source of timing constraints**
  - Control applications: stability of physical process
  - Delay until system failure
Classes of real-time systems

- **Hard real-time**
  - Missing a deadline can cause catastrophic consequences in the systems: need for a priori guarantees in the worst-case scenario
  - Ex: control in transportation systems, nuclear applications, etc.

- **Soft real-time**
  - Missing a deadline decreases the quality of service of the system
  - Ex: multimedia applications (VoD, etc.)
Temporal validation of real-time systems

- Testing
  - Input data + execution (target architecture, simulator)
  - Necessary but not sufficient (coverage of scenarios)

- Schedulability analysis (models)
  - Hard real-time: need for guarantees in all execution scenarios, including the worst-case scenario
  - Task models
  - Schedulability analysis methods (70s ⇒ today)
Schedulability analysis
Introduction (1/2)

- Definition
  - Algorithms or mathematical formulas allowing to prove that deadlines will be met

- Classification
  - Off-line validation
    - Target = hard real-time systems
  - On-line validation
    - Acceptance tests executed at the arrival of new tasks
    - Some tasks may be rejected → target = soft real-time systems
Input: system model
- Task model
  - Arrival: periodic, sporadic, aperiodic
  - Inter-task synchronization: precedence constraints, resource sharing
  - Worst-case execution time (WCET)
- Architecture
  - Known off-line for hard real-time systems

Output
- Schedulability verdict
Schedulability analysis
Example (1/2)

- System model
  - Periodic tasks \( (P_i) \), deadline \( D_i \leq P_i \)
  - Fixed priorities (the smaller \( D_i \) the higher priority)
  - Worst-case execution time: \( C_i \)

- Necessary condition

\[
U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1
\]

- Sufficient condition

\[
\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^n - 1)
\]

- Low complexity
Schedulability analysis
Example (2/2)

- Same task model as before
- Estimation of response time: limit of series
  \[ w_i^0 = C_i \]
  \[ w_i^{k+1} = C_i + \sum_{j \in \text{np}(i)} \left\lfloor \frac{w_i^k}{P_j} \right\rfloor C_j \]

- The series limit is the task response time (when converges)
- The system is schedulable when \( R_i \leq D_i \)
**WCET**

**Definition**

- **Upper bound** for executing an isolated piece of code
  - Code considered in *isolation*
  - WCET ≠ response time
- WCET = variable Ci in schedulability tests
WCET

Different uses

- Temporal validation
  - Schedulability tests
- System dimensioning
  - Hardware selection
- Optimization of application code
  - Early in application design lifecycle
WCET

Challenges in WCET estimation

- **Safety** (WCET > any possible execution time):
  - confidence in schedulability analysis methods

- **Tightness**
  - Overestimation $\Rightarrow$ schedulability test may fail, or too much resources might be used
WCET

Influencing elements

- Sequencing of actions (execution paths)
  - Input data dependent
- Duration of every action on a given processor
  - Hardware dependent

```c
void f(int a) {
    for (int i=0; i<10; i++) {
        if (a==1) X; else Y;
    }
}
```
WCET estimation methods

Dynamic methods

- **Principle**
  - Input data
  - Execution (hardware, simulator)
  - Timing measurement

- **Generation of input data**
  - User-defined: reserved to experts
  - Exhaustive
    - Risk of combinatory explosion
  - Automatic generation: genetic algorithms, etc.
  - Safety?
WCET estimation methods

Static analysis methods

- **Principle**
  - Analysis of program structure *(no execution)*
  - Computation of WCET from program structure

- **Components**
  - Flow analysis
    - Determines possible flows in program
  - Low-level (hardware-level) analysis
    - Determines the execution time of a sequence of instructions (basic block) on a given hardware
  - Computation
    - Computation from results of other components
    - All paths need to be considered *(safety)*
Static WCET estimation methods

Overview of components

- Source code
- Compiler
- Object code
- Flow analysis
- (Annotations)
- Flow representation
- Low-level analysis
- Computation
- WCET
Static WCET estimation methods

Flow analysis (1/4)

Structurally feasible paths
(infinite)

Basic finiteness
(bounded loops)

Actually feasible
(infeasible paths,
mutually exclusive paths)

WCET estimation methods: terminating programs
Static WCET estimation methods

Flow analysis (2/4)

- Infeasible paths

```c
int baz (int x) {
    if (x<5) // A
        x = x+1; // B
    else x=x*2; // C
    if (x>10) // D
        x = sqrt(x); // E
    return x; // F
}
```

