SimpleScalar v3.0 Tutorial

CS752
Fall 2013

Department of Computer Sciences
University of Wisconsin-Madison

Simulator 101

- What is an arch. simulator ?
  - Tool that reproduces the behavior of a computing device
- Why use a simulator ?
  - Flexible
    - Rapid design space exploration
    - Debugging
  - Cheap
    - evaluating different hardware designs without building costly physical hardware systems
- Why not use a simulator
  - Slow
  - Correctness ?

Functional vs. Performance

- Functional simulators implement architecture
  - emphasize achieving the same function as the modeled components
  - Implement what programmers see
- Performance/timing simulators implement Uarch
  - Model system resources/internals
  - Measure time
  - Implement what programmers do not see

Trