Benchmark Generation for Timing Analysis

Peter Wägemann, Tobias Distler, Christian Eichler, Wolfgang Schröder-Preikschat

System Software Group @

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

23rd Real-Time and Embedded Technology and Applications Symposium

RTAS 2017

Funded by

DFG

April 20, 2017
Numerous tools available
- WCET tools used to guarantee timeliness
- AbsInt’s aiT for Airbus A380

⚠️ Problem: Accuracy & validity of analyzers unknown
Problem: What is the WCET?

```c
int f(int x, int n) {
    if (0 == x) { g(); }
    else { h(); }
    for(int i = n; i > 0; i--) {
        for(int j = 0; j < i; j++) {
            k();
        }
    }
    if (0 != x) {
        m();
    }
}
```

- Input-dependent computation
- Nested loop
- Mutual-exclusive paths
Problem: Determining Flow Facts

What are the flow facts?

1. What are the loop bounds?
2. What are the feasible paths?
3. What are the worst-case input values?

Having an existing benchmark

- Impossible to extract all flow facts automatically \(\Leftrightarrow\) Rice’s theorem
- Explicit path enumeration not feasible
- Manually determining flow facts labor-intensive & error-prone

Overestimation of WCET

- Nested loop: \(n \times (n - 1)/2 \mapsto n^2\)
- Include infeasible paths
- Assume cache misses
  ...

...
Problem Statement: Necessity of Baselines

- Necessary to know facts on **global scale**
- Common **baseline** required: **actual WCET**
- **Monolithic structure** of benchmarks prevents detailed evaluation

**Knowledge of flow facts** required for benchmarking
Problem Statement: What is the WCET?

What is the WCET? How to know all flow facts?

GENE: generate benchmarks whose flow facts are known
Outline

1. Motivation

2. GenE Approach

3. Evaluation

4. Conclusion
GenE in a Nutshell
Worst-case path known by construction
Overview

- Pattern suite
- Path budget
- Worst-case input value

Pattern library → Patterns → Benchmark generator

Patterns → Benchmark

Flow facts → Actual WCET

Input → Program
Patterns I

Patterns obtained from
1. Existing benchmarks
2. Industry applications
3. Literature

Patterns are
- Realistic
- Challenging
- Low-level representations (LLVM)
- Woven into each other
Patterns II

- Pattern **know flow facts**
- Patterns are **parameterizable**

```
constant loop pattern

for(i=0; i<2; ++i)
insertion point

executed 2 times
```

```
mutual exclusive path pattern

a < 5 ?

true

false

insertion point

false

true

insertion point

end
```

```
Pattern
know flow facts
Patterns are parameterizable
```
Pattern Suites

Reveal Strength & Weaknesses of Analyzers

- Avoid **monolithic structure** of existing benchmarks
- Stages of WCET analysis
  1. High-level **path analysis**
  2. Low-level **hardware analysis**
- Generate benchmarks with **specific properties**
  1. Avoid overestimations by loops & paths (suite: singlepath)
  2. Generate arithmetic operations, assignments, branches (suite: value_analysis)
- Reveal **individual strength & weaknesses** of analyzers

![Diagram](image)

**Benchmarks tailored to evaluation purpose**
Overview

Benchmark Generator

- Pattern suite
- Pattern library
- Path budget
- Worst-case input value

Benchmark generator

- Benchmarks
- Flow facts

Actual WCET

Program Input
Path Budget

Problem: Varying Instruction Times

Number of instructions along the worst-case path (e.g., 10,000)

Problem: Varying Instruction Times

1. Caches
2. Pipelining
   ...

Requirement: Model hardware-specific behavior

Goals
1. Avoid modeling low-level details during generation process
2. Worst-case input value triggers worst-case path
Path Budget

**Requirement for GenE:**
worst case(non-WC-paths) ≤ best case(WC-path)

Timing of instruction:
\[ t_{\text{instr}} = t_{\text{icache}} + t_{\text{dcache}} + t_{\text{pipeline}} + t_{\text{calc}} \]

