

CSE 560 Computer Systems Architecture

Performance Modeling

1

1

Computer Architecture Simulator Primer

Q: What is an architectural simulator?

A: tool that reproduces the behavior of a computing device



Why use a simulator?

- leverage faster, more flexible S/W development cycle
- permits more design space exploration
- facilitates validation before H/W becomes available
- level of abstraction can be throttled to design task
- *can tell us quite a bit about performance*

2

2

Functional vs. Behavioral Simulators

Functional Simulators

- Implement instruction set architecture (what programmers see)
 - **Execute** each instruction
 - Takes real inputs, creates real outputs

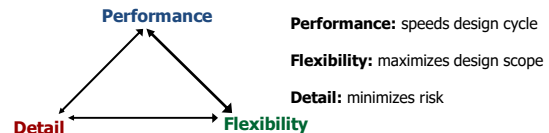
Behavioral simulators (also called *Performance Simulators*)

- Implement the microarchitecture (system internals)
 - 5 stage pipeline
 - Branch prediction
 - Caches
- Go through the internal motions to estimate time (usually)
- Might not actually execute the program

3

3

The Zen of Simulator Design



- Design goals will drive which aspects are optimized
- Previous versions of this class have used:
 - SimpleScalar: optimizes performance and flexibility
 - VHDL: optimizes detail
- This year we will use gem5
 - Cycle accurate chip multiprocessor
 - Used *lots* of places!

4

4

Parallel vs. Sequential – start here

Hardware is parallel, software is not.

- 5 stage pipeline vs. our simulate() method
 - Can't execute 5 stages in parallel, so... traverse the pipeline backwards
- HW table = software array
`dm_cache[index].data, dm_cache[index].tag`
- Anything more complicated? Serial approximation of parallel structure
 - Accessing all 4 ways in a set at once? Nope.
 - CAM lookup (find all entries with value X). Nope.
 - Flush entire instruction window? Nope.
- Simulator is slower b/c it's in software **and** its serial

5

5

Simulator Types

- Software Simulators
 - Processor Core Simulators
 - Cache Simulators
 - Full-system Simulators
- Hardware Simulators (VHDL, etc.)
 - You instantiate every wire
 - 3 Register read ports in SW vs. HW
 - Less flexible
 - More complex (and complete) model of real system
 - Slower to develop
 - Can use FPGAs for emulation (huge benefit for speed!)

6

6

Execution vs. Trace-Driven Simulation

What is the input to the simulator?

Trace-based Simulator (input = dynamic insns)

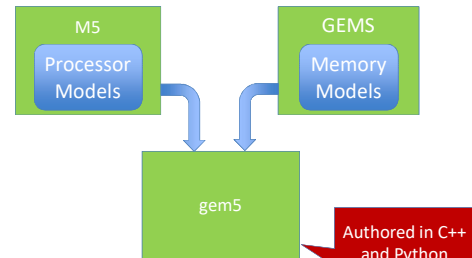
- Reads "trace" of insns captured from a previous execution
- Easiest to implement, no functional component needed

Execution-driven Simulator (input = static insns)

- 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

7

gem5 Simulator Heritage



8

Simulator Options

Configuration File:

- Configure the system being modeled (e.g., ISA, size of cache line, in order vs. out of order execution)
- Specify the binary executable to simulate
- Control the simulation (start, stop, etc.)
- Literally is a Python file
 - Anything available in Python is available here
 - Python interpreter included in simulator!

9

Simulator Output

Three output files:

- `config.ini` and `config.json`
 - Lists every `SimObject` created and its parameters
 - Indicates "what did I actually simulate?"
- Results of simulation in `stats.txt` file
 - Dump of pretty much everything collected during simulation
- Command line option:
 - `-d DIR` Specify directory for output files
 - Overwrites output files if present

