Performance Validation of SW CS 4271 Lectures 8, 9

Abhik Roychoudhury
National University of Singapore

http://www.comp.nus.edu.sg/~abhik/

Timing in Embedded Systems

Example Set-up

```
system-level view of a video encoder in a video phone
```

```
raw video stream
media processor + micro-architecture
```

```
video decoding and playout at the receiver at a specified frame-rate
```

```
minimum buffer size required?
```

Need to look inside the different processing tasks, and analyze their timing!

Timing analysis of programs

```
Estimating uninterrupted software execution time on a given hardware (processor).
```

```
A building block for more complicated performance analysis.
```

```
Helps estimate performance of a design point.
```

```
Serves as a sub-routine for Design Space Exploration.
```

Timing analysis of programs

```
Schedulability analysis of Hard Real-time systems.
```

```
Such analysis assumes knowledge of WCET of each task being scheduled.
```

```
WCET stands for Worst-case Execution Time
```

```
Rate Monotonic scheduling with tasks T1,…,Tn
```

```
Computation times C1,…,Cn
```

```
Period = deadline D1,…,Dn
```

```
Here C1,…,Cn are the WCET (not average execution times of the programs)
```

Time is abstracted!

```
Our programming languages do not mention time
```

```
C, Java, C/C++, C++
```

```
Even an instruction takes variable time
```

```
Hit/miss in instruction cache
```

```
Hit/miss in data cache
```

```
Pipeline stalls
```

```
Data hazard
```

```
Resource contention
```

```
Branch prediction ...
```

```
Need timing analysis of programs!
```

Difficulty to analyze & debug!
Organization
- Software timing analysis
- WCET analysis
- System level analysis
- Schedulability analysis
- Design issues to improve timing predictability
- Scratchpad memories

WCET
- Worst Case Execution Time (WCET) of a program for a given hardware platform.
- Sequential Terminating Programs.
- Gets input, computes, produces output.
- Many inputs are possible.
- Leads to different execution times.
- WCET: An upper bound on the execution time for all possible inputs.

Why need analysis?
- To find WCET of a program, execute it for all possible inputs.
- WCET by measurement.
- Exponentially many possible inputs in terms of input size.
- Insertion sort program
- Similar problems will be encountered for WCET Analysis via platform simulation.
- Need access to platforms/simulators also!
- Go for static analysis.

WCET by measurement?
- What about single path programs such as matrix multiplication?
- Execution path is independent of input data.
- Still execution time can be variable.
- Latency of floating point operation (e.g., multiplication) depends on the input data.
- Not possible to try it on all possible platforms and then choose one.
- Often trying to decide the platform as well.

Why Platform-aware Analysis?

Distribution of execution times across inputs in a quicksort program on a simple and complex processor

WCET Analysis
- Employ static analysis to compute an upper bound on actual WCET (Estimated WCET)
- Run program on selected inputs get a lower bound on actual WCET (Observed WCET)

Estimated WCET ≥ Actual WCET ≥ Observed WCET
**BCET and WCET**

Diagram showing the distribution of execution time comparing Actual BCET, Estimated BCET, Actual WCET, and Estimated WCET.

**Program path analysis**
- All paths in control flow graph are not feasible.

**Micro-architectural modeling**
- Dynamically variable instruction execution time.
  - Cache, Pipeline, Branch Prediction
  - Out-of-order Pipelines

**WCET Analysis**

**Clarification: Control flow graph**
- \( x = 1; y = 0; z = 0; \)
- \( \text{while} (x < 10) \)
  - \( \text{if} (x > 5) \)
    - \( y = y + x; \)
  - \( \text{else} \) \( z = z + x; \)
  - \( x = x + 1; \)
- \( \text{printf}(\ldots); \)

