Design and Analysis of Real-Time Systems
Static WCET Analysis

Jan Reineke

Advanced Lecture, Summer 2013
What does the execution time of a program depend on?

**Input-dependent control flow**

**Microarchitectural State**

Pipeline, Memory Hierarchy, Interconnect
Formalization of WCET Analysis Problem

Consider all possible program inputs

Consider all possible initial states of the hardware

\[ WCET_H(P) := \max_{i \in \text{Inputs}} \max_{h \in \text{States}(H)} ET_H(P, i, h) \]

Measuring the execution time for all inputs and all hardware states is not feasible in practice:

- There are too many.
- We cannot control the initial hardware states.

\rightarrow Need for approximation!
High-level Requirements for WCET Analysis

- Upper bounds must be safe, i.e. not underestimated.
- Upper bounds should be tight, i.e. not far away from real execution times.
- Analysis effort must be tolerable.
Standard WCET Analysis Approach Today: Divide and Conquer + Abstraction

1. **Divide:** split program into fragments (e.g. basic blocks).
2. Determine **safe bounds** on execution time of each fragment using **abstractions**.
3. Determine **constraints on control flow** (e.g. loop bounds) through program by **abstractions**.
4. **Conquer:** combine 2 + 3 into bound of execution time of the whole program.
Structure of WCET Analyzers

Input Executable

- CFG Reconstruction
- Value Analysis
- Control Flow Analysis
- Micro-architectural Analysis
- Global Bound Analysis

-> WCET Bound

Reconstructs a control flow graph from the binary.
Determines invariants for the values in registers and in memory.
Determines constraints on the control flow, by determining loop bounds, and identifying infeasible paths.
Determines bounds on execution times of basic blocks.
Based on an abstract model of the microarchitecture, including detailed models of the pipeline, and the memory hierarchy.
Determines a worst-case path and an upper bound on the execution time.
Usually formulated as integer linear program.
Structure of WCET Analyzers

- Input Executable
  - CFG Reconstruction
    - Reconstructs a control-flow graph from the binary.
    - Value Analysis
      - Control Flow Analysis
      - Micro-architectural Analysis
    - Global Bound Analysis
      - WCET Bound
Structure of WCET Analyzers

- **Input Executable**: Reconstructs a control-flow graph from the binary.
- **CFG Reconstruction**: Determines invariants for the values in registers and in memory.
- **Value Analysis**: Determines constraints on the control flow, by determining loop bounds, and identifying infeasible paths.
- **Control Flow Analysis**: Determines bounds on execution times of basic blocks.
- **Micro-architectural Analysis**: Based on an abstract model of the microarchitecture, including detailed models of the pipeline, and the memory hierarchy.
- **Global Bound Analysis**: Determines a worst-case path and an upper bound on the execution time.
- **WCET Bound**: Usually formulated as integer linear program.
Structure of WCET Analyzers

- Reconstructs a control-flow graph from the binary.
- Determines invariants for the values in registers and in memory.
- Determines invariants on the control flow, by
  - Determining loop bounds,
  - Identifying infeasible paths.
Structure of WCET Analyzers

- **Input Executable**: Reconstructs a control-flow graph from the binary.
- **CFG Reconstruction**: Determines invariants for the values in registers and in memory.
- **Value Analysis**: Determines invariants on the control flow, by:
  - Determining loop bounds,
  - Identifying infeasible paths.
- **Control Flow Analysis**
- **Micro-architectural Analysis**: Determines bounds on execution times of program fragments.
- **Global Bound Analysis**
- **WCET Bound**

\[\text{Jan Reineke} \quad \text{Timing Analysis and Timing Predictability} \quad 2. \text{April 2012} \quad 7 / 54\]
Structure of WCET Analyzers

- **Input Executable**
  - Reconstructs a control-flow graph from the binary.

- **CFG Reconstruction**
  - Determines invariants for the values in registers and in memory.

- **Value Analysis**
  - Determines invariants on the control flow, by:
    - Determining loop bounds,
    - Identifying infeasible paths.

- **Control Flow Analysis**
  - Determines bounds on execution times of program fragments.

- **Micro-architectural Analysis**
  - Determines a worst-case path and an upper bound on the WCET.

