Measuring Execution Times

Peter Puschner
Benedikt Huber

slides credits: P. Puschner, R. Kirner, B. Huber
VU 2.0 182.101 SS 2015
Why (not to) measure execution-times ...  
Instrumentation  
Measurement-based approaches  
  • Industrial practice  
  • Evolutionary algorithms  
  • Probabilistic WCET analysis  
  • Measurement-based timing analysis
Measuring -- a Simple Solution!? 

Why not obtain a WCET estimate by measuring the execution time?

Start Timing Measurement → Execute Program on Target HW → Stop Timing Measurement → Timer, Logic Analyzer, etc. → WCET estimate?
Why not just Measure WCET?

• Measuring all different traces is intractable (e.g., $10^{40}$ different paths in a mid-size task)

• Selected test data for measurement may fail to trigger the longest execution trace

• Test data generation: rare execution scenarios may be missed (e.g., exception handling, …)

• Partitioning: combining WCET of parts does not necessarily yield the global WCET (anomalies)
Why not just Measure WCET? (2)

- Problem of setting the processor state to the worst-case start state.

Conclusions:
- Measurements in general underestimate the worst-case execution time.
- More systematic WCET analysis techniques are required to obtain a trustworthy WCET bound!
On the other hand ...

- Not all applications require a safe WCET bound
  - Soft real-time systems (e.g., multimedia)
  - Fail-safe hard real-time systems (e.g., windmill)
- Easy to adapt to new platform (limited platform support by tools for static analysis)
- Low annotation effort \(\Rightarrow\) get a quick rough estimate of the execution time
- Complement to static analysis, produces “hard evidence” about correct timing
- Feedback for improving static WCET analysis
Measurement-Based WCET Analysis

• Key Idea: Timing information is acquired by measuring the execution time of the code executed dynamically on the (physical) target hardware.

• Instrumentation Points (IP): observable events required to trigger timing measurements

• Trace: timing and path information (path = sequence of basic blocks) are gathered in combination
Execution Time Measurements

Goal: Obtain execution time for a path

Instrumentation Interface

System Under Test

FST: set 2 .dcall df ldab _S1 ldab OFST-1,s bitb #15 bne L22 ldab #1 stab L5r
L22: leas 2,s rti quit:

P₁

P₂

Code

Hardware

Hardware Interfaces
- Simple I/O ports
- Address lines
- Debug interfaces
- Communication devices

Execution Time Measurement System
Instrumentation Methods

- Pure hardware instrumentation
- External execution time measurements using software triggering
- Pure internal (software) instrumentation
Instrumentation Decisions

Persistent vs. non-persistent instrumentation
Code instrumentation vs. hardware instrumentation

Possible design decisions:
- Counter location? Interface data?
- Control flow manipulation? Input data generation?
- Number of measurement runs?
- Resource consumption?
- Required devices?
- Installation effort?
Measurement Considerations

How to measure what we want to measure:

• Instrumentation (IPs) must not alter program flow or execution time in an unknown or unpredictable way. IPs have to be persistent if changing either.

• How can we make sure that executions always start from the same (known) state (cache, pipeline, branch prediction, ...)?
Measurement-Based Methods

- Industrial practice
- Evolutionary algorithms
- Probabilistic WCET analysis (pWCET)
- Measurement-based timing analysis (mTime)
Industrial Approach

Example of Industrial Design Flow

Design
Matlab + Simulink
Matlab + RTW

Test
Prototype Boards
Custom Hardware

Deploy
Custom automotive Hardware
Industrial Approach
Industrial Approach – Input Vectors
Industrial Approach – Random Data
Industrial Approach – Pitfalls

• Test-coverage metrics for functional tests are not sufficient for a measurement-based WCET assessment

• Random data may miss the path with the longest Execution Time

• The state of the system is typically not taken into consideration
Evolutionary Algorithms (EA)

Gene = an independent property of an individual
Individual = vector of genes
Population = set of a number individuals
Fitness value = chance of survival of an individual
Recombination = mating of two individuals, exchange of genes
Mutation = random change of a gene

