HW/SW Codesign

WCET Analysis

29 November 2017

Andres Gomez

gomeza@tik.ee.ethz.ch
Outline

Today’s exercise is one long question with several parts:
- Basic blocks of a program
- Static value analysis
- WCET of basic blocks (with cache analysis)
- ILP formulation

We will review and solve each subsection
WHY WCET ANALYSIS?

Keyword: Abstraction
BASIC BLOCKS
Basic Block

- **Definition:** A basic block is a sequence of instructions where the control flow enters at the beginning and exits at the end, without stopping in-between or branching (except at the end).

\[
\begin{align*}
  t1 & := c - d \\
  t2 & := e \times t1 \\
  t3 & := b \times t1 \\
  t4 & := t2 + t3 \\
  \text{if } t4 & < 10 \text{ goto L}
\end{align*}
\]
Basic Blocks

• *Determine basic blocks of a program:*

1. *Determine the block beginnings:*
   - the first instruction
   - targets of un/conditional jumps
   - instructions that follow un/conditional jumps

2. *Determine the basic blocks:*
   - there is a basic block for each block beginning
   - the basic block consists of the block beginning and runs until the next block beginning (exclusive) or until the program ends
Example

/* k >= 0 */
s = k;
WHILE (k < 10) {
    IF (ok)
        j++;
    ELSE {
        j = 0;
        ok = true;
    }
    k ++;
}
r = j;
Exercise

• From the hand out. Problem (a)

• Basic blocks:

```plaintext
a = 1;

if (b >= 0) then {
    i = a + 2;
    n = (a + 3)^2 + 2;
    m = a + 3;
    b = 6;
} else {
    l = a + 6;
    n = l + 8;
    i = a + 4;
}

while (n >= i) do {
    if (b > 5) {
        a = 3;
        l = l + 1;
    } else {
        m = m + 1;
        a = 4;
    }

    i = i + 1;
}
```
Solution
VALUE ANALYSIS
Value Analysis

• **Motivation:**
  – Provide access information to data-cache/pipeline analysis
  – Detect infeasible paths
  – Derive loop bounds

• **Method:** calculate intervals at all program points, i.e. lower and upper bounds for the set of possible values occurring in the machine program (addresses, register contents, local and global variables).
Value Analysis – Abstract Interpretation

- **abstract domain** – related to concrete domain by abstraction and concretization functions, e.g. L → Intervals where Intervals = LB × UB, LB = UB = Int ∪ {-∞, ∞}

- **abstract transfer functions** for each statement type – abstract versions of their semantics e.g. + : Intervals × Intervals → Intervals where [a,b] + [c,d] = [a+c, b+d] with + extended to -∞, ∞

- **a join function** combining abstract values from different control-flow paths e.g. t : Interval × Interval → Interval where [a,b] t [c,d] = [min(a,c), max(b,d)]
Value Analysis

- Intervals are computed along the CFG edges
- At joins, intervals are “unioned”

D1: [-2, +2]
D1: [-4, 0]
D1: [-4, +2]
Question

• Is the abstraction LB x UB always sufficient?
• Consider $f(x) = |x - 2|$, and $x = [1, 5]$. What is the abstraction of $f(x)$
Exercise

• Perform static value analysis of each block:

• Solution for first block:

\[ a, b, i, l, m, n \in (-\infty, \infty) \]

B1:
\[ a = 1 \]
\[ a \in [1, 1] \]
\[ b, i, l, m, n \in (-\infty, \infty) \]
\[ b \geq 0 \]
\[ a \in [1, 1] \]
\[ b, i, l, m, n \in (-\infty, \infty) \]
Solutions

B2:
\[ b \in [6, 6] \]
\[ m \in [4, 4] \]
\[ n \in [18, 18] \]
\[ i \in [3, 3] \]
\[ a \in [1, 1] \]
\[ l \in (-\infty, \infty) \]