- **Global Bound Analysis**
  - WCET Bound
Structure of WCET Analyzers

Employed Techniques

- **Input Executable**
- **CFG Reconstruction**
- **Value Analysis**
- **Control Flow Analysis**
- **Micro-architectural Analysis**
- **Global Bound Analysis**
- **WCET Bound**

**Abstract Interpretation of the Program**

**Abstract Interpretation of Program + Hardware Model**

**Integer Linear Programming**
int main(int x, int[] a) {
    int x = x % 5;
    int y = 42;
    while (x < y) {
        if (a[x] < a[x+1])
            x++
        else
            x += 2;
    }
    return x;
}
Structure of WCET Analyzers

- Reconstructs a control-flow graph from the binary.
- Determines invariants for the values in registers and in memory.
- Determines invariants on the control flow, by
  - Determining loop bounds,
  - Identifying infeasible paths.
- Determines bounds on execution times of program fragments.
- Determines a worst-case path and an upper bound on the WCET.

Timing Analysis Framework

Input Executable

CFG Reconstruction

Value Analysis

Control Flow Analysis

Micro-architectural Analysis

Global Bound Analysis

WCET Bound
Value Analysis

Determines invariants on values of registers at different program points. Invariants are often in the form of enclosing intervals of all possible values.

Where is this information used?

- Microarchitectural Analysis
  - Pipeline Analysis
  - Cache Analysis
- Control-Flow Analysis
  - Detect infeasible paths
  - Derive loop bounds

```
R1 = R1 % 5
R2 = 42
R1 = R1 + 1
R3 = MEM[a+R1]
R4 = MEM[a+R1+4]
R3 < R4?
return R1
R1 < R2?
R1 = R1 + 2
R1 = R1 + 1
```
Value Analysis
Intuition of Interval Analysis

\[ R1 = \mathbb{R} \]
\[ R2 = \mathbb{R} \]

1. \[ R1 = R1 \% 5 \]
2. \[ R2 = 42 \]
3. \[ R1 < R2 ? \]
4. \[ R3 = \text{MEM}[a+R1] \]
5. \[ R4 = \text{MEM}[a+R1+4] \]
6. \[ R3 < R4? \]
7. \[ R1 = R1 + 2 \]
8. \[ R1 = R1 + 1 \]
9. return \( R1 \)
Value Analysis
Intuition of Interval Analysis

\[ R1 = \{-\infty, +\infty\} \]
\[ R2 = \{-\infty, +\infty\} \]

\[ R1 = [0, 4] \]
\[ R2 = [42, 42] \]

\[ R1 = R1 \mod 5 \]
\[ R2 = 42 \]

\[ R1 < R2 ? \]

\[ R3 = \text{MEM}[a+R1] \]
\[ R4 = \text{MEM}[a+R1+4] \]
\[ R3 < R4? \]

\[ R1 = R1 + 2 \]
\[ R1 = R1 + 1 \]

return R1
Value Analysis
Intuition of Interval Analysis

R1 = [-infty, +infty]
R2 = [-infty, +infty]

R1 = [0, 4]
R2 = [42, 42]

R1 = [0, 4]
R2 = [42, 42]

R1 = R1 % 5
R2 = 42

R1 < R2 ?

R3 = MEM[a+R1]
R4 = MEM[a+R1+4]
R3 < R4?

R1 = R1 + 2
R1 = R1 + 1

return R1
Value Analysis
Intuition of Interval Analysis

\[ R1 = \{-\infty, +\infty\} \]
\[ R2 = \{-\infty, +\infty\} \]

\[ R1 = [0, 4] \]
\[ R2 = [42, 42] \]

\[ R1 = [0, 4] \]
\[ R2 = [42, 42] \]

\[ R1 = [0, 4] \]
\[ R2 = [42, 42] \]

\[ R1 = [0, 4] \]
\[ R2 = [42, 42] \]

\[ R1 = [2, 6] \]
\[ R2 = [42, 42] \]
Value Analysis
Intuition of Interval Analysis

\[
R1 = [-\infty, +\infty] \\
R2 = [-\infty, +\infty]
\]

\[
R1 = [0, 6] \\
R2 = [42, 42]
\]

\[
R1 = [0, 4] \\
R2 = [42, 42]
\]
Value Analysis
Intuition of Interval Analysis

R1 = R1 % 5
R2 = 42

R1 = R1 + 1
R3 = MEM[a+R1]
R4 = MEM[a+R1+4]
R3 < R4?

