Quantitative Analysis of Execution Time: An ILP Formulation to Estimate WCET

(Based on the book: Introduction to embedded systems: A cyber-physical systems approach. 2011.)

Estagiário de Docência
M.Sc. Vinícius dos Santos Livramento

Prof. Dr. Luiz Cláudio Villar dos Santos

Embedded Systems - INE 5439
Federal University of Santa Catarina

November, 2014
Outline

1. Overview

2. Programs as Graphs

3. Optimization Formulation of WCET as an ILP

4. Using GNU GLPSOL to solve the ILP

5. Exercise to estimate WCET
Extreme-Case Analysis

- Execution time of the software is an example of quantitative property of an embedded system.
- Analysis of quantitative properties for conformance with quantitative constraints.
- Program analysis technique that can ensure that execution time constraints will be met.
Extreme-Case Analysis

- The typical quantitative analysis problem involves a software task defined by a program $P$, the environment $E$ in which the program executes, and the quantity of interest $q$.
- There is also a set $x$ that represents the inputs to the program $P$ and $w$ denotes the environment parameters.

$$ q = f_p(x, w) $$

- Defining the function $f_p$ completely is often neither feasible nor necessary; instead, practical quantitative analysis will yield extreme values for $q$ (highest or lowest values).
- We will focus on the **worst-case execution time**

$$ \max_{x,w} = f_p(x, w) $$

- The computed bound is considered tight for the case of WCET, which is essential to ensure correctness of critical tasks.
Modular Exponentiation

```c
#define EXP_BITS 32

typedef unsigned int UI;

UI modexp(UI base, UI exponent, UI mod) {
    int i;
    UI result = 1;

    i = EXP_BITS;
    while(i > 0) {
        if ((exponent & 1) == 1) {
            result = (result * base) % mod;
        }
        exponent >>= 1;
        base = (base * base) % mod;
        i--;    
    }
    return result;
}
```
Basic Blocks

```c
#define EXPBITS 32

typedef unsigned int UI;

UI modexp(UI base, UI exponent, UI mod) {
    int i;
    UI result = 1;
    i = EXPBITS;
    while (i > 0) {
        if ((exponent & 1) == 1) {
            result = (result * base) % mod;
        }
        exponent >>= 1;
        base = (base * base) % mod;
        i--;
    }
    return result;
} BB1
BB2
BB3 BB4 BB5 BB6
```
Control-Flow Graphs (CFG)

- A control-flow graph (CFG) of a program $P$ is a directed graph $G = (V, E)$, where the set of vertices comprises basic block of $P$, and the set of edges $E$ indicates the flow of control between basic blocks.
Control-Flow Graphs (CFG)

```
result = 1;
i = EXP_BITS;

(i > 0)?

(result * base) % mod;

exponent >>= 1;
base = (base * base) % mod;
i--;

return result;
```
Factors Determining the WCET

- **Loop bounds** must be considered to ensure that a program terminates or not.
  - In order to guarantee this, one must determine a bound on the number of times that loop will execute in the worst case.
  - The problems of determining bounds on loop iterations or recursion depth are undecidable.
  - We will focus on simple loops in which determining bounds is trivial.

- **Exponential path space** comes from the fact that the number of program paths can be very large – exponential in the size of the program.
  - We will use an implicit path enumeration technique (IPET) to overcome this problem.
Optimization Formulation

- Given a program $P$, let $G = (V, E)$ denote the CFG. Let $n = |V|$ be the number of basic blocks in $G$ and $m = |E|$ denote the number of edges. We refer to the basic block by their index $i$, where $i$ ranges from 1 to $n$.
- We assume that the CFG has a unique start (source) node $s$ and a unique end (sink) node $t$.
- Let $x_i$ denote the number of times a basic block $i$ is executed (an integer). Let $x = (x_1, x_2, ..., x_n)$ be a vector of variables recording execution counts.
Implicit Path Enumeration (IPET)

- The implicit path enumeration (IPET) can be formalized through flow constraints using the theory of network flow.
- The flow from source node $s$ to sink node $t$ corresponds to a unit flow.

\[ x_1 = 1 \]
\[ x_n = 1 \]
Implicit Path Enumeration (IPET)

- The conservation flow implies that for each BB \( i \), the incoming flow to \( i \) equals to the outgoing flow from \( i \).
- Let the additional variable \( d_{ij} \) denote the number of times that an edge is executed.
- Let \( P_i \) be the set of predecessors of node \( i \) and \( S_i \) the set of successors.

\[
x_i = \sum_{j \in P_i} d_{ji} = \sum_{j \in S_i} d_{ij}
\]
Loop Bounds

- Note that the constraints presented previously impose no upper bound on $x_2$ or $x_3$.
- We need to add an additional constraint otherwise the WCET will be infinite.
- The following single constraint suffices: $x_3 \leq 32$.
- From this constraint on $x_3$, we derive that $x_2 \leq 33$ and also upper bounds on $x_4$ and $x_5$. 
**WCET as an ILP**

