Embedded Systems
Lecture 11: Worst-Case Execution Time

Björn Franke
University of Edinburgh
Overview

• Motivation

• Worst-Case Execution Time Analysis
  • Types of Execution Times
  • Measuring vs. Analysing
  • Flow Analysis
  • Low-Level Analysis
  • Calculation
Motivation: Characteristics of Real-Time Systems

- Concurrent control of separate system components
- Reactive behaviour
- **Guaranteed response times**
- Interaction with special purpose hardware
- Maintenance usually difficult
- Harsh environment
- Constrained resources
- Often cross-development
- Large and complex
- Often have to be extremely dependable
What is the “Execution Time” of a program?

The WCET/BCET is the longest/shortest execution time possible for a program. Must consider all possible inputs—including perhaps inputs that violate specification.
Why may we care about the WCET?

• We are interested in WCET to . . .
  
  • perform schedulability analysis
  
  • ensure meeting deadlines
  
  • assess resource needs for real-time systems
  
• WCET accuracy may be safety-critical!
And why may we care about the BCET?

• We are interested in BCET to.
  • benchmark hardware
  • assess code quality
  • assess resource needs for non/soft real-time systems
  • ensure meeting livelines (new starting points)
What is the “Execution Time” of a program?

- Approaches for approximating WCET or BCET

  - **Measuring**: Measure run time of program on target hardware

  - **Analysis**: Compute estimate of run time, based on program analysis and model of target hardware

  - **Hybrid**: Combine measurements with program analysis
Measuring WCET/BCET

• Execution time may depend on program inputs
  • In this case, quality of measurements depends on judicious choice of inputs

• Execution time may depend on execution context (cache content, state of pipeline, ...)

• Typically need to add safety margin to best/worst result measured

• Extensive testing/measurement still common practice
Measuring Program Run Times

• Call OS timing routines
  • Account for cost of calls to timing routines themselves
• Access hardware timers directly
• Use external hardware
  • Oscilloscope, Logic analyser
• Count emulator cycles
• High water marking
  • Continuously record max execution times
  • Standard feature of RTOSs
  • May include this in shipped products
  • Read at service intervals
Analysing WCET/BCET

• Instead of measuring execution times, compute them

• **Advantages**
  • Can ensure safety of result
  • Saves testing effort

• **Disadvantages**
  • Try to be as tight as possible—may not always succeed
  • Typically requires extensive analysis effort

• **Accuracy depends on**
  • Complexity of hardware
  • Program structure
  • Quality of hardware model
  • Program analysis capabilities
Analysing WCET/BCET
Flow Analysis

- Analyse dynamic behaviour of program
  - Number of loop iterations, Recursion depth, Input dependences, Infeasible paths, Function instances, ...

- Get information from
  - Static Analysis
  - Manual Annotation

- Analysis level
  - Object code
  - Source code (may need non-trivial mapping to object code)
Flow Analysis

Structurally possible flows (infinite)

Basic finiteness

Statically allowed

Actual feasible paths

WCET found here=desired result

WCET found here=overestimation

[EngblomES01]
Flow Analysis

- The set of **structurally possible** flows for a program, i.e. those given by the structure of the program, is usually infinite, since e.g. loops can be taken an arbitrary number of times.

- The executions are made finite by bounding all loops with some upper limit on the number of executions (**basic finiteness**).

- Adding even more information, e.g. about the input data, allows the set of executions to be narrowed down further, to a set of **statically allowed** paths. This is the “optimal” outcome of the flow analysis.
Flow Analysis

```
const int max = 100;
foo (float x) {
A:   for(i = 1; i <= max; i++) {
B:     if (x > 5)
C:       x = x * 2;
    else
D:     x = x + 2;
E:     if (x < 0)
F:       b[i] = a[i];
G:     bar (i)
}]
```

- **Loop bounds**: Easy to find in this example; in general, very difficult to determine

- **Infeasible paths**: Can we exclude a path, based on data analysis? A-B-C-E-F-G is infeasible—since if x>5, it is not possible that x * 2 < 0.