R1 < R2?
R1 = R1 + 2
R1 = R1 + 1

return R1

R1 = R1 % 5
R2 = 42

R1 = [0, 6]
R2 = [42, 42]

R1 = [0, 6]
R2 = [42, 42]

R1 = [0, 6]
R2 = [42, 42]

R1 = [0, 4]
R2 = [42, 42]

R1 = [0, 6]
R2 = [42, 42]

R1 = [0, 6]
R2 = [42, 42]

R1 = [0, 4]
R2 = [42, 42]

R1 = [0, 6]
R2 = [42, 42]

R1 = [0, 6]
R2 = [42, 42]

R1 = [1, 5]
R2 = [42, 42]

R1 = [2, 6]
R2 = [42, 42]
Value Analysis
Intuition of Interval Analysis

R1 = [-\infty, +\infty]
R2 = [-\infty, +\infty]

R1 = [0, 43]
R2 = [42, 42]

R1 = [0, 41]
R2 = [42, 42]

R1 = R1 % 5
R2 = 42

R1 < R2 ?

R3 = MEM[a+R1]
R4 = MEM[a+R1+4]
R3 < R4?

R1 = R1 + 2
R1 = R1 + 1

R1 = [-\infty, +\infty]
R2 = [-\infty, +\infty]

R1 = [0, 4]
R2 = [42, 42]

R1 = [0, 6]
R2 = [42, 42]

R1 = [2, 6]
R2 = [42, 42]

R1 = [1, 42]
R2 = [42, 42]

R1 = [0, 4]
R2 = [42, 42]

R1 = [0, 6]
R2 = [42, 42]

R1 = [0, 41]
R2 = [42, 42]

R1 = [0, 43]
R2 = [42, 42]

R1 = [0, 6]
R2 = [42, 42]

R1 = [1, 42]
R2 = [42, 42]

R1 = [2, 43]
R2 = [42, 42]

R1 = [1, 42]
R2 = [42, 42]

return R1

R1 = [1, 42]
R2 = [42, 42]

R1 = [2, 43]
R2 = [42, 42]
Value Analysis
Intuition of Interval Analysis

R1 = [-\infty, +\infty]
R2 = [-\infty, +\infty]

R1 = [0, 43]
R2 = [42, 42]

R1 = [0, 41]
R2 = [42, 42]

R1 = [42, 43]
R2 = [42, 42]

R1 = [42, 43]
R2 = [42, 42]

R1 = [1, 42]
R2 = [42, 42]

R1 = [2, 43]
R2 = [42, 42]
Value Analysis
Intuition of Interval Analysis

Can be formalized as Abstract Interpretation. ➔ Yields soundness and termination guarantees.
Control-Flow Analysis

R1 increases by at least 1 in every iteration.

R1 = [0, 41]
R2 = [42, 42]

R1 = R1 % 5
R2 = 42

R1 < R2?

R3 = MEM[a+R1]
R4 = MEM[a+R1+4]
R3 < R4?

R1 = R1 + 2
R1 = R1 + 1

return R1
Control-Flow Analysis

R1 = R1 % 5
R2 = 42
R1 = R1 + 1
R3 = MEM[a+R1]
R4 = MEM[a+R1+4]
R3 < R4?
return R1
R1 < R2?
R1 = R1 + 2
R1 = R1 + 1
R1 increases by at least 1 in every iteration
Can enter loop at most 42 times

R1 = [0, 41]
R2 = [42, 42]
Control-Flow Analysis

\[ R1 = R1 \mod 5 \]
\[ R2 = 42 \]
\[ R1 = R1 + 1 \]
\[ R3 = \text{MEM}[a+R1] \]
\[ R4 = \text{MEM}[a+R1+4] \]
\[ R3 < R4? \]
\[ \text{return } R1 \]

\[ R1 < R2? \]
\[ R1 = R1 + 2 \]
\[ R1 = R1 + 1 \]

\[ R1 = [0, 41] \]
\[ R2 = [42, 42] \]

*R1 increases by at least 1 in every iteration*

\[ \rightarrow \text{Can enter loop at most 42 times} \]

Can we also come up with a lower bound?
Structure of WCET Analyzers

- **Input Executable**
  - Reconstructs a control-flow graph from the binary.
  - Determines invariants for the values in registers and in memory.

- **Value Analysis**
  - Determines invariants on the control flow, by:
    - Determining loop bounds,
    - Identifying infeasible paths.

- **Control Flow Analysis**
  - Determines bounds on execution times of program fragments.

- **Global Bound Analysis**
  - Determines a worst-case path and an upper bound on the WCET.

- **Micro-architectural Analysis**
  - Determines bound on execution times of program fragments.