**Solution: Maximum timings**
- Find **maximum of worst-case timings** between all instructions
- Example (ARM Cortex-M4)
  
  \[
  \begin{align*}
  \text{add} & \quad r4, r5 && ldr & \quad r0, [r0, 42] \\
  t_{\text{add, min}} &= 1 && t_{\text{ldr, max}} &= 7
  \end{align*}
  \]

  - 7 \times \text{add} compensates 1 \times \text{ldr} in the worst case
  - Overweighting factor: \( F = 7 \)

  \[ \text{\textcolor{green}{Choose sufficiently large: } } F = 25 \]
Example: Benchmark Generation

worst-case input: \[ \cdots 0 \, 1 \]

overweighting factor: \( F = 25 \)

\[
\text{if}(\text{input}[0]) \} \quad \text{budget} -= 10
\]

True

False

budget: 1000

\[
\begin{align*}
  a &= 5 \\
  n &= a \times (\neg \text{input}[1])
\end{align*}
\]

budget: 960

\[
\text{for}(i = 0; i < n; ++i) \\
\ldots
\]

Pattern descriptors: tracking value constraints & side effects

see paper
Overview

WCET Determination

Pattern suite

Path budget

Worst-case input value

Gene

Pattern library

Patterns

Benchmark generator

Input

Program

Actual WCET

Benchmark

Flow facts

GenE

GenE Approach
Determination of Actual WCET

Concrete Execution

- **Two phases** of cost modeling
- Determine actual WCET by **concrete execution**
  1. Cycle-accurate simulator
  2. **Target platform**

*Most accurate timing model is the CPU itself*
Implementation – GenE & Aladdin

- **ALADDIN** framework around **GenE**
- **GenE**\(^1\) implemented as **LLVM** tool
- **GenErate** **LLVM** assembly code
  - Low-level representation
  - Platform independent

\(^1\)github.cs.fau.de/gene
Outline

1. Motivation

2. GenE Approach

3. Evaluation

4. Conclusion
Evaluation: Setup

- ARM Cortex-M4
  - 3-stage pipeline
  - 120 MHz
  - 4 KB instruction cache
- Available WCET analyzers
  - Platin
  - AbsInt’s aiT
## Evaluation: Strengths and Weaknesses

<table>
<thead>
<tr>
<th></th>
<th>constant loop</th>
<th>input-dependent loop</th>
<th>down-sampling loop</th>
<th>triangular loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>aiT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>PLATIN</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

- Modular structure of Gene’s suites
- Create benchmark with only **one challenging pattern**
  - Arithmetic operations, assignments
  - Challenging loop pattern

- ✓ Gene is able to **reveal strength & weaknesses**
Evaluation: Overestimation of Platin/aiT

- Evaluation setup
  1. Generate 10,000 different benchmarks
  2. Calculate overestimation

- Optimum: 0% overestimation

Platin
- Instruction cache
  - enabled: 301%
  - disabled: 96%

aiT
- Instruction cache
  - enabled: 36%
  - disabled: 23%
Evaluation: Overestimation of Platin

Platin (Cortex-M4, disabled instruction cache)
Evaluation: Overestimation of aiT

aiT (Cortex-M4, disabled instruction cache)
**Evaluation: Detecting Bugs in WCET Analyzers**

- **Observation**: Underestimations of actual WCET
- **AbsInt**: **GENE**’s benchmarks helped to **detect bugs** in aiT
  1. Pipeline model: speculative branch prediction
  2. Memory model: missing cycle for access

- **Assess level of overestimation & **detect underestimations**
1. Motivation

2. GenE Approach

3. Evaluation

4. Conclusion
**Conclusion**

*Motivation:* Missing baselines of existing code

Generate path and then labyrinth around it

Flow facts known in generated program

**GenE**

✓ WCET known
✓ Challenging & tailored benchmarks
✓ Practical relevance:
  detected bugs in WCET analyzer
Questions?

Gene & Aladdin Source Code

https://gitlab.cs.fau.de/gene

Questions?
Thank you for your attention!