- Path ABDEF is infeasible
- Identification of infeasible paths: improves tightness.
- Methods: abstract interpretation
Static WCET estimation methods

Flow analysis (3/4)

- Maximum number of iterations of loops

```
for i := 1 to N do
  for j := 1 to i do
    begin
      if c1 then A.long
      else B.short
      if c2 then C.short
      else D.long
    end
```

- Loop bound: \( N \)
- Loop bound: \( N \)

\[ \frac{(N+1)N}{2} \] executions

- A tight estimation of loop bounds reduces the pessimism of the WCET estimate.
Static WCET estimation methods

Flow analysis (4/4)

- Determination of flow facts
  - Automatic (static analysis): infeasible in general
  - Manual: annotations
    - Loop bounds: constants, or symbolic annotations
    - Annotations for infeasible / mutually exclusive paths

- Some numbers (P. Puschner)
Static WCET methods: computation

Tree-based computation

- **Data structures**
  - Syntax tree (control structures)
  - Basic blocks

- **Principle**
  - Determination of execution time of basic block (low-level analysis)
  - Computation based on a bottom-up traversal of the syntax tree (timing schema)
### Static WCET methods: computation

#### Tree-based computation

<table>
<thead>
<tr>
<th>WCET(SEQ)</th>
<th>S1;...;Sn</th>
<th>WCET(S1) + ... + WCET(Sn)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCET(IF)</td>
<td>if(test) then else</td>
<td>WCET(test) + max(WCET(then), WCET(else))</td>
</tr>
<tr>
<td>WCET(LOOP)</td>
<td>for(;tst;inc) {body}</td>
<td>maxiter * (WCET(tst)+WCET(body+inc)) +</td>
</tr>
</tbody>
</table>

#### Timing schema

\[
\begin{align*}
\text{WCET(Seq1)} &= \text{WCET}_{BB0} + \text{WCET(Loop)} + \text{WCET}_{BB6} \\
\text{WCET(Loop)} &= 4 * (\text{WCET}_{BB1} + \text{WCET(Seq2)}) + \text{WCET}_{BB1} \\
\text{WCET(Seq2)} &= \text{WCET(If)} + \text{WCET}_{BB5} \\
\text{WCET(If)} &= \text{WCET}_{BB2} + \max(\text{WCET}_{BB3}, \text{WCET}_{BB4})
\end{align*}
\]
Static WCET methods: computation

Tree-based computation

<table>
<thead>
<tr>
<th>WCET(SEQ)</th>
<th>S1;...;Sn</th>
<th>WCET(S1) + ... + WCET(Sn)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCET(IF)</td>
<td>if(test) then else</td>
<td>WCET(test) + max(WCET(then), WCET(else))</td>
</tr>
<tr>
<td>WCET(LOOP)</td>
<td>for(;tst;inc) {body}</td>
<td>maxiter * (WCET(tst)+WCET(body+inc)) +</td>
</tr>
</tbody>
</table>

Timing schema

WCET(Seq1) = WCET_BB0 + WCET(Loop) + WCET_BB_6
WCET(Loop) = 4 * (WCET_BB1 + WCET(Seq2)) + WCET_BB1
WCET(Seq2) = WCET(If) + WCET_BB5
WCET(If) = WCET_BB2 + max(WCET_BB3, WCET_BB4)
Integer linear programming
- Objective function: \( \text{max: } f_1 t_1 + f_2 t_2 + \ldots + f_n t_n \)
- Structural constraints
  \[ \forall v: f_i = \sum_{a_i \in \text{In}(v)} a_i = \sum_{a_i \in \text{Out}(v)} a_i \]
  \[ f_1 = f_7 = 1 \]
- Extra flow information
  \[ f_i \leq k \text{ (loop bound)} \]
  \[ f_i + f_j \leq 1 \text{ (mutually exclusive paths)} \]
Static WCET methods: low-level analysis

Introduction

- Simple architecture
  - Execution time of an instruction only depends on instruction type and operands
  - No overlap between instructions, no memory hierarchy

- Complex architecture
  - Local timing effects
    - Overlap between instructions (pipelines)
  - Global timing effects
    - Caches (data, instructions), branch predictors
    - Requires a knowledge of the entire code
Static WCET methods: low-level analysis

Pipelining

- **Principle:** parallelism between instructions
  - **Intra basic-block**
  - **Inter basic-block**

<table>
<thead>
<tr>
<th>Fetch</th>
<th>Decode</th>
<th>Execute</th>
<th>Memory</th>
<th>Write Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>ME</td>
<td>WB</td>
</tr>
</tbody>
</table>