- Now we can formulate the overall optimization problem to determine the WCET.
- Let $w_i$ (defined later) denote an upper bound on the execution of the basic block $i$, then the WCET is given by the maximum $\sum_{i=1}^{n} w_i x_i$ over $x_i$.
- Putting together the objective and constraints we get:

  \[
  \begin{align*}
  \text{maximize} & & \sum_{i=1}^{n} w_i x_i \\
  \text{subject to} & & x_1 = x_n = 1 \\
  & & x_i = \sum_{j \in P_i} d_{ji} = \sum_{j \in S_i} d_{ij}, \ i = 1..n \\
  & & x_i \in \mathbb{Z}_+ \ i = 1..n
  \end{align*}
  \]

- This optimization problem forms a system of linear equations in which the variables to be optimized are integer. This problem is known as an **integer linear programming** (ILP).
GNU GLPSOL

- The GLPK package supplies a solver for large scale linear programming (LP) and for integer programming (ILP), called glpsol.
- The solver supports different file formats and also an API to write programs in C/C++.
- In this experiment we will adopt the CPLEX LP format.
- We will use the following reserved keywords:
  - **Maximize** or **Minimize** according to the objective.
  - The objective itself, e.g. $+2x1 + 1x2$.
  - **Subject To** define the constraints, where each line states a constraint labeled as $c_i$, for each $i = 1...n$ constraints.
  - **Bounds** to specify the variable bounds, e.g. $x1 \geq 0$.
  - **General** to specify the integer variables.
  - **End** to specify the end of file.
Example of ILP Formulation using GLPSOL

Maximize

\[ +2x_1 + 1x_2 \]

Subject To

\[ c_1 : +x_1 + 2x_2 \leq 5 \]
\[ c_2 : +3x_1 + x_2 \leq 10 \]

Bounds

\[ x_1 \geq 0 \]
\[ 0 \leq x_2 \leq 7 \]

General

\[ x_1x_2 \]

End
Solving an ILP Formulation using GLPSOL

- Write the program in a text file and save as `problem.lp`. The `.lp` format corresponds to the CPLEX file format.
- Run the following line command to optimize and write the solution:
  - `$ glpsol --lp problem.lp -o solution`
- The result should be as follows:

```
Solving LP relaxation...
GLPK Simplex Optimizer, v4.45
2 rows, 2 columns, 4 non-zeros
* 0: obj = 0.0000000000e+00 infeas = 0.000e+00 (0)
* 2: obj = 7.0000000000e+00 infeas = 0.000e+00 (0)
OPTIMAL SOLUTION FOUND
Integer optimization begins...
  2: mip = not found yet <= +inf (1; 0)
  2: >>>>> 7.0000000000e+00 <= 7.0000000000e+00 0.0% (1; 0)
  2: mip = 7.0000000000e+00 <= tree is empty 0.0% (0; 1)
INTEGER OPTIMAL SOLUTION FOUND
Time used: 0.0 secs
Memory used: 0.1 Mb (54403 bytes)
Writing MIP solution to `solution`...
```
Analysing the results from the GLPSOL solver

- The solution file has a label **obj** with the optimal solution and tables with the variables values in a row called **Activity**, as follows:

```
Problem:
Rows: 2
Columns: 2 (2 integer, 0 binary)
Non-zeros: 4
Status: INTEGER OPTIMAL
Objective: obj = 7 (MAXimum)

<table>
<thead>
<tr>
<th>No.</th>
<th>Row name</th>
<th>Activity</th>
<th>Lower bound</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c1</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>c2</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No.</th>
<th>Column name</th>
<th>Activity</th>
<th>Lower bound</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x1</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>x2</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>

Integer feasibility conditions:
KKT.PE: max.abs.err = 0.00e+00 on row 0  
max.rel.err = 0.00e+00 on row 0  
High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0  
max.rel.err = 0.00e+00 on row 0  
High quality

End of output
Problem Details

<table>
<thead>
<tr>
<th>Basic Block</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

- State the objective
- Define the flow constraints
- Specify the bound of each variable
- Ensure that variables are integer
- The correct WCET is 292 cycles
Quantitative Analysis of Execution Time: An ILP Formulation to Estimate WCET

(Based on the book: Introduction to embedded systems: A cyber-physical systems approach. 2011.)

Estagiário de Docência

M.Sc. Vinicius dos Santos Livramento

Prof. Dr. Luiz Cláudio Villar dos Santos

Embedded Systems - INE 5439
Federal University of Santa Catarina

November, 2014