Estimation of Worst-Case Execution Time (WCET)

Structure

- What is Worst-Case Execution Time (WCET)?
- What is WCET analysis?
- Why WCET analysis is hard?
- An established WCET analysis framework ...
Execution Time

```c
void signal() {
    read_signal(&curr_signal);
    if (curr_signal < threshold) {
        signal_transformation();
    } else {
        error_handling_routine();
    }
}
```

Finish before each period!

Measuring Execution Time

Program Starts ➔ Program Execution ➔ Logical Analyzer ➔ Program Terminates ➔ Observed ET
Measuring is Unsafe

Execution Time

Max. Observed ET < Actual WCET

Unsafe

The Consequence

- Unsafe estimation $\rightarrow$ incorrect timing prediction

Using Measured WCET

Real System Behavior
Then, What Do We Need?

How?

-- Static Analysis
void main() {
    int b;
    int i = 0, j = 0;
    while (i < 100) {
        if (b) j++;
        else j--;
        i++;
    }
}
Estimating WCET

- Enumerating all possible executions
  - All possible initial states
  - All possible program paths
- Problem: \textit{scalability}

\[
\text{If this loop iterates 100 times, there will be } 2^{100} \text{ different paths.}
\]

Better Solutions?

- \textit{Separate} path and micro-architecture analysis

  - STEP 1: estimate the WCET for each basic block under given hardware features
    - How to manage exponential incoming states?
  - STEP 2: Find an \textit{upper bound} on the “maximal” execution time (no enumeration)
Implicit Path Enumeration

- Main idea of path analysis

**ORIGINAL GOAL**
Finding the actual path with the maximal execution time

**NEW GOAL**
Finding the execution count of each block, implying the longest path

- Some variables
  - $X_i$: the execution count of basic block $B_i$
  - $C_i$: the WCET of basic block $B_i$ (assuming known for now)

![Graph with execution time formula](image)

$$\sum_{i=0}^{7} X_i \times C_i$$
Implicit Path Enumeration

• Now, the path analysis problem becomes
• Finding a valuation of \(< X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7 >\)
• Such that the execution time is maximized

\[
WCET = \max \sum_{i=0}^{7} X_i \times C_i
\]

• \(X_i\) cannot have an arbitrary value \(\Rightarrow\) The constraints

• Structural constraints
  • \(d_{i,j}\): the execution count of the edge from \(B_i\) to \(B_j\)

For each basic block, we have

\[
X_j = \sum_{\text{all } B_i \rightarrow B_j} d_{j,i} = \sum_{\text{all } B_i \rightarrow B_k} d_{j,k}
\]
Implicit Path Enumeration

- Constraints for the start/end nodes
  - $X_0 = 1$
  - $X_7 = 1$

- Bounding loop iterations

Maximize

$$5 X_0 + 1 X_1 + 4 X_3 + 7 X_4 + 5 X_5 + 8 X_6 + 5 X_2 + 7 X_7$$

Subject to

- $X_0 = 1$
- $X_0 - d_{0,1} = 0$
- $X_1 - d_{0,1} - d_{6,1} = 0$
- $X_1 - d_{1,2} - d_{1,3} = 0$
- $X_2 - d_{1,2} = 0$
- $X_2 - d_{2,7} = 0$
- $X_3 - d_{1,3} = 0$
- $X_3 - d_{3,4} - d_{3,5} = 0$
- $X_4 - d_{3,4} = 0$
- $X_4 - d_{4,6} = 0$
- $X_5 - d_{3,5} = 0$
- $X_5 - d_{5,6} = 0$
- $X_6 - d_{4,6} - d_{5,6} = 0$
- $X_6 - d_{6,1} = 0$
- $X_7 = 1$
- $X_6 - 10 d_{0,1} \leq 0$  // loop bound
How to estimate the WCET for each basic block?

Micro-Architecture Analysis

- Goal
  - Given the hardware features, estimate an upper bound for each instruction (then, basic block)

- Why is it hard?
  - Caches: instruction/data, multi-level, shared, replacement
  - Pipelines (not so often in embedded processors)
  - Branch predictor (not so often in embedded processors)
  - Memory controller, main memory
  - Etc.
Cache in a Nutshell

Why caches work?