10

Sample Output

```
system.cpu.apic_clk_domain.clock 16000 # Clock period in ticks
system.cpu.workload.num_syscalls 11 # Number of system calls
system.cpu.numCycles 345518 # Number of cpu cycles simulated
system.cpu.numWorkItemsStarted 0 # Number of work items this cpu start
system.cpu.numWorkItemsCompleted 0 # Number of work items this cpu compl
system.cpu.committedInits 5712 # Number of instructions committed
system.cpu.committedOps 10314 # Number of ops (including micro-ops)
system.cpu.num_int_alu_accesses 10205 # Number of integer alu accesses
system.cpu.num_fp_alu_accesses 0 # Number of float alu accesses
system.cpu.num_func_calls 221 # Number of times a function call or
system.cpu.num_conditional_control_inits 986 # Number of instructions that are con
system.cpu.num_int_inits 10205 # Number of integer instructions
system.cpu.num_fp_inits 0 # Number of float instructions
system.cpu.num_int_register_reads 19296 # Number of times the integer registe
system.cpu.num_int_register_writes 7977 # Number of times the integer registe
system.cpu.num_fp_register_reads 0 # Number of times the floating regist
system.cpu.num_fp_register_writes 0 # Number of times the floating regist
system.cpu.num_cc_register_reads 7028 # Number of times the CC registers we
system.cpu.num_cc_register_writes 3025 # Number of times the CC registers we
system.cpu.num_mmu_refs 2025 # Number of memory refs
system.cpu.num_load_inits 1004 # Number of load instructions
system.cpu.num_store_inits 941 # Number of store instructions
system.cpu.num_idle_cycles 0.001000 # Number of idle cycles
system.cpu.num_busy_cycles 345517.999000 # Number of busy cycles
system.cpu.num_idle_fraction 1.000000 # Percentage of non-idle cycles
system.cpu.idle_fraction 0.000000 # Percentage of idle cycles
system.cpu.branches 1386 # Number of branches fetched
```

11

But where is CPI?

- CPI is not one of the statistics that is provided directly in the `stats.txt` file
- What if we want to know CPI?
- Definition of CPI is average cycles/instruction
 - Simulator tells us cycles – `sim_ticks` (almost! wrong units, however; also need clock period)
 - Simulator tells us dynamic instructions – `sim_insts` (don't confuse this with micro-operations, `sim_ops`)
- Divide
- In effect, we are using perf. eqn. to solve for CPI
- Simulation tick time is 1 picosecond

12

Simulation and Performance Equation

- Program runtime:
- $$\frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$
- **Instructions per program:** simulator can tell us directly
 - Including fractions of instruction types (e.g., %loads)
 - **Cycles per insn:** "CPI" also can come from simulation
 - Sometimes indirectly (e.g., output is $\text{CPI} \times t_{\text{CLK}}$)
 - This is often a complex function of other things:
 - Branch predictor
 - Cache behavior
 - Simulator can tell us model inputs (e.g., % predicted right)
 - **Seconds per cycle:** clock period, t_{CLK} – simulator input

13

13

How to learn more about gem5

- There is a great tutorial text:

learning.gem5.org
- Tutorial talks available on youtube:

www.youtube.com/watch?v=5UT41VsGTsg

14

14

Honesty is the Best Policy

- It is your job to design an honest simulator

`sim_cycle = sim_cycle/2`
 → 2x performance improvement! Woo hoo!
- Intel simulators have strict types

Latched structures "know" about cycles

 - throw error if you read more than n times per cycle
- What about cycle time?
 - What can you accomplish in hardware?
 - What can you accomplish in a cycle?

15

15

Sanity Checks

- You must convince yourself that your simulator is working
 - If you cannot, you will never convince anyone else!
- Branch predictor gets 50% performance improvement?
 - Initial stats showing the phenomenon you exploit
 - How many branches are there?
 - What does perfect branch prediction offer?
 - What does a stupid branch predictor offer?
 - Sensitivity studies showing how your idea changes across different values

If you don't back up your results with secondary data, people will just think you're lying.

16

16

What about power?

- Static power – "charge" each structure for length of run
 - Cache leaks certain amount of power just sitting there
 - Run for 200 ms, charge for that much leakage
- Dynamic power – "charge" per use
 - Read the cache 10,000 times in a run, charge for that
- What do we "charge" ? → **really hard** to get right
- In fact, $0 \rightarrow 1$ different cost than $1 \rightarrow 0$ (*yikes*)
- Most academic power numbers are basically crap
 - Squint. Trust the trends, not the numbers.

17

17