[Wegener et al., 96]
Process of Evolutionary Computation

- **Selection**: Survival of the fittest: Stochastical selection and modification of fittest individuals to form a new generation
- **Recombination**: exchange of genes between individuals e.g., 1-point crossover, n-point crossover
- **Mutation**: probabilistic changing of genes
WCET by Evolutionary Algorithms

- Gene = input or state variable
- Fitness value = measured execution time (longer execution time ⇒ higher fitness)
- Result = “fittest individual” = individual with the longest execution time
- Gives good estimations of the execution time but no safe upper bound of the WCET
WCET by Evolutionary Algorithms

- Start:
  [0] x=0, y=0 → ET: 40
  [1] x=1, y=1 → ET: 40

- Crossover:
  [2] x=0, y=1 → ET: 50
  [3] x=1, y=0 → ET: 30

Algorithm terminates if fitness does not improve for a given number of iterations

```java
if(x) {
    fast();
} else {
    slow();
}
if(y){
    slow();
} else {
    fast();
}
```
## Results of Applying an EA

<table>
<thead>
<tr>
<th>Program</th>
<th>WCET</th>
<th>WCET SA</th>
<th>WCET EA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>13,190,619</td>
<td>15,357,471</td>
<td>13,007,019</td>
</tr>
<tr>
<td>Sort</td>
<td>11,872,718</td>
<td>24,469,014</td>
<td>11,826,117</td>
</tr>
<tr>
<td>Graphics</td>
<td>N/A</td>
<td>2,602</td>
<td>2,176</td>
</tr>
<tr>
<td>Railroad</td>
<td>N/A</td>
<td>23,466</td>
<td>22,626</td>
</tr>
<tr>
<td>Defense</td>
<td>N/A</td>
<td>72,350</td>
<td>35,226</td>
</tr>
</tbody>
</table>

SA .... Static analysis  
EA .... Evolutionary algorithms

[Mueller, Wegener, RTSS1998]
Malignant Scenarios

Functions with many local XT maxima (e.g., sorting)

Example: \( xt_i = f(x, y) \quad x, y \in [0..129] \)

\( wcet(f_3) = f_3(127, 129) \)
Probabilistic WCET Analysis

• Goal: determine the probability distribution of the WCET

• Solution: syntax tree representation of the program and a probabilistic timing schema
  — Tree leafs are basic blocks
  — Inner nodes: sequential composition, conditional composition, iterative composition
  — Timing measurements between IPs
Probabilistic WCET Analysis

- **Timing Schema**
  - $W(A) = \text{WCET of } A$
  - $W(A;B) = W(A)+W(B)$
  - $W(\text{if } E \text{ then } A \text{ else } B) = W(E) + \max(W(A), W(B))$
  ...

- **Probabilistic Schema**
  - $X, Y$ random variables for execution times $A, B$
  - Distribution function $F(x) = P[X \leq x], G(y) = P[Y \leq y]$
  - Sequence: $A;B \Rightarrow Z = X + Y \Rightarrow H(z) = P[X + Y \leq z]$
    
    In case of independence: standard convolution
    $$H(z) = \int F(x)G(z-x)dx$$
Probabilistic WCET Tool (pWCET)

- RapiTime
  - Convenient reporting, “hot spot analysis”
  - Probabilistic model using extreme-value statistics
  - Cumulative probability that a randomly chosen sample from the end-to-end execution times during testing exceeds a given time budget

⇒ Quality depends on chosen test data
Path-Based Timing Analysis (mTime) with Program Segmentation

Analyzer tool  
Execution time measurement framework  
Calculation tool

C-Source

Analysis phase

Measurement phase

Calculation phase

WCET bound
Path-Based Timing Analysis (mTime) with Program Segmentation

\textit{mTime} performs the following five steps in the analysis:

1. Static program analysis at C source code level
2. Automatic control-flow graph partitioning
3. Test data generation (using model checking)
4. Execution time measurements
5. WCET bound calculation step
Control-Flow Graph Partitioning

