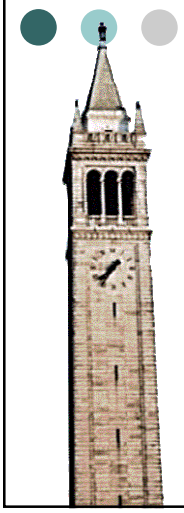


Introduction to Embedded Systems

Edward A. Lee & Sanjit A. Seshia
UC Berkeley
EECS 124
Spring 2008

Lecture 19: Execution Time Analysis

Source

Material in this lecture is drawn from the following sources:

- "The Worst-Case Execution Time Problem – Overview of Methods and Survey of Tools", R. Wilhelm et al., ACM Transactions on Embedded Computing Systems, 2007.
- Chapter 9 of "Computer Systems: A Programmer's Perspective", R. E. Bryant and D. R. O'Hallaron, Prentice-Hall, 2002.
- "Performance Analysis of Real-Time Embedded Software," Y-T. Li and S. Malik, Kluwer Academic Pub., 1999.

Worst-Case Execution Time (WCET) of a Task

The longest time taken by a software task to execute
→ Function of input data and environment conditions

BCET = Best-Case Execution Time
(shortest time taken by the task to execute)

Worst-Case Execution Time (WCET) & BCET

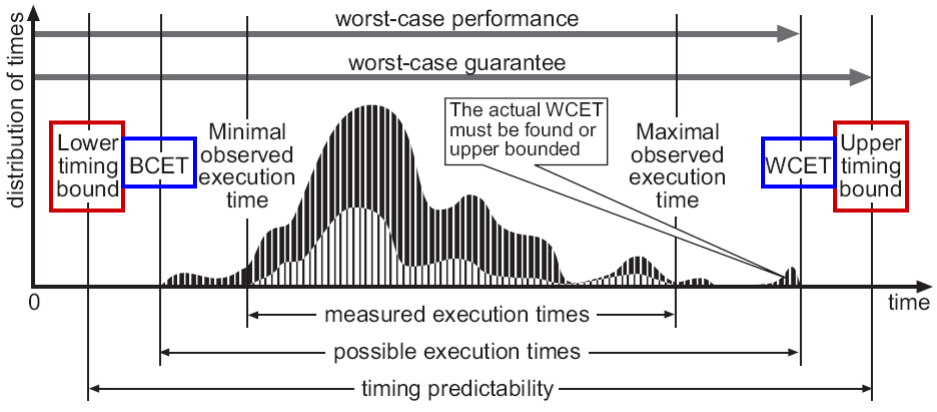


Figure from R.Wilhelm et al., ACM Trans. Embed. Comput. Sys, 2007.

The WCET Problem

Given

- the code for a software task
- the platform (OS + hardware) that it will run on

Determine the WCET of the task.

Why is this problem important?

The WCET is central in the design of RT Systems:
Needed for Correctness (does the task finish in time?) and
Scheduling (find optimal schedule for tasks, last Wed's lecture)

Can the WCET always be found?

In general, no, because the problem is *undecidable*.

Typical WCET Problem

Task executes within an infinite loop

```
while(1) {
    read_sensors();
    compute();
    write_to_actuators();
}
```

This code typically has:

- loops with finite bounds
- no recursion

Additional assumptions:

- runs uninterrupted
- single-threaded

Components of Execution Time Analysis

- Program path (Control flow) analysis
  - Want to find longest path through the program
  - Identify feasible paths through the program
  - Find loop bounds
  - Identify dependencies amongst different code fragments
- Processor behavior analysis
  - For small code fragments (basic blocks), generate bounds on run-times on the platform
  - Model details of architecture, including cache behavior, pipeline stalls, branch prediction, etc.
- Outputs of both analyses feed into each other

Program Path Analysis: Path Explosion

```
for (Outer = 0; Outer < MAXSIZE; Outer++) {
/* MAXSIZE = 100 */
      for (Inner = 0; Inner < MAXSIZE; Inner++) {
            if (Array[Outer][Inner] >= 0) {
                  Ptotal += Array[Outer][Inner];
                  Pcnt++;
            } else {
                  Ntotal += Array[Outer][Inner];
                  Ncnt++;
            }
      Postotal = Ptotal;
      Poscnt = Pcnt;
      Negtotal = Ntotal;
      Negcnt = Ncnt;
}
```

Example cnt.c from WCET benchmarks, Malardalen Univ.