- Time:

```plaintext
IF   ID   EX   ME   WB
```
Static WCET methods: low-level analysis

Pipelining (simple-scalar)

- Intra basic block
  - Reservation tables describing the usage of pipeline stages
  - Obtained by WCET analysis tool or external tool (simulator, processor)

- Inter basic-block: modification of computation step
Static WCET methods: low-level analysis

Instruction caches

- **Cache**
  - Takes benefit of temporal and spatial locality of references
  - **Speculative**: future behaviour depends on past behaviour
  - Good average-case performance, but predictability issues

- **How to obtain safe and tight estimates?**
  - Simple solution (all miss): overly pessimistic
  - Objective: predict if an instruction will (**certainly**) cause a hit or might (**conservatively**) cause miss.
Static WCET methods: low-level analysis

Instruction caches

- Static cache simulation [Mueller]
- Computation of Abstract Cache States (ACS)
  - Contains all possible cache contents considering all possible execution paths
  - Computed using data-flow analysis (fixed-point)
- Instruction categorization from ACS
  - *always hit*
  - *always miss*
  - *first miss, first hit* (instruction in loop)
input_state(top) = all invalid lines

while any change do
    for each basic block instance B do
        input_state(B) = null
        for each immediate predecessor P of B do
            input_state(B) += output_state(P)
        end for
        output_state(B) = (input_state(B) + prog_lines(B)) - conf_lines(B)
    end for
end while
Static WCET methods: low-level analysis

Other hardware elements

- Data caches
  - Extra issue: determination of addresses of data

- Branch predictors
  - Local predictors [Colin]
  - Global predictors [Mitra], [Rochange]
Static WCET estimation methods

Some quantitative results

- Impact of hardware modelling (analysis of RTEMS, Heptane, Irisa)
Static WCET estimation methods
A method for every usage

- Static WCET estimation
  - Safety 😊
  - Pessimism 😞
  - Need for a hardware model 😞
  - Trade-off between estimation time and tightness (tree-based / IPET) 😊

- Measurement-based methods
  - Safety ? Probabilistic methods
  - Pessimism 😊
  - No need for hardware models 😊 But need to know the hardware 😞
Open issues (1/4)

- Low-level analysis: increase of hardware complexity
  - Complexity of static WCET estimation tools
  - Timing anomalies, integration of sub-analyses
  - Analysis tools may be released a long time after the hardware is available
### Open issues (2/4)

#### Timing anomalies

<table>
<thead>
<tr>
<th>Disp.</th>
<th>Cycle</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>LD r4, 0(r3)</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>ADD r5, r4, r4</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>ADD r11, r10, r10</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>MUL r11, r11, r11</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
<td>MUL r13, r12, r12</td>
</tr>
</tbody>
</table>

Diagram showing timing anomalies for hit and miss scenarios.
Open issues (3/4)

- Low-level analysis: research directions
  - Design of predictable hardware [Vienna]
  - Models of complex hardware: speculative loading of instructions [IRIT], branch prediction [Séoul, York, IRIT], multi-cores, SMT
  - Software-controlled caches (cache locking) and scratchpad memories: [IRISA, Mälardalen]
  - Hardware effects as probabilities: [York]
  - Identification of timing anomalies
Open issues (4/4)

- Flow analysis and computation
  - Automation of flow analysis [Mälardalen]
  - Paradigms for real-time programming: single path paradigm [Vienna]
  - Measurement-based methods: automatic test generation for full path coverage [CEA]
- Worst-case oriented compilation [IRISA]
WCET estimation tools

- **Academic**
  - Cinderella [Princeton]
  - Heptane [IRISA]
  - Otawa [IRIT Toulouse]
  - SWEET [Sweden]

- **Industrial**
  - Bound-T [SSF]
  - AIT [AbsInt, Sarbrüken]
  - Rapitime [Rapita systems, York]
Some pointers

- **Bibliography**
  - Survey paper in TECS, 2007
  - Journal of real-time systems, special issue on static worst-case execution-time analysis, 2 issues in 1999 et 2000
  - Workshop on static worst-case execution time analysis (2001..2007), in conjunction with ECRTS

- **Projects**
  - Artist2 working group on timing estimation
Any question?

Submit to RTNS’08

- Rennes, october 2008, 16-17th
- Deadline for submission
  - Apr, 26th 2008
- Junior researcher workshop
- More information
  http://rtns08.inria.fr