B3:
\[ i \in [5, 5] \]
\[ n \in [15, 15] \]
\[ l \in [7, 7] \]
\[ a \in [1, 1] \]
\[ b \in (-\infty, 0) \]
\[ m \in (-\infty, \infty) \]

Join of B2 and B3:
\[ i \in [3, 5] \]
\[ n \in [15, 18] \]
\[ a \in [1, 1] \]
\[ b \in (-\infty, 6] \]
\[ l, m \in (-\infty, \infty) \]
Solutions

B4, B5:
\[ \begin{align*}
  i &\in [3, 5] \\
  n &\in [15, 18] \\
  a &\in [1, 1] \\
  b &\in (-\infty, 6] \\
  l, m &\in (-\infty, \infty)
\end{align*} \]

B6:
\[ \begin{align*}
  i &\in [3, 5] \\
  n &\in [15, 18] \\
  a &\in [3, 3] \\
  b &\in (5, 6] \\
  l, m &\in (-\infty, \infty)
\end{align*} \]

Join of B6 and B7:
\[ \begin{align*}
  i &\in [3, 5] \\
  n &\in [15, 18] \\
  a &\in [3, 4] \\
  b &\in (-\infty, 6] \\
  l, m &\in (-\infty, \infty)
\end{align*} \]

B7:
\[ \begin{align*}
  i &\in [3, 5] \\
  n &\in [15, 18] \\
  a &\in [4, 4] \\
  b &\in (-\infty, 5] \\
  l, m &\in (-\infty, \infty)
\end{align*} \]
Solutions

B8:

\[
\begin{align*}
i & \in [4, 6] \\
n & \in [15, 18] \\
a & \in [3, 4] \\
b & \in (-\infty, 6] \\
l, m & \in (-\infty, \infty)
\end{align*}
\]

Join of B8, B2, and B3:

\[
\begin{align*}
i & \in [3, 6] \\
n & \in [15, 18] \\
a & \in [1, 4] \\
b & \in (-\infty, 6] \\
l, m & \in (-\infty, \infty)
\end{align*}
\]

Join of B2 and B3:

\[
\begin{align*}
i & \in [3, 5] \\
n & \in [15, 18] \\
a & \in [1, 1] \\
b & \in (-\infty, 6] \\
l, m & \in (-\infty, \infty)
\end{align*}
\]
On loop bounds

• Without boundary conditions, there would be infinite loops

• Simple numerical counters are easy to analyze and calculate upper bounds

• From exercise: \( i \in [3, \infty) \) and \( n \in [15, 18] \)
  - Intersection shows an upper bound of 16 executions
CACHE ANALYSIS + BLOCK WCET’S
Cache Analysis

• How to statically precompute cache contents:

  – **Must Analysis**: For each program point (and calling context), find out which blocks are in the cache. Determines safe information about cache hits. Each predicted cache hit reduces WCET.

  – **May Analysis**: For each program point (and calling context), find out which blocks may be in the cache. Complement says what is not in the cache.

    Determines safe information about cache misses. Each predicted cache miss increases BCET.
Must Analysis

Abstraction

\[ \alpha \]

```
{ }  
{ }  
{z,x}  
{z}  
```

"young"  
"old"  

\[ \text{Age} \]
LRU: Transfer for Must

concrete
[ access s ]

abstract
[ access s ]

Age
“young”
“old”

{x, y, t}
{s}
{x, y, t}
{s}

{x}
{y, t}
{}
Contexts

- Cache contents depends on the context, i.e. calls and loops

  - First Iteration loads the cache:
    - Intersection looses most of the information.