**Exercise: CFG**
- procedure Check_data()
  - \{ \)
    - \( \text{int} \ i = 0, \text{morecheck} = 1, \text{wrongone} = -1, \text{datasize} = 10; \)
    - \( \text{L:} \) \( \text{while} (\text{morecheck}) \)
      - \( \text{LB:} \) \{ \)
        - \( \text{if} (\text{data}[i] < 0) \)
          - \( \text{A:} \) \{ \)
            - \( \text{wrongone} = i; \)
            - \( \text{morecheck} = 0; \)
          - \}
        - \( \text{else} \) \{ \)
          - \( \text{B:} \) \{ \)
            - \( \text{++i} \) \( \text{if} (\text{++i} \geq \text{datasize}) \)
              - \( \text{morecheck} = 0; \)
            - \}
          - \}
        - \}
    - \( \text{if} (\text{wrongone} \geq 0) \)
      - \( \text{C:} \) \{ \)
        - \( \text{h} \) \( \text{andle_exception(wrongone); return 0;} \)
      - \}
    - \( \text{else} \) \{ \)
      - \( \text{return i;} \)
    - \}
  - \}

**Exercise on CFG**
- Construct the CFG of the procedure given in the previous slide.
- How to construct an inter-procedural CFG for a program with many procedures?

```c
main(){
  f1(){
    \}
  f2(){
    \}
  \}
}
```

**Why all paths may not be feasible?**
- \( \text{if} (x > 0) \)
  - \( \text{y} = 1; \)
  - \( \text{else} \)
    - \( y = 2; \)
  \}
- \( \text{if} (x > 1) \)
  - \( \text{z} = 10; \)
  - \( \text{else} \)
    - \( z = 20; \)
  \}
- \( \ldots \)
Restrictions of analysis – (1)

- Static analysis need not be on source program.
- We can perform static analysis on assembly code of a given program.
- The analysis is only for time taken, and not for the memory locations / values accessed.
- No restriction on program data structures used for WCET analysis.
- What about control flow?

Restrictions of analysis – (2)

- Restrictions on control flow
  1. No unbounded loops
     - Common sense.
     - Otherwise how to guarantee time?
  2. No unbounded recursion
     - Similar issue.
  3. No dynamic function calls
     - Need to statically know the functions called, and the possible call sites of these functions.

Organization of WCET Analysis

- What is Timing Analysis?
  - Primarily Control flow.
  - Modeling timing effects of Micro-architecture.
  - Cache, pipeline.

Timing Schema

- One of the first works on WCET analysis.
- Basically, perform control flow analysis to find the “longest” program path.
- The notion of “longest” is weighted
  - Take into account the cost of executing individual program elements.
  - Timing schema is a simple way of composing these costs.
- Does not work on Control Flow Graphs
  - Works on Abstract Syntax Tree

Example

```plaintext
sum = 0;
for (i=0; i< 10; i++){
    if (i % 2 == 0)
        sum += i;
    if (sum < 0)
        break;
}
return sum;
```

AST and CFG

- Hierarchy
  - AST shows the different scopes at different levels
  - CFG has no hierarchy.
- Loops
  - AST is tree, free from cycles
  - Any loop in the program is a cycle in the CFG.
AST and CFG

Timing schema – (works on AST)
- $\text{Time}(S_1; S_2) = \text{Time}(S_1) + \text{Time}(S_2)$
- $\text{Time}(\text{if } B \{ S_1 \} \text{ else } \{ S_2 \})$
  - $= \text{Time}(B) + \max(\text{Time}(S_1), \text{Time}(S_2))$
- $\text{Time}(\text{while } B \{ S_1 \})$
  - $= (n+1) \cdot \text{Time}(B) + n \cdot \text{Time}(S_1)$
  - $n$ is the loop bound.
- $\text{Time}(\text{for}(\text{Init}; B; \text{Incr.})\{ S \})$
  - $= \text{Time}(\text{Init}) + (n+1) \cdot \text{Time}(B) + n \cdot \text{Time}(S) + n \cdot \text{Time}(\text{Incr.})$

Timing schema

Problems with Timing Schema
- Language Level:
  - Just a control flow analysis.
  - Insensitive to knowledge of infeasible paths.
- Compiler level:
  - How to integrate effect of compiler opt?
  - Easy to handle – schema on optimized code.
- Architecture level:
  - Instructions take constant time – Not true.
  - Cache hits, pipelining and other performance enhancing features.