- **WCET Bound**
  - Usually formulated as integer linear program.
Microarchitectural Analysis

Ideal 1970s world: one instruction = one cycle

Real world:
- Pipelining
- Branch prediction + speculative execution
- Caches
- DRAM

→ Execution time of individual instruction highly variable and dependent on state of microarchitecture

→ Need to determine in which states the microarchitecture may be at a point in the program
Pipelining

- Instruction execution is split into several **stages**
- Several instructions can be executed in parallel
- Some pipelines can start more than one instruction per cycle: VLIW, Superscalar
- Some processors can execute instructions out-of-order
- Practical Problems: Hazards and cache misses
Hardware Features: Pipelines

Ideal Case: One Instruction per Cycle
Pipeline Hazards:

- **Data Hazards**: Operands not yet available (Data Dependences)
- **Resource Hazards**: Consecutive instructions use same resource
- **Control Hazards**: Conditional branch
- **Instruction-Cache Hazards**: Instruction fetch causes cache miss

Assuming worst case everywhere is not an option!
Static exclusion of hazards
Static exclusion of hazards

Cache analysis: prediction of cache hits on instruction or operand fetch or store
Static exclusion of hazards

Cache analysis: prediction of cache hits on instruction or operand fetch or store

`lwz r4, 20(r1)`
Cache analysis: prediction of cache hits on instruction or operand fetch or store

`lwz r4, 20(r1)`
Static exclusion of hazards

Cache analysis: prediction of cache hits on instruction or operand fetch or store

\texttt{lwz r4, 20(r1)} \quad \text{Hit}

Dependence analysis: elimination of data hazards
Static exclusion of hazards

Cache analysis: prediction of cache hits on instruction or operand fetch or store

```
lwz r4, 20(r1)
```

Hit

Dependence analysis: elimination of data hazards

```
add r4, r5,r6
lwz r7, 10(r1)
add r8, r4, r4
```
Static exclusion of hazards

Cache analysis: prediction of cache hits on instruction or operand fetch or store

lwz r4, 20(r1)
Hit

Dependence analysis: elimination of data hazards

add r4, r5, r6
lwz r7, 10(r1)
add r8, r4, r4
Operand ready
Static exclusion of hazards

Cache analysis: prediction of cache hits on instruction or operand fetch or store

```
lwz r4, 20(r1)
```

Hit

Dependence analysis: elimination of data hazards

```
add r4, r5, r6
lwz r7, 10(r1)
add r8, r4, r4
```

Operand ready

Resource reservation tables: elimination of resource hazards
Static exclusion of hazards

Cache analysis: prediction of cache hits on instruction or operand fetch or store

\textit{lwz r4, 20(r1)}

Dependence analysis: elimination of data hazards

\textit{add r4, r5, r6}
\textit{lwz r7, 10(r1)}
\textit{add r8, r4, r4}

Resource reservation tables: elimination of resource hazards

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
\textbf{IF} & & & & & \\
\hline
\textbf{EX} & & & & & \\
\hline
\textbf{M} & & & & & \\
\hline
\textbf{F} & & & & & \\
\hline
\end{tabular}
\end{center}
View of Processor as a State Machine

- Processor (pipeline, cache, memory, inputs) viewed as a *big state machine*, performing transitions every *clock cycle*
- Starting in an *initial state* for an instruction, transitions are performed, until a *final state* is reached:
  - End state: instruction has left the pipeline
  - # transitions: *execution time* of instruction
A Concrete Pipeline Executing a Basic Block

```plaintext
function exec (b : basic block, s : concrete pipeline state)  
t : trace
interprets instruction stream of b starting in state s producing trace t.

Successor basic block is interpreted starting in initial state last(t)

length(t) gives number of cycles for basic block b
```
function exec (b : basic block, s : abstract pipeline state) t : trace
interprets instruction stream of b (annotated with cache information) starting in state s producing abstract trace t
length(t) gives number of cycles
What is different?

- Abstract states may lack information, e.g. about cache contents.
- More than one trace may be possible.
- Starting state for successor basic block?
  In particular, if there are several predecessor blocks.
What is different?

- Abstract states may lack information, e.g., about cache contents.
- More than one trace may be possible.
- Starting state for successor basic block? In particular, if there are several predecessor blocks.