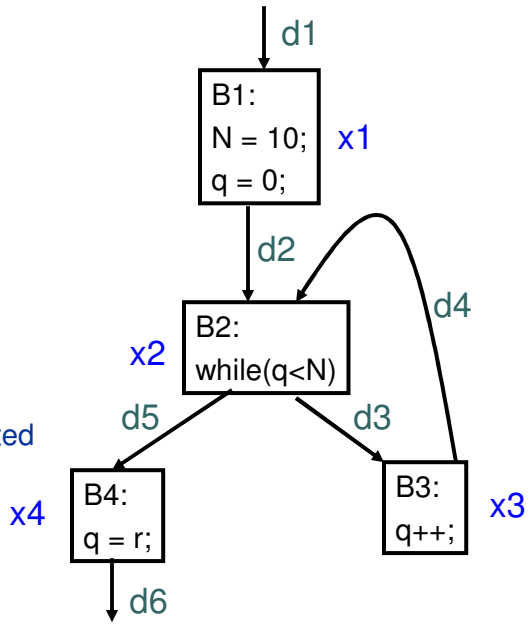
Program Path Analysis: Overall Approach

- Construct Control-Flow Graph (CFG) for the task
  - Nodes represent Basic Blocks of the task
  - Edges represent flow of control (jumps, branches, calls, …)
- The problem is to identify the longest path in the CFG
  - Note: CFG can have loops, so need to infer loop bounds and unroll them
  - This gives us a directed acyclic graph (DAG). How do we find the longest path in this DAG?

Example

```
N = 10;
q = 0;
while(q < N)
  q++;
q = r;
```

xi → # times Bi is executed
dj → # times edge is executed



Example due to Y.T. Li and S. Malik

Program Path Analysis: Dependencies

```
void altitude_pid_run(void) {
  float err = estimator_z - desired_altitude;
  desired_climb = pre_climb + altitude_pgain * err;
  if (desired_climb < -CLIMB_MAX)
      desired_climb = -CLIMB_MAX;
  if (desired_climb > CLIMB_MAX)
      desired_climb = CLIMB_MAX;
}
```

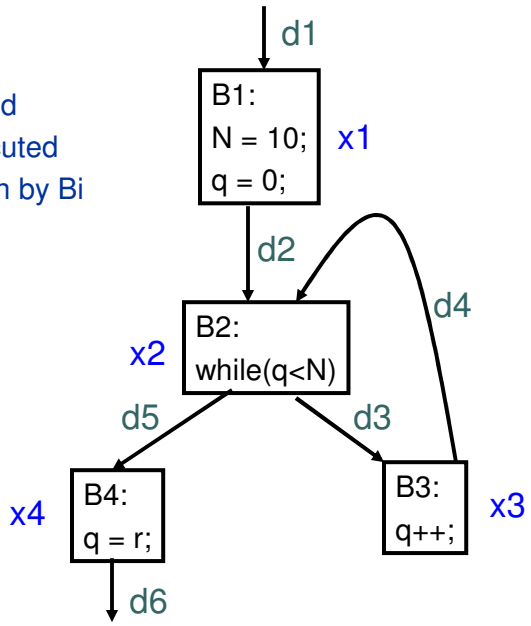
Only one of these statements is executed

Example from "PapaBench" UAV autopilot code, IRIT, France

Example, Revisited

xi → # times Bi is executed
dj → # times edge is executed
C_i → measured time taken by Bi

Want to
maximize $\sum_i C_i x_i$
subject to constraints
x1 = d1 = d2
x2 = d2+d4 = d3+d5
x3 = d3 = d4 = 10
x4 = d5 = d6



Example due to Y.T. Li and S. Malik

Timing Analysis and Compositionality

Consider a task T with two parts A and B composed in sequence: T = A; B

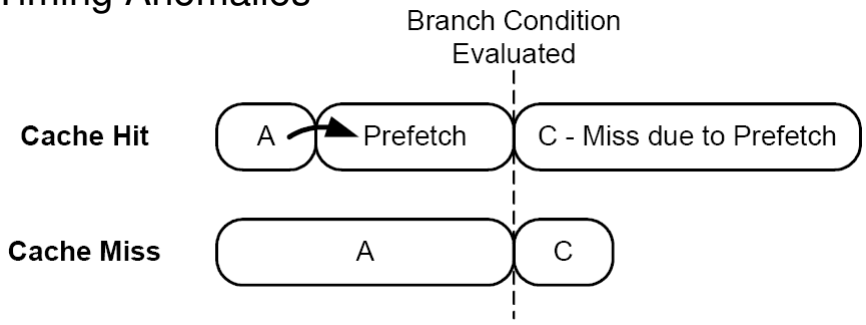
Is WCET(T) = WCET(A) + WCET(B) ?

NO!

WCETs cannot simply be composed ☹

→ Due to dependencies "through environment"

Timing Anomalies



Scenario 1: Instr A hits in I-cache, triggers a branch speculation, and prefetch of instructions, then predicted branch is wrong, so Instr C must execute, but it's been evicted from I-cache, execution of C delayed.

Scenario 2: Instr A misses in I-cache, no branch prediction, then C hits in I-cache, C completes.

