Worst-Case Execution Time Analysis

EECE 494
The University of British Columbia
Lecture Outline

• How do we estimate the execution time of a task?
  – Specifically, WCET.
  – Testing.
  – Simulation.
  – Program analysis.

CPUs and Software Performance Analysis

• System performance cannot be determined without choosing a CPU
• Software execution times definitely don’t scale across CPU architectures, perhaps not even within a CPU family
• Architectural features which influence program performance:
  – pipelining
  – caching
  – bus bandwidth
Hierarchical Performance Modeling

• We would like to have a hierarchy of increasingly accurate performance models, from system specification to assembly code.

• Very little work in C-level performance modeling---hard to separate the program from the CPU.

• Two types of questions:
  – which variety of Brand X CPU do I use?
  – should I use Brand X or Brand Y?
Caches and Code Speed

• Worst-case:
  – tight-deadline device interrupts
  – driver is not in cache
  – multiple high-priority drivers knock each other out of the cache

• Cache miss costs from a few cycles on up; the faster the CPU, the more costly is a miss

• Worst-case execution time is much larger than best-case, leading to extreme overengineering.
Alternative Approaches to Performance Analysis

• Conservative analysis – performance always within bounds.
  – WCET gives bounds from analysis, limited simulation.

• Detailed analysis – more info on a particular case but no bounds.
  – Execution-based methods provide lots of details but only for given input data.
Performance of HLLs

• HLL: High-Level Language
• We would like to bound or estimate program performance from high-level code:
  – simplifies identification of paths
  – provides early performance estimates
• Realistically, we need to know the execution platform.
Paths and Performance

• Branches of conditional have different execution times.
• Loops:
  – Multiple iterations.
  – Varying number of iterations.
Measurements of Interest

• Execution time bounds: worst, best.
  – Upper/lower bounds important in multitasking systems.

• Execution time of incomplete code.
  – Must be able to handle time estimates for pieces of code.

• Bounds of varying quality:
  – Loose bounds quickly.
  – Tight bounds with more effort.
Early Work: Explicit Path Analysis

• Shaw developed techniques to prove bounds on the number of times through paths.
• Park and Shaw developed techniques for path analysis and for measurement of execution times of HLL statements on a 68000.
Challenges and Approaches

• Exponential number of paths.
  – Limit program constructs.
  – Add annotations.
  – Implicit path analysis.

• Instruction times are not independent: pipeline effects; cache effects.
  – State-dependent instruction execution time.
  – Simulation.
Abstract Program Flow Analysis

• Bound set of feasible paths without exhaustive simulation.
  – May include some infeasible paths in the feasible set.

• Perform abstract interpretation of the program to find feasible paths.
  – Generate safe bounds on the values of variables.
Bounding Loop Iterations

Four phases:

1. Iteratively identify branches that affect the number of iterations.
2. Identify loop iteration on which loop index-dependent branches change direction.
3. Use step 2 to determine when step 1 branches are reached.
4. Calculate bounds on number of iterations.
Implicit Path Analysis

• Schedl, Li/Malik—find path length without explicitly finding path.

• Formulate as constraint solving problem:
  – Generate constraints that describe program, annotations.
  – Solve using constraint solver, ILP (depending on types of constraints allowed).

• IPET (Implicit Path Enumeration).
WCET and Optimizing Compilers

• Optimizing compilers can radically change program control flow.

• Must analyze timing of the optimized code.
  – Annotations must be transferred to the optimized code.
  – Must be able to perform the program transformations on the optimizations.
Cache Analysis Extensions

• Must segment program units around cache lines.
• Different execution times for in-cache and out-of-cache.
• Conservative assumption: use in-cache time only if statement is known to be in cache.
• Add constraints which model cache state based on program flow.
Cache Analysis Model (Colin & Puaut)

• **Instruction block (iblock):** a basic block fragment that fits into a cache line.
  – Decomposition of program into iblocks depends on cache organization.

• Determine paths on which iblocks result in hits, misses.
Cache Interference Example

for (i=0; i<N; i++) {
    f1();
    f2();
    f3();
}
Branch Prediction Bounding (Colin & Puaut)

• Missed branch prediction causes pipeline bubble.
  – Branch predictor has finite capacity.
  – Predictor may make wrong prediction.

• Keep track of branch history.
  – Memoryless predictors are a special case.

• Determine what prediction the machine will make to determine whether a bubble may be caused.
  – Known correct predictions cause no bubble.
  – Known incorrect predictions cause a bubble.
  – Indeterminate results are pessimistically presumed to cause a bubble.
Data Caching

• Data address may not be known at compile time:
  – Pointers.
  – Stack variables.

• Caching of stack variables is easier to compute.
  – Offset from stack pointer is known.
  – Can compute cache block based upon sp offset.
Timing Through Simulation

• Use a simulator to time a sequence of instructions.
  – Can simulate basic blocks with boundary conditions for branches.

• WCET tools often use custom simulators designed for small pieces of code, call by subroutine.
Behavioral Performance Analysis

• Use program behavior to analyze performance.
• Advantages:
  – Handles arbitrary program.
  – Captures realistic behavior.
• Disadvantages:
  – Doesn’t guarantee worst-case/best-case behavior.
Methodology

• Sources of a behavior:
  – Program execution on platform.
  – Simulated execution.
Cycle-Accurate Simulator

• Models the microarchitecture.
  – Simulating one instruction requires executing routines for instruction decode, etc.

• Models pipeline state.
  – Microarchitectural registers are exposed to the simulator.
Sources of Timing Information

• Data book tables:
  – Time of individual instructions.
  – Penalties for various hazards.

• Microarchitecture:
  – Depends from the structure of machine.
  – Derived from execution of the instruction in the microarchitecture.
Levels of Detail in Simulation

• Instruction schedulers:
  – Models availability of microarchitectural resources.
  – May not capture all interactions.

• Cycle timers:
  – Models full microarchitecture.
  – Most accurate, requires exact model of the microarchitecture.
Modular Simulators

• Model instructions through a description file.
  – Drives assembler, basic behavioral simulation.

• Assemble a simulation program from code modules.
  – Can add your own code.
What To Learn

• The need for timing analysis
• Principles of execution time analysis
  – By measurements and simulations
  – Static analysis
    • What is flow analysis?
    • What is low-level analysis?
    • What is IPET?

The required reading supplements this lecture (and is quite an easy read).