Step 2

- Decomposing CFG into smaller units
- Program segmentation (PSG)
  - Set of program segments (PS)
  - $PS_i = (s_i, t_i, \Pi_i)$ with start node $s_i$, termination node $t_i$ and set of paths $\Pi_i$
- „Good“ program segmentation balances
  - number of PSs and
  - average number of paths per PS
- Maximum number of paths within PSs can be configured by the path bound parameter
Control-Flow Graph Partitioning

- Results of applying the partitioning algorithm for case study application

<table>
<thead>
<tr>
<th>Path bound</th>
<th>PS</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>5</td>
<td>1455</td>
</tr>
<tr>
<td>100</td>
<td>7</td>
<td>336</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>242</td>
</tr>
<tr>
<td>20</td>
<td>11</td>
<td>130</td>
</tr>
<tr>
<td>15</td>
<td>13</td>
<td>106</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>92</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
<td>83</td>
</tr>
<tr>
<td>4</td>
<td>38</td>
<td>84</td>
</tr>
<tr>
<td>2</td>
<td>88</td>
<td>117</td>
</tr>
<tr>
<td>1</td>
<td>171</td>
<td>171</td>
</tr>
</tbody>
</table>
Control-Flow Graph Partitioning

Step 2

Example of generated code:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Paths</td>
<td>2.19e+32</td>
</tr>
<tr>
<td>Path bound</td>
<td>5,000</td>
</tr>
<tr>
<td>Identified PS</td>
<td>25</td>
</tr>
<tr>
<td>#Measurements</td>
<td>30.000</td>
</tr>
</tbody>
</table>
Test Data Generation

• 4-stage process
  – Level 4: Model Checking
  – Level 3: Heuristics
  – Level 2: Random Search
  – Level 1: Cache

• Full path coverage (determinism)
• Key challenge: identification of infeasible paths
Model Checking (1)

Model checkers are used for automatic formal verification of concurrent finite automata.

Model
C source code

Assertions
Path ‘x’ will not be executed

Model Checker

Model is safe
Path ‘x’ is infeasible

Model is unsafe
(counterexample)
Counterexample holds data to reach path ‘x’
Model Checking (2)

- Enforcement of execution paths
- Each path $\pi$ requires an individual model $M$
- Model checking answers for $M$:
  - $\pi$ is infeasible, or
  - a counter example including the input data to enforce the execution of $\pi$
Execution Time Measurements

Internal or external devices for execution-time measurements

Step 4
WCET Bound Calculation Step

- Program segment execution times are combined by integer linear programming (ILP) or longest path search.
  - Advantage: only feasible paths within each PS contribute.
  - Deficiency: lack of global path information ⇒ refinement possible.
Experiments

Model-checker performance: SAL vs. CBMC

<table>
<thead>
<tr>
<th></th>
<th>#Paths MC</th>
<th>Time Analysis [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CBMC</td>
</tr>
<tr>
<td>TestNicePartitioning</td>
<td>63</td>
<td>11.2</td>
</tr>
<tr>
<td>ActuatorMotorControl</td>
<td>280</td>
<td>1202.2</td>
</tr>
<tr>
<td>ADCConv</td>
<td>136</td>
<td>65.2</td>
</tr>
<tr>
<td>ActuatorSysCtrl</td>
<td>96</td>
<td>32.7</td>
</tr>
</tbody>
</table>

