GenE: A Benchmark Generator for WCET Analysis

Peter Wägemann, Tobias Distler, Timo Höning, Volkmar Sieh, Wolfgang Schröder-Preikschat

System Software Group

15th International Workshop on Worst-Case Execution Time Analysis (WCET’15)

July 7, 2015
Motivating Example

```c
void f(a, n) {
    if (a)
        g();
    else
        h();
    a = 1;
    for (i = n-1; i >= 1; i--) {
        for (j = 0; j < i; j++) {
            k();
        }
    }
    ++a;
    if (a % 2)
        f2();
}
```

What is the WCET of function \( f() \)?

- Input-dependent computation
- Nested loop
- Infeasible path
## Motivation – Determining Flow Facts

### What are the flow facts?

1. What are the **loop bounds**?
2. What are the **feasible paths**?
3. What are the **concrete input values** triggering the WCET?

### Having an existing benchmark

- Difficult to extract all possible flow facts automatically
- Explicit path enumeration not feasible for real-world applications
- Manually determining flow facts labor-intensive and error-prone

### Over-approximations of actual WCET

- Nested loop: $n \times (n - 1)/2 \mapsto n^2$
- Include infeasible paths

...
**Motivation – Baseline**

- **WCET Tool Challenge**: comparison of WCET analyzers
- Necessary to know flow facts for evaluation on **global scale**
- Create common **baseline** (actual WCET)

**How to know all flow facts?**

**GENE**: generate a benchmark of which the flow facts are known
Agenda

1. Motivation

2. GenE
   2.1 Patterns and Pattern Selection
   2.2 Benchmark Weaving
   2.3 Cost Modeling & WCET Determination

3. Conclusion
**GenE – Application Scenario**

GenE’s hardware analysis receives flow facts

Comparison of **actual WCET** with upper bound of WCET analyzer
- Evaluation: Extent of over-approximations?
- Validation: WCET-analyzer result below actual WCET?

**Incremental process** imaginable
1. Loop bound not found by Analyzer_A
2. GenE: provide annotation
Pattern pools (e.g., paths, loops)

GenE
- Weaving patterns
- Tracking flow facts

Outputs
1. Generated benchmark
2. Flow facts of the benchmark
Patterns

Patterns are inserted into code
New **insertion points** are created
**Flow facts known** for patterns
Patterns store (parametric) costs
Patterns extracted from existing benchmark suites

```c
for(i = n-1; i >= 1; i--){
    for(j = 0; j < i; j++){
        // insertion_point_1
    }
} // insertion_point_2
```

Loop pattern

```c
if( condition ){
    // insertion_point_1
} else {
    // insertion_point_2
} // insertion_point_3
```

Path pattern

```c
void f() {
    static int inited = 0;
    if ( !inited ){
        init();
        inited = 1;
    }
    ...
```
Pattern Selection

```java
switch (state) {
    case GROUND:
        ...
    case ENGINE_START:
        ...
}
```

```java
for (i=1; i < m; i += 2) {
    for (j=i; j <= n; j += step) {
        temp = data[j-1] - data[j];
        ...
    }
}
```

State-machine pattern

Signal-processing pattern

- Assignment of **weights** to patterns
- Weights considered during pattern selection
- Configurable benchmark properties
Benchmark Weaving – Grammar

1. \( S \) \( \rightarrow \) FunctionBegin \( \cdot \) inp \( \cdot \) FunctionEnd
2. inp \( \rightarrow \) \( \varepsilon \)
3. inp \( \rightarrow \) Statement \( \cdot \) inp
4. Statement \( \rightarrow \) Assignment
5. Statement \( \rightarrow \) IfBegin \( \cdot \) inp \( \cdot \) ElseBegin \( \cdot \) inp \( \cdot \) EndIf
6. Statement \( \rightarrow \) LoopHead \( \cdot \) inp \( \cdot \) LoopEnd

(Additional production rules)

- Limited grammar to reduce complexity
- Grammar considered by weaving algorithm
function GenE(vars, cost)
  if (cost == 0)
    return vars

switch (select_production(cost))
  case Statement · inp:
    ...
  case Assignment:
    (new_vars, operation) ← select_assignment(vars)
    emit_Assignment(operation)
  case IfBegin · inp · ElseBegin · inp · EndIf:
    ...
  case LoopHead · inp · LoopEnd:
    ...
return new_vars // return updated variables

- Pattern selection based on distributable cost
- **Top-down** value tracking
Top-down generation and cost distribution

Recursive application on nodes
Benchmark Weaving – Example

input: \[ \cdot 0 1 \]

\begin{align*}
\text{cost: } & 1010 \\
\text{if (input[0])} & \text{cost } - = 10 \\
\text{True} & \text{cost: } 990 \\
\text{False} & \text{cost: } 10 \\
a &= 5 \\
n &= a * (!\text{input}[1]) \\
\text{for (i = 0; i < n; ++i)} & \{ \\
& \ldots \\
\text{cost: } & 950
\end{align*}
Cost Modeling – Problem

WCET analysis
1. High-level analysis: **program structure**
2. Low-level analysis: **target platform**

**Flow facts must respect target platform!**
Avoid generating target-dependent benchmarks
Cost Modeling – Solution

```java
if( condition ){
    f(); // worst-case path  
    cost: 990
} else {
    g();  
    cost: 10
}
```

- **Input-/path-oriented** approach
- **Relative** cost modeling
- **Overweighting** branches by large factors
- Refinement possible through target-specific knowledge
- \( \text{costof}( f() ) \gg \text{costof}( g() ) \)

**Most important flow fact**

- **Input values** for triggering worst-case path
Determine WCET from worst-case inputs: \textbf{concrete execution}

1. Target platform
2. Cycle-accurate simulator

\textbullet\ Measured/simulated execution time yields \textbf{actual WCET}

\begin{itemize}
    \item Program \hspace{1cm} \begin{itemize}
        \item generated.elf
    \end{itemize}
    \item Worst-Case Input Values \hspace{1cm} \begin{itemize}
        \item 10100\ldots
    \end{itemize}
\end{itemize}
Two phases of modeling

1. Overweighting of branches $\sim$ input values
2. Input values for measurement/simulation on hardware $\sim$ actual WCET

Hardware features (i.e., caching, pipelining) not explicitly modeled

**Hardware implicitly modeled** through measurement/simulation
Agenda

1. Motivation

2. GenE

3. Conclusion
void f(a, n) {
    if (a)
        g();
    else
        h();
    a = 1;
    for (i = n-1; i >= 1; i --){
        for (j = 0; j < i; j ++){
            k();
        }
    }
    ++a;
    if (a % 2)
        f2();
}

Problem
- Comprehensive WCET evaluations: baseline necessary
- Know all flow facts?

Solution
- Generate flow facts!
- Recombining patterns
- Weaving benchmark top-down
- Relatively overweighing branches
- Concrete execution on target to determine WCET

Benchmarks with known WCET for WCET-analyzer evaluation ✓
Discussion

- Work in progress
- Flow-fact format?
- WTC’16?

Questions? Discussion!
Thanks for the attention!