[from R.Wilhelm et al., ACM Trans. Embed. Comput. Sys, 2007]

How to Measure Run-Time

Several techniques, with varying accuracy:

- Instrument code to sample CPU cycle counter
  - relatively easy to do, read processor documentation for assembly instruction
- Use cycle-accurate simulator for processor
  - useful when hardware is not available/ready
- Use Logic Analyzer
  - non-intrusive measurement, more accurate
- …

Cycle Counters

Most modern systems have built in registers that are incremented every clock cycle

Special assembly code instruction to access

On Intel 32-bit x86 machines:

- 64 bit counter
- RDTSC instruction sets `%edx` to high order 32-bits, `%eax` to low order 32-bits

Wrap-around time for 2 GHz machine

- Low order 32-bits every 2.1 seconds
- High order 64 bits every 293 years

[slide due to R. E. Bryant and D. R. O'Hallaron]

Measuring with Cycle Counter

Idea

- Get current value of cycle counter
  - store as pair of unsigned's `cyc_hi` and `cyc_lo`
- Compute something
- Get new value of cycle counter
- Perform double precision subtraction to get elapsed cycles

```
/* Keep track of most recent reading of cycle counter */
static unsigned cyc_hi = 0;
static unsigned cyc_lo = 0;

void start_counter()
{
  /* Get current value of cycle counter */
  access_counter(&cyc_hi, &cyc_lo);
}
```

[slide due to R. E. Bryant and D. R. O'Hallaron]

Accessing the Cycle Counter

- GCC allows inline assembly code with mechanism for matching registers with program variables
- Code only works on x86 machine compiling with GCC

```
void access_counter(unsigned *hi, unsigned *lo)
{
  /* Get cycle counter */
  asm("rdtsc; movl %%edx,%0; movl %%eax,%1"
      : "=r" (*hi), "=r" (*lo)
      : /* No input */
      : "%edx", "%eax");
}
```

- Emit assembly with `rdtsc` and two `movl` instructions

[slide due to R. E. Bryant and D. R. O'Hallaron]

Completing Measurement

- Get new value of cycle counter
- Perform double precision subtraction to get elapsed cycles
- Express as `double` to avoid overflow problems

```
double get_counter()
{
  unsigned ncyc_hi, ncyc_lo
  unsigned hi, lo, borrow;
  /* Get cycle counter */
  access_counter(&ncyc_hi, &ncyc_lo);
  /* Do double precision subtraction */
  lo = ncyc_lo - cyc_lo;
  borrow = lo > ncyc_lo;
  hi = ncyc_hi - cyc_hi - borrow;
  return (double) hi * (1 << 30) * 4 + lo;
}
```

[slide due to R. E. Bryant and D. R. O'Hallaron]

Timing With Cycle Counter

Time Function P

- First attempt: Simply count cycles for one execution of P

```
  double tcycles;
  start_counter();
  P();
  tcycles = get_counter();
```

- What can go wrong here?

[slide due to R. E. Bryant and D. R. O'Hallaron]

Measurement Pitfalls

- Instrumentation incurs small overhead
  - measure long enough code sequence to compensate
- Cache effects can skew measurements
  - "warm up" the cache before making measurement
- Multi-tasking effects: counter keeps going even when the task of interest is inactive
  - take multiple measurements and pick "k best" (cluster)
- Multicores/hyperthreading
  - Need to ensure that task is 'locked' to a single core
- Power management effects
  - CPU speed might change, timer could get reset during hibernation

Some WCET Estimation Tools

Tool	Flow	Proc. Behavior	Bound Calc.
aiT	value analysis	static program analysis	IPET
Bound-T	linear loop-bounds and constraints by Omega test	static program analysis	IPET per function
RapiTime	n.a.	measurement	structure-based
SymTA/P	single feasible path analysis	static program analysis for I/D cache, measurement for segments	IPET
Heptane	-	static prog. analysis	structure-based, IPET
Vienna S. Vienna M. Vienna H.	- Genetic Algorithms Model Checking	static program analysis segment measurements segment measurements	IPET n.a. IPET
SWEET	value analysis, abstract execution, syntactical analysis	static program analysis for instr. caches, simulation for the pipeline	path-based, IPET-based, clustered
Florida		static program analysis	path-based
Chalmers		modified simulation	
Chronos		static prog. analysis	IPET

[R.Wilhelm et al., ACM Trans. Embed. Comput. Sys, 2007]

Open Problems

- Architectures are getting much more complex.
  - Can we create processor behavior models without the agonizing pain?
  - Can we change the architecture to make timing analysis easier? [See PRET machine project by Prof. Lee and colleagues]
- Analysis methods are “Brittle” – small changes to code and/or architecture can require completely re-doing the WCET computation
- Need more reliable ways to measure execution time