  Well, really? What about integer overflows? *Must be sure that these do not happen in the example...*
Low-Level Analysis

- Determine execution time for program parts
- Account for hardware effects (pipeline, caches...)
- Work on object code
- Exact analysis generally not possible
Low-Level Analysis

- Global Low-Level Analysis
  - Considers execution time effects of machine features that reach across entire program
  - Instruction/data caches, branch predictors, translation lookaside buffers (TLBs)
- Local Low-Level Analysis
  - Considers machine features that affect single instruction & its neighbours
  - Scalar/superscalar pipelines
Local Low-Level Analysis - Pipelining

- Pipeline effect of two successive instructions

- Pipeline overlap reduces overall computation time by $\delta = -2$
Local Low-Level Analysis - Pipelining

- Pipelining effect of three successive instructions

- Reduction of combining three instructions can be larger than sum of savings when combining them pair-wise!
Global Low-Level Analysis - Caches

- Instruction Caches
  - Predictable from control flow

- Data Caches
  - No simple way to predict accesses
  - Very difficult analysis problem

- Unified Caches
  - Very pessimistic as a result of combining instructions & data
Global Low-Level Analysis - Caches

- May split loops to differentiate between first and successive loop iterations
- Must combine with pipelining effects
WCET Calculation

- Task: Find the path that results in the longest execution time
- Several approaches in use
- Properties of approaches
  - Program flow allowed
  - Object code structure (optimisations?)
  - Pipeline effect modelling
  - Solution complexity
WCET Calculation

- Path-based
- Constraint-based
  Implicit Path Enumeration Technique - IPET
- Structure-based
WCET Calculation

(a) Control-flow graph with timing

(b) Path-based calculation

(c) IPET calculation

(d) Structure-based calculation

**Syntax-tree**

\[ T(seq(S1,S2)) = T(S1) + T(S2) \]

\[ T(if(Exp) S1 else S2) = T(Exp) + max(T(S1),T(S2)) \]

\[ T(loop(Exp,Body)) = T(Exp) + (T(Exp) + T(Body)) \times (\text{maxiter}-1) \]

**Transformation rules**

**Fig. 5. Bound calculation.**

**longest path marked**

// Unit timing

\[ t_{\text{path}} = 31 \]

\[ t_{\text{header}} = 3 \]

// WCET Calc

\[ WCET = t_{\text{header}} + t_{\text{path}} \times (\text{maxiter}-1) = 3 + 31 \times 99 = 3072 \]

// Start and exit constraints

\[ x_{\text{start}} = 1, x_{\text{exit}} = 1 \]

// Structural constraints

\[ x_{\text{start}} = x_{\text{start}}A \]

\[ x_{A} = x_{\text{start}}A + x_{HA} = x_{\text{exit}} + x_{AB} \]

\[ x_{B} = x_{AB} + x_{BC} + x_{BD} \]

\[ x_{C} = x_{BC} + x_{CE} \]

\[ x_{H} = x_{FH} + x_{GH} = x_{HA} \]

\[ x_{\text{exit}} = x_{\text{exit}}A \]

// Loopbound constraint

\[ x_{A} \leq 100 \]

// WCET Expression

\[ WCET = \max(x_{A} \times 3 + x_{B} \times 5 + \ldots + x_{H} \times 2) = 3072 \]

**[Wilhelm+08]**
Path-Based Bound Calculation

• Upper bound for a task is determined by computing bounds for different paths in the task, searching for the overall path with the longest execution time.

• Defining feature is that possible execution paths are represented explicitly.

• Natural within a single loop iteration, but problems with flow information extending across loop nesting levels.

• Number of paths is exponential in the number of branch points.

• Possibly requiring heuristic search methods.
Implicit Path Enumeration

- Program flow and basic block execution time bounds are combined into sets of arithmetic constraints.

- Each basic block and program flow edge in the task is given a time coefficient, expressing the upper bound of the contribution of that entity to the total execution time every time it is executed.
Structure-based Bound Calculation

- Upper bound is calculated in a bottom-up traversal of the syntax tree of the task combining bounds computed for constituents of statements according to combination rules for that type of statement.

- Not every control flow can be expressed through the syntax tree

- Assumes straight-forward correspondence between source structures and the target program
  - Not easily admitting code optimisations

- In general, not possible to add additional flow information (as in IPET).
Summary

• Motivation

• Worst-Case Execution Time Analysis
  • Types of Execution Times
  • Measuring vs. Analysing
  • Flow Analysis
  • Low-Level Analysis
  • WCET Calculation
Preview

- Real-Time Operating Systems
- MQX