- Memory reuse (think of a loop)
- The principle of locality
  - Temporal locality: the reuse of specific data within a relatively small time duration
  - Spatial locality: the use of data elements within relatively close storage locations
Cache in a Nutshell

- Set-associative caches

<table>
<thead>
<tr>
<th></th>
<th>way-1</th>
<th>way-2</th>
<th>way-3</th>
<th>way-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>set-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Replacement policy (set nr = adr mod #set)
Cache in a Nutshell

- Cache Replacement
  - E.g. Least-Recently-Used (LRU)

Access “x”
The MISS case

Access “x”
The HIT case

Cache Analysis

- The purpose of cache analysis for WCET analysis is to statically determine whether each memory reference is hit or miss, regarding the worst-case execution.

- In case precise estimations are hard to get, you are allowed to make mistakes in your prediction, as long as they do not underestimate the WCET. (Safety Requirement)

- But, try to make less mistakes. (Precision Requirement)
Cache Analysis

• A program
  • 100 instructions, 50% cache hit in real execution
  • Hit latency = 2; miss latency = 100

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Result</th>
<th>WCET</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Analysis</td>
<td>Assuming all accesses are cache</td>
<td>10,000</td>
</tr>
<tr>
<td></td>
<td>miss for safety</td>
<td></td>
</tr>
<tr>
<td>With Analysis</td>
<td>90% of the cache hits are</td>
<td>5,590</td>
</tr>
<tr>
<td></td>
<td>successfully identified</td>
<td></td>
</tr>
</tbody>
</table>

44.1% reduction in the estimated WCET!

The Fundamental Challenge

Possible incoming states: \( S = I \times P \)

\( I \): all possible initial HW states

\( P \): all possible program paths

How can we ensure all the possibilities are considered?
How to efficiently manage some many states?
Example cache states

4 possible outcomes in accessing a basic block

1. Always hit (AH)
2. Always miss (AM)
3. First miss (FM)
4. Not Classified (NC)

- Access times e.g. AH → 2, AM → 100, FM → (100,2), NC → 100

This can be predicted by Static Analysis (Abstract Interpretation)
There are commercial tools e.g. aiT from Absint
WCET Calculation

- Integration cache analysis results into IPET

Maximize

\[
(2 \times x_{0h} + 10 \times x_{0m}) + (2 \times x_{1h} + 10 \times x_{1m}) + (2 \times x_{2h} + 10 \times x_{2m}) + (2 \times x_{3h} + 10 \times x_{3m}) + (2 \times x_{4h} + 10 \times x_{4m}) + (2 \times x_{5h} + 10 \times x_{5m}) + (2 \times x_{6h} + 10 \times x_{6m}) + (2 \times x_{7h} + 10 \times x_{7m})
\]

// hit latency = 2; miss latency = 10

// cache constraints

\[
X_0 = x_{0h} + x_{0m} \\
0 \leq x_{0m} \leq X_0 \\
x_{0h} = 0
\]

\[
X_1 = x_{1h} + x_{1m} \\
0 \leq x_{1h} \leq X_1 \\
x_{1m} = 0
\]

\[
X_2 = x_{2h} + x_{2m} \\
0 \leq x_{2h} \leq X_2 \\
x_{2m} = 0
\]


WCET Calculation

Maximize

\[
5 \times X_0 + 1 \times X_1 + 4 \times X_3 + 7 \times X_4 + 5 \times X_5 + 8 \times X_6 + 5 \times X_2 + 7 \times X_7
\]

Subject to

\[
X_0 = 1; \\
X_0 - d_{0.1} = 0; \\
X_1 - d_{0.1} - d_{6.1} = 0; \\
X_1 - d_{1.2} - d_{1.3} = 0; \\
X_2 - d_{1.2} = 0; \\
X_2 - d_{2.7} = 0; \\
X_3 - d_{1.3} = 0; \\
X_3 - d_{3.4} - d_{3.5} = 0; \\
X_4 - d_{3.4} = 0; \\
X_4 - d_{4.6} = 0; \\
X_5 - d_{3.5} = 0; \\
X_5 - d_{5.6} = 0; \\
X_6 - d_{4.6} - d_{5.6} = 0; \\
X_6 - d_{6.1} = 0; \\
X_7 = 1; \\
X_6 - 10 \times d_{0.1} = 0; \quad \text{// loop bound}
\]
How to predict the possible cache states?