- Distinguish as many contexts as useful
  - 1 unrolling for caches
  - 1 unrolling for branch prediction (pipeline)
Exercise

• Determine the WCET for each basic block
• Remember contexts for loops

• Solution for first block:

H for hit, M for miss,
ci is WCET in cycles for the i-th basic block
Initial cache state: (b,-,-,-)

B1:
  a = 1
  M, (a,b,-,-)
  if (b >= 0)
    H, (b,a,-,-)
  c1: 102
## Solutions

<table>
<thead>
<tr>
<th>Block</th>
<th>Ci</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>102</td>
</tr>
<tr>
<td>B2</td>
<td>406</td>
</tr>
<tr>
<td>B3</td>
<td>306</td>
</tr>
<tr>
<td>B4</td>
<td>201</td>
</tr>
<tr>
<td>B5</td>
<td>101</td>
</tr>
<tr>
<td>B6</td>
<td>202</td>
</tr>
<tr>
<td>B7</td>
<td>202</td>
</tr>
<tr>
<td>B8</td>
<td>102</td>
</tr>
</tbody>
</table>
A note on cache abstraction

Code:

```plaintext
while (n >= i) do {
    if (b > 5) {
        a = 3;
        l = l + 1;
    } else {
        m = m + 1;
        a = 4;
    }
    i = i + 1;
}
```

Cache analysis:

- **Join of B2 and B3:** 
  
  
  ```plaintext
  Join of B2 and B3: ({ },{ },{a},{n})
  ```

- **B8:**
  
  ```plaintext
  i = i + 1
  M, ({{i},{ },{a},{b}})
  H, ({{i},{ },{a},{b}})
  ```

- **Join of B8, B2, and B3:**
  
  ```plaintext
  Join of B8, B2, and B3: ({ },{ },{a},{}))
  ```

Given LRU policy, variable `i` should be in cache for while condition

Variable `i` is never in the abstract cache after the ‘join’

WCET is longer because the cache state is pessimistic. How could this be improved?
CALCULATING TOTAL WCET
Calculation of the WCET

- **Definition:** A program consists of \( N \) basic blocks, where each basic block \( B_i \) has a worst-case execution time \( c_i \) and is executed for exactly \( x_i \) times. Then, the WCET is given by

\[
WCET = \sum_{i=1}^{N} c_i \cdot x_i
\]

- the \( c_i \) values are determined using the static analysis.

- how to determine \( x_i \)?
  - structural constraints given by the program structure
  - additional constraints provided by the programmer (bounds for loop counters, etc.; based on knowledge of the program context)
Structural Constraints

Flow equations:

\[
\begin{align*}
d1 &= d2 = x_1 \\
d2 + d8 &= d3 + d9 = x_2 \\
d3 &= d4 + d5 = x_3 \\
d4 &= d6 = x_4 \\
d5 &= d7 = x_5 \\
d6 + d7 &= d8 = x_6 
\end{align*}
\]
Additional Constraints

s = k;
WHILE (k<10)
  if (ok)
    j++;
    ok = true;
  j = 0;
  k++;

loop is executed for at most 10 times:

x_3 <= 10 \cdot x_1

B5 is executed for at most one time:

x_5 <= 1 \cdot x_1
**WCET - ILP**

- **ILP with structural and additional constraints:**

  The program is executed once:

  \[
  WCET = \max \left\{ \sum_{i=1}^{N} c_i \cdot x_i \right\} \quad \text{subject to} \quad d_1 = 1 \wedge \\
  \sum_{j \in \text{in} (B_i)} d_j = \sum_{k \in \text{out} (B_i)} d_k = x_i, \quad i = 1 \ldots N \wedge \\
  \text{additional constraint s} \}
  \]
Total WCET Solution

\[ WCET = \max \left\{ \sum_{i=1}^{8} c_i \cdot x_i \right\} \]

- Objective function

\begin{align*}
  d_1 &= d_2 + d_3 = x_1 \\
  d_2 &= d_4 = x_2 \\
  d_3 &= d_5 = x_3 \\
  d_4 + d_5 + d_{11} &= d_6 + d_{12} = x_4 \\
  d_6 &= d_7 + d_8 = x_5 \\
  d_7 &= d_9 = x_6 \\
  d_8 &= d_{10} = x_7 \\
  d_9 + d_{10} &= d_{11} = x_8 \\
  x_5 &\leq 16 \cdot (x_2 + x_3)
\end{align*}

- Structural constraints

- Loop bounds