¹ Model size is too big, memory error of the model checker (core dump)
| Path Bound | #Paths (\(\Sigma |\pi|\)) | #Program Segments | #Paths Random | #Paths MC | Coverage (#Paths) | WCET Bound | Time (Analysis) [s] | Time (ETM) [s] | Overall Time [s] | Time Analysis / Path MC [s] | Time ETM / Covered Path [s] | #Paths / Program Segment |
|------------|----------------|------------------|---------------|-----------|------------------|------------|------------------|---------------|----------------|-----------------|-----------------|-----------------|
| ActuatorMotorControl | 1 | 171 | 171 | 165 | 6 | 165 | N.A. | 468 | 1289 | 1757 | 78.00 | 7.8 | 1.0 |
| | 10 | 92 | 14 | 63 | 29 | 68 | 3445 | 841 | 116 | 957 | 29.00 | 1.7 | 6.6 |
| | 100 | 336 | 7 | 57 | 279 | 89 | 3323 | 7732 | 62 | 7794 | 27.71 | 0.7 | 48.0 |
| | 1000 | 1455 | 5 | 82 | 1373 | 130 | 3298 | 41353 | 49 | 41402 | 30.12 | 0.4 | 291.0 |
| ActuatorSysCtrl | 1 | 30 | 30 | 6 | 24 | 30 | 151 | 34 | 175 | 209 | 1.42 | 5.8 | 1.0 |
| | 10 | 36 | 14 | 36 | 0 | 36 | 173 | 10 | 85 | 95 | N.A. | 2.4 | 2.6 |
| | 100 | 97 | 1 | 18 | 79 | 25 | 131 | 191 | 10 | 201 | 2.42 | 0.4 | 97.0 |
| ADCConv | 1 | 31 | 31 | 31 | 0 | 31 | 872 | 24 | 192 | 216 | N.A. | 6.2 | 1.0 |
| | 10 | 17 | 3 | 8 | 9 | 9 | 870 | 31 | 22 | 53 | 3.44 | 2.4 | 5.7 |
| | 100 | 74 | 2 | 8 | 66 | 14 | 872 | 220 | 17 | 237 | 3.33 | 1.2 | 37.0 |
| | 1000 | 144 | 1 | 12 | 132 | 12 | 872 | 483 | 11 | 494 | 3.66 | 0.9 | 144.0 |
| TestNicePartitioning | 1 | 14 | 6 | 4 | 10 | 14 | 151 | 15 | 39 | 54 | 1.50 | 2.8 | 2.3 |
| | 10 | 14 | 3 | 3 | 11 | 14 | 151 | 16 | 21 | 37 | 1.45 | 1.5 | 4.7 |
| | 20 | 18 | 2 | 2 | 16 | 15 | 150 | 22 | 16 | 38 | 1.38 | 1.1 | 9.0 |
| | 100 | 72 | 1 | 1 | 71 | 26 | 129 | 106 | 12 | 118 | 1.49 | 0.5 | 72.0 |
Experiments and Results

Case study: nice_partitioning

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>30</td>
<td>6</td>
<td>24</td>
<td>30</td>
<td>151</td>
<td>34</td>
<td>175</td>
<td>209</td>
<td>1,42</td>
<td>5,8</td>
<td>1,0</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>14</td>
<td>151</td>
<td>15</td>
<td>39</td>
<td>54</td>
<td>1,50</td>
<td>2,8</td>
<td>2,3</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
<td>3</td>
<td>3</td>
<td>11</td>
<td>14</td>
<td>151</td>
<td>16</td>
<td>21</td>
<td>37</td>
<td>1,45</td>
<td>1,5</td>
<td>4,7</td>
</tr>
<tr>
<td>20</td>
<td>18</td>
<td>2</td>
<td>2</td>
<td>16</td>
<td>15</td>
<td>150</td>
<td>22</td>
<td>16</td>
<td>38</td>
<td>1,38</td>
<td>1,1</td>
<td>9,0</td>
</tr>
<tr>
<td>100</td>
<td>72</td>
<td>1</td>
<td>1</td>
<td>71</td>
<td>26</td>
<td>129</td>
<td>106</td>
<td>12</td>
<td>118</td>
<td>1,49</td>
<td>0,5</td>
<td>72,0</td>
</tr>
</tbody>
</table>
Path-based Measurements

• Customizable fast CFG partitioning (Segmentation)
  – Trade-off: Quality vs. #paths

• Automated test data generation using mix of strategies
  – Full path coverage per segment
  – Scalability

• Over- or under-estimation of WCET?
  – pessimistic as well as optimistic effects
  – measurements ⇒ under-estimation
  – combination of segment results ⇒ over-estimation
Summary

• WCET measurements play an important role

• Measuring WCET is far from trivial
  − Identifying the right inputs
  − Dealing with state
  − Instrumentation

• Pure measurements

• Hybrid approaches combine measurements and elements from static analysis (pWCET, mTime)

• Research: HW randomization to obtain independence
Reading Material


Reading Material (2)