Infeasible paths
- Infeasible sequence of statements in general
- if $\{ i++ \}$
  - $K = 1$
  - else $\{ K = 0 \}$
- if $\{ K < 5 \}$
  - $J++$
  - else $\{ J-- \}$

The issue with Infeasible Paths
- What if $T_1 > T_2$ and $S_1$ is executed only in the first loop iteration?

Infeasible paths
- Such infeasible paths should not be a witness to our WCET estimate.
Infeasible Path handling in Timing Schema

```java
if (J== 0) {
    K = 1;
} else {
    K = 10;
}
```

How will timing schema work on this example?

**Working of timing schema**

Time(first-if-statement): $1 + \max(1,5) = 6$

Time(second-if-statement): $1 + \max(5,1) = 6$

Estimated worst case time: $6 + 6 = 12$

Actual worst-case time: $2 \times 4 = 8$

Why?
Where is the overestimate from?

Control flow graph (CFG)

```
J == 0 ??
  Y N
K = 1
K < 5 ??
  Y N
  J++
```

Infeasible path in CFG

```
J == 0 ??
  Y N
K = 1
K < 5 ??
  Y N
  J++
```

Modeling of control flow

- **Path-based**
  - Enumerate paths and find longest path
  - Expensive!
  - Need to remove longest path if it is infeasible.
- **Tree-based**
  - Bottom-up pass of Syntax Tree
  - Timing Schema
  - Difficult to integrate infeasible path info
- **Integer Linear Programming**
  - Can take into account certain infeasible path information if available.
  - Efficient solvers available e.g. CPLEX
  - Forms the back-end of most state-of-the-art timing analyzers.

Integer Linear Programming

- **ILP:** Integer Linear Programming
  - Variables and linear constraints on them.
  - Cost function (linear) to optimize.

\[
f = 3x + 5y + z \\
0 \leq x, y, z \leq 100 \\
x + y + z = 200 \\
x + 2y \leq 160 \\
\]

Optimal: $f = 520; x = 40; y = 60; z = 100$
Non-Optimal: $f = 480; x = 80; y = 30; z = 80$
ILP Modeling

We are dealing with aggregated execution counts of nodes/edges of CFG.

Maximize

\[ \text{Time} = c_1 N_1 + c_2 N_2 + c_3 N_3 + c_4 N_4 + c_5 N_5 + c_6 N_6 + c_7 N_7 + c_8 N_8 \]

\[ 1 = N_1 = E_{1,2} \]
\[ E_{2,3} + E_{4,5} = N_2 = E_{2,7} \]
\[ E_{3,4} = N_3 = E_{3,5} \]
\[ E_{5,6} = N_5 = E_{5,8} \]
\[ E_{5,8} = N_8 = E_{8,7} \]
\[ E_{8,7} + E_{2,7} = N_7 = 1 \]
\[ E_{6,2} \leq 10 \]

Blocks 3 and 6 are never executed in same loop iteration

\[ N_3 + N_6 \leq \text{loopbound} \]

Edges (2,3) and (5,6) are not executed together.

\[ E_{2,3} = E_{5,6} \]

Infeasible path

The break statement is executed at most once.

\[ N_4 \leq 1 \]

How to express this inf. path constraint?

