CS/COE1541: Introduction to Computer Architecture

Architecture Simulation Basics

Sangyeun Cho

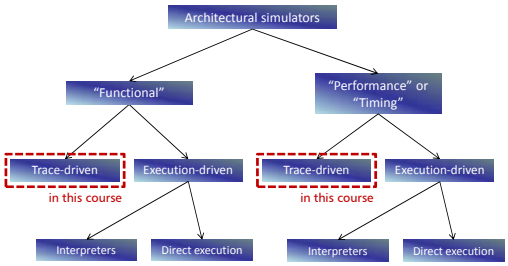
Dept. of Computer Science
University of Pittsburgh

Architecture simulator

- What is an architecture (or architectural) simulator?
  - A tool that reproduces the behavior of a computing device

- Why use a simulator?
  - Leverage faster, more flexible software development cycle
    - Permits more design space exploration
    - Facilitates validation before hardware becomes available
    - Level of abstraction can be throttled to design task
    - Possible to increase/improve system instrumentation

A taxonomy of simulation tools



Functional vs. timing simulators

- Functional simulators implement the architecture
  - The architecture is what programmers see
  - One can collect basic program execution information
    - How many instructions in different classes? (or instruction mix)
    - What is cache hit/miss rate?
- Timing simulators implement the microarchitecture
  - Model system internals (microarchitecture)
    - Pipeline, cache, branch predictor, …
  - One can collect many timing-related statistics

Execution- vs. trace-driven sim.

- Trace-driven simulation
  - Simulator reads a "trace" of instructions captured during a previous execution
  - Both functional and timing simulator can be built

- Execution-driven simulation
  - Simulator "runs" the program, generating a trace on-the-fly
  - More difficult to implement, but has many advantages
  - Direct-execution: instrumented program runs on host

Microarchitecture simulation

- Arbitrary level of details and precision
  - Pipelines, branch predictors, caches, main memory, …
  - Novel microarchitectural ideas
  - Hardware devices (e.g., hard drive, timer, …)
- Full observability
- Challenges
  - Modeling time
  - Validation
  - Simulation time (speed)
    - If you have an idea to check, you may want to perform "proof-of-concept" experiments before undertaking full microarchitecture modeling for simulations
      - E.g., collect necessary trace from program execution and process trace data

Full system simulator?

- Widely used microarchitecture simulators, such as *SimpleScalar*, use user-code-only simulation
  - They do not simulate OS codes
  - Some of them have a proxy call mechanism (to get help from the host OS on a simulated program's system calls)
- Full system simulator provides a view of a more realistic machine so that a native OS can be booted
  - OS codes are actually simulated
- Accuracy?

Function vs. timing simulation

- Functional simulation
  - Captures and simulates basic instruction semantics
  - Updates processor states (register, memory, …)
  - Generates correct program output
- Timing simulation
  - (Typically) performs functional simulation
  - Implements various microarchitectural structures
  - Captures the timing of events to obtain program execution time
- Functional simulation takes less time

Simulation time example

- Using *gcc* (in spec2k) with a small input as our benchmark

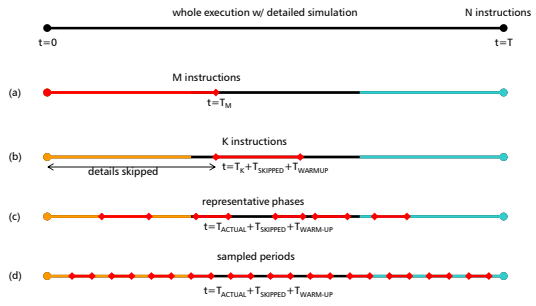
Case	Time (sec)	Ratio to "native"	Ratio to "functional"
Native	1.054	1	
sim-fast	167	158	1
sim-outorder	4,247	4,029	25
simics (bare)	461	437	1
simics w/ ruby	41,245	39,131	89
simics w/ ruby+opal	155,621	147,648	338

Measured by Lei Jin on *antimony* (3.8GHz Xeon w/ 8GB memory)

Collecting simulation statistics

- The reason for performing timing simulation is to collect timing-related or timing-sensitive information
  - E.g., IPC, average # of cycles for memory accesses, on-chip network contention, …
- During what period of program execution do you want to collect statistics from?
  - Initialization phase
  - Main phases
  - Wrap-up phase

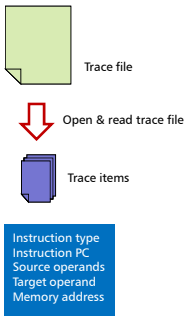
Collecting simulation statistics



In this semester

- We will build a trace-driven simulator
- We will start from a very simple functional simulator
  - No need to model ALU, shifter, multiplier, …
  - Derive instruction type, instruction address, operands information directly from the trace file
- We will build on the simple functional simulator and develop a timing simulator modeling
  - MIPS 5-stage pipeline
  - Cache memory and main memory
  - Branch predictor

In this semester



- We will provide trace files
- You will open and read from a trace file
- You will extract trace items from the file, which correspond to MIPS instructions
- You will be able to determine what the instruction is, its operands, and its address

For your further investigation

- SimpleScalar tool (www.simplescalar.com)
  - A few slides from the SimpleScalar Hacker's Guide were adopted in this material
- Virtutech Simics full-system simulator (www.simics.net)
- David Lilja's book: "Measuring Computer Performance: A Practitioner's Guide" from Cambridge Univ. Press.