1. Always hit (AH)
2. Always miss (AM)
3. First miss (FM)
4. Not Classified (NC)

- Access times e.g. AH → 2, AM → 100, FM → (100,2), NC → 100

This can be predicted by Static Analysis (Abstract Interpretation)
There are commercial tools e.g. aiT from Absint

The Fundamental Challenge

Possible incoming states: \( S = I \times P \)
- \( I \): all possible initial HW states
- \( P \): all possible program paths

Whenever the program reaches the node and accesses block “x”
Can we predict, it is always a cache hit?
The naïve way: remember all Cache States (too many!)

Abstracting Cache States

Many concrete cache states $\rightarrow$ one abstract state
The Abstract States

- How to construct abstract states for a specific purpose?
  - E.g., predicting Always Hit → MUST analysis

![Diagram of abstract states]

- Set intersection
- Abstract domain
- Concrete domain
Manipulating the Abstract States

- Operations needed for Fixed-Point Iteration
  - **Update**: \( U(C_{\text{in}}, m) \rightarrow C_{\text{out}} \)
  - **Join**: \( J(C_1, C_2) \rightarrow C_3 \)

- MUST Update (the miss case)

```
{a} {c} {d}  x  \{x\} {a} {c}  
```

```
a b c d  x  x a b c  
```

```
a c f d  x  x a c f  
```

```
a e c g  x  x a e c  
```
Manipulating the Abstract States

• MUST Update (the hit case)

\[
\begin{align*}
\{a\} & \quad \{c\} & \quad \{d\} \\
a & b & c & d \\
a & c & f & d \\
a & d & c & g
\end{align*}
\]

\[\rightarrow \]

\[
\begin{align*}
\{c\} & \quad \{a\} & \quad \{d\} \\
c
\end{align*}
\]

\[
\begin{align*}
a & b & c & d \\
c & a & b & d \\
c & a & f & d \\
c & a & d & g
\end{align*}
\]

• MUST Join

\[
\begin{align*}
\{a\} & \\
\{\} & \\
\{c,f\} & \\
\{d\}
\end{align*}
\]

\[
\begin{align*}
\{e\} & \\
\{\} & \\
\{a\} & \\
\{d\}
\end{align*}
\]

\[
\begin{align*}
\{\} & \\
\{\} & \\
\{a,c\} & \\
\{d\}
\end{align*}
\]

Set intersection + max. age
Fixed-Point Iteration

Fixed-Point Iteration
Running MUST Analysis

Running MUST Analysis
Running MUST Analysis

Diagram showing a network of nodes labeled 'a', 'b', 'c', and 'd', with arrows indicating connections between them. The nodes are color-coded, with blue representing one set of data and gray representing another. The diagram illustrates the flow of data through the network.
Abstract State $\rightarrow$ Classification

• To determine the classification for $x$, check the abstract state before $x$

WCET Calculation

• Integration cache analysis results into IPET

// hit latency = 2; miss latency = 10
Maximize

$$(2 \cdot x_{0h} + 10 \cdot x_{0m}) + (2 \cdot x_{1h} + 10 \cdot x_{1m}) + (2 \cdot x_{2h} + 10 \cdot x_{2m}) + (2 \cdot x_{3h} + 10 \cdot x_{3m}) + (2 \cdot x_{4h} + 10 \cdot x_{4m}) + (2 \cdot x_{5h} + 10 \cdot x_{5m}) + (2 \cdot x_{6h} + 10 \cdot x_{6m}) + (2 \cdot x_{7h} + 10 \cdot x_{7m})$$

// cache constraints
$X_0 = x_{0h} + x_{0m}$
$0 \leq x_{0m} \leq X_0$
$x_{0h} = 0$

$X_1 = x_{1h} + x_{1m}$
$0 \leq x_{1h} \leq X_1$
$x_{1m} \leq 1$

$X_2 = x_{2h} + x_{2m}$
$0 \leq x_{2h} \leq X_2$
$x_{2m} = 0$

......