Alternatives:
- sets of states
- combine by least upper bound (join), hard to find one that
  - preserves information and
  - has a compact representation.
Nondeterminism

- In the concrete pipeline model, one state resulted in one new state after a one-cycle transition
- Now, in the abstract model, one state can have several successor states
  - Transitions from set of states to set of states
Non-Locality of Local Contributions

- Interference between processor components produces **Timing Anomalies**:
  - Assuming local best case leads to higher overall execution time.
  - Assuming local worst case leads to shorter overall execution time
    Ex.: Cache miss in the context of branch prediction

- Treating components in isolation may be unsafe

- Implicit assumptions are not always correct:
  - Cache miss is not always the worst case!
  - The empty cache is not always the worst-case start!
An Abstract Pipeline Executing a Basic Block

**function** `analyze` *(b : basic block, S : analysis state)*  
**T**: set of trace

Analysis states = $2^{PS \times CS}$

$PS$ = set of abstract pipeline states
$CS$ = set of abstract cache states

interprets instruction stream of $b$ (annotated with cache information) starting in state $S$ producing set of traces $T$

$max(length(T))$ - upper bound for execution time

$last(T)$ - set of initial states for successor block

Union for blocks with several predecessors.
function analyze (b : basic block, S : analysis state) T: set of trace

Analysis states = 2^{PS \times CS}

PS = set of abstract pipeline states

CS = set of abstract cache states

interprets instruction stream of b (annotated with cache information) starting in state S producing set of traces T

max(length(T)) - upper bound for execution time

last(T) - set of initial states for successor block

Union for blocks with several predecessors.
Integrated Analysis: Overall Picture

Fixed point iteration over Basic Blocks in abstract state \{s_1, s_2, s_3\}

Cyclewise evolution of processor model for instruction
Structure of WCET Analyzers

- Reconstructs a control-flow graph from the binary.
- Determines invariants for the values in registers and in memory.
- Determines invariants on the control flow, by:
  - Determining loop bounds,
  - Identifying infeasible paths.
- Determines bounds on execution times of program fragments.
- Determines a worst-case path and an upper bound on the WCET.
Global Bound Analysis
aka Path Analysis aka Implicit Path Enumeration

- Determines a worst-case path and an upper bound on the WCET.
- Formulated as integer linear program (ILP).

```
x_b = frequency of executing b

R1 = R1 % 5
R2 = 42
R1 < R2 ?
R3 = MEM[a+R1]
R4 = MEM[a+R1+4]
R3 < R4?
R1 = R1 + 2
R1 = R1 + 1
R1 < R2 ?
R1 = R1 + 2
R1 = R1 + 1
R1 < R2 ?
return R1
```

Constraints:
- Loop bounds
- Structural constraints, due to CFG

Mathematical formulation:

```
max c_a x_a + c_b x_b + c_c x_c + c_d x_d + c_e x_e + c_f x_f
s.t.  x_b = x_a + x_d + x_e
      x_c = x_d + x_e
      x_a = x_f = 1
      x_a >= 0, x_b >= 0, ...
      lb <= x_b <= ub
```
Global Bound Analysis
aka Path Analysis aka Implicit Path Enumeration

- Determines a worst-case path and an upper bound on the WCET.
- Formulated as integer linear program (ILP).

\[
\begin{align*}
\text{Max} & \quad 2x_a + 3x_b + 6x_c + 3x_d + 2x_e + 2x_f \\
\text{s.t.} & \quad x_b = x_a + x_d + x_e, \\
& \quad x_c = x_d + x_e, \\
& \quad x_a = x_f = 1, \\
& \quad x_a \geq 0, x_b \geq 0, \ldots \\
& \quad 19 \leq x_b \leq 42
\end{align*}
\]

\(x_b\) = frequency of executing \(b\)
\(c_b\) = time to execute \(b\) once

Structural constraints, due to CFG
Loop bounds
Integer linear programming

Linear programming (LP)

\[
\begin{align*}
\text{maximize} & \quad c^T x & \text{Objective function} \\
\text{subject to} & \quad Ax \leq b & \text{Linear constraints} \\
\text{and} & \quad x \geq 0
\end{align*}
\]

... + Restriction to integers = ILP.

LP is in polynomial time, yet, ILP is NP hard, but often efficiently solvable in practice.

Solvers (e.g. CPLEX) determine the maximal value of the objective function + corresponding valuation of variables.
Global Bound Analysis
aka Path Analysis aka Implicit Path Enumeration

Max \(2x_a + 3x_b + 6x_c + 3x_d + 2x_e + 2x_f\)

s.t. \(x_b = x_a + x_d + x_e,\)
     \(x_c = x_d + x_e,\)
     \(x_a = x_f = 1,\)
     \(x_a \geq 0, x_b \geq 0, \ldots\)
     \(19 \leq x_c \leq 42\)

Objective function = \(2*1 + 3*43 + (6+3)*42 + 2*1 = 511\)
Summary and Outlook

- Divide and conquer:
  - Analyze worst-case timing of program fragments separately
  - Combine results using integer linear program

- Abstraction:
  - Employ sound abstractions to solve undecidable problems approximately

Next week:
theoretical background of Abstract Interpretation