\[ J = 0 ?? \]
\[ K = 1 \]
\[ K < 5 ?? \]
\[ J = J++ \]
\[ Y \]
\[ \text{No} \]
\[ \text{Yes} \]
\[ J = J++ \]
Exercise: What are the infeasible paths?

```c
procedure Check_data()
{i
    int i = 0, morecheck = 1, wrongone = -1, datasize = 10;
    while (morecheck)
        {
            if (data[i] < 0)
                {
                    wrongone = i;   morecheck = 0;
                }
            else
                {
                    if (++i >= datasize) morecheck = 0;
                }
            }
        }
    if (wrongone >= 0)
        {
            handle_exception(wrongone);
            return 0;
        }
    else
        return i;
}
```

Organization

- What is Timing Analysis?
- An Early solution – Timing schema
  - Primarily Control flow.
  - Modeling timing effects of Micro-architecture.
    - Cache, pipeline.

The two phases

- WCET analysis involves
  - Program path analysis – ILP
  - Micro-architectural modeling.
- How do the two analyses interact?
  - Time = c1*N1 + c2*N2 + c3*N3 + ...
  - cl, c2, ...: exec time of basic blocks 1,2, ...
  - N1, N2, ...: exec count of basic blocks 1,2, ...
  - cl, c2, ... are estimated by µ-arch. modeling
  - N1, N2, ... are fixed by control flow analysis via Integer Linear Programming (ILP).

Instruction Cache Modeling

- One concrete hardware data structure.
- With no hardware modeling, all instructions should be taken as misses.
- Instead we can categorize some instructions as “always hit”
  - Coarse modeling.
- For certain instructions, even the “worst case” may not be a miss!

Categorization

-… of instructions
  - AH (always hit)
  - AM (always miss)
  - PS (Persistent: second and all further executions are guaranteed to produce a hit)
  - Effect of cold misses
  - NC (not AH, AM, PS)

Cache - basics

- Redundant storage to reduce memory access time.
- Many memory blocks map to a single cache line
- F: Memory Block → Cache lines
  - Given a memory block m, F(m) returns the set of cache lines it can map to.
  - If F(m) is always a singleton set, then we have a direct mapped cache.
  - If |F(m)| = n, we have n-way set associative cache.
  - If F(m) = Set of all cache lines, then we have a fully associative cache (any memory block can map to any cache line).
Cache - basics
- Fully associative with LRU policy.
- Cache lines = L_1, L_2, ..., L_n
- L_1 is the youngest line
- L_n is the oldest line
- Do not refer to physical cache lines
- Memory blocks = S_1, ..., S_m
- Any block S_i can map to any cache line L_j during program execution

Concrete cache update
- s is in cache

Abstract cache state
- In the concrete cache state c, if a block is in cache line x, its age is x
- Cache line 1 is youngest.
- In the abstract cache state c', each line x contains a set of memory blocks
  B ∈ c'(L_x) at a program point p means …
- When control reaches p, B may (must) be in cache with min (max) age = x
- Direction of approximation in abstraction.

May analysis
1. In cache in some path.
2. If so, take min. age

Must analysis
1. In cache in both paths.
2. If yes, take max age.
How to use such analyses?

- Let I be an instruction at control loc. CL.
- Let M be the memory block containing I.
- Consider cache state at CL obtained via “must analysis”.
  - If M is in some cache line within this abstract cache state, then I is Always Hit.
- For cache state at CL obtained via “may analysis”
  - If M is not in any cache line within this abstract cache state, then I is Always Miss.
- For abstract cache state at CL obtained from persistence analysis
  - If M is not in the evicted line, then I is Persistent.

Use of may-must analyses

- Let hit_time = t1, miss_time = t2
- Number of accesses of I == #I (ILP variable)
  - I is AH
    - #I * t1 = contribution of I to WCET
  - I is AM
    - #I * t2 = contribution of I to WCET
  - I is PS
    - (#I - #miss(I)) * t1 + #I * t2 = contribution of I to WCET
  - Formulation is still linear, solve via ILP.

Improving precision

- If we can bound the number of misses of instr. I (via constraints)
  - No need to reduce exec. Time of I to constant
  - Contribution of I to WCET
    - #miss(I) * t2 + (#I - #miss(I)) * t1
    - Need constraints to bound #miss(I)
    - How to develop such constraints?
  - ILP Expensive !!
    - See cache conflict graph approach in textbook
      - Pages 147 – 149.

Micro-arch. modeling so far

- Modeling timing effects of I-cache
  - Abstract Interpretation to categorize instr
  - ILP based modeling is more expensive.
  - I-cache does not have timing anomalies
  - Can assume all accesses are misses.
  - Very pessimistic, but estimate still safe!
  - For certain processors, even this is not true!
    - Adding worst-case of each instruction may produce an estimate lower than the global worst-case!

Pipelined execution

Divide the execution of an instruction into stages
Instruction I+1 can proceed before I completes
Increased throughput, lower overall execution time
Out-of-order pipeline:

- Mem => I-buffer (in-order)
- IBUF => ROB (in-order)
- ROB => FU (out-of-order, Instr still in ROB)
- FU => ROB (out-of-order, forward data)
- Update register file, free ROB entry (in-order)

O-o-o execution (1):

```
# Ready Instruction
Cycle
A 0 mult r3 r1 r2
B 1 add r3 r3 r8
C 2 and r3 r3 0xff
D 3 addu r5 r4 r8
E 4 mult r5 r5 r6
```

```
Instruction sequence
MULTU 1 - 4 cycles
alu 1 cycle
```

```
Latencies
MULTU 1 - 4 cycles
alu 1 cycle
```

```
Instruction A executes 4 cycles
```

O-o-O execution (2):

```
# Ready Instruction
Cycle
B 1 add r3 r3 8
C 2 and r3 r3 0xff
D 3 addu r5 r4 r8
E 4 mult r5 r5 r6
```

```
Instruction sequence
MULTU 1 - 4 cycles
alu 1 cycle
```

```
Latencies
MULTU 1 - 4 cycles
alu 1 cycle
```

```
Instruction A executes 3 cycles
```

Difficulty in modeling:

```
# Ready Instruction
Cycle
A 0 mult r3 r1 r2
B 1 add r3 r3 r8
C 2 and r3 r3 0xff
D 3 addu r5 r4 r8
E 4 mult r5 r5 r6
```

```
Instruction sequence
MULTU 1 - 4 cycles
alu 1 cycle
```

```
Latencies
MULTU 1 - 4 cycles
alu 1 cycle
```

```
Instruction A executes 4 cycles
```

Timing Anomaly:

- Overall WCET of an instruction sequence cannot be obtained from WCET of each instruction
- Need to consider all possible execution times of each instruction to safely estimate WCET!
- Expensive enumeration
- Very different from cache modeling
- Worst-case cache behavior of an instruction sequence can be safely estimated by considering all cache accesses as misses

Difficulty in modeling:

- Expensive enumeration
- Different from cache modeling
- Worst-case not found by adding up worst-cases of code fragments – non compositional
- Efficient analysis developed to overcome this (not discussed).

Summing up …

- WCET Analysis
  - Program flow modeling (typically by ILP)
  - Combine reasoning about timing of program fragments.
  - Exploiting infeasible path information.
  - Difficult to use model checking for this purpose.
  - Micro-architectural modeling (customized analysis)
  - Exact time of each instruction is not constant.
  - Worst-case not found by adding up worst-cases of code fragments – non compositional.
  - Efficient analysis developed to overcome this (not discussed).
Chronos Tool for WCET analysis

Chronos Tool for WCET analysis

Chronos – views of program

Set loop bounds

Set loop bounds

Constraints specified by user at source code level. These are automatically translated to node counts of assembly level control flow graph.

Flexible micro-arch modeling

Flexible micro-arch modeling

WCET Estimation

WCET Estimation

Simulation vs Estimation

Simulation vs Estimation

Simulation produces Observed WCET. Estimation produces Estimated WCET. Observed WCET ≤ Actual WCET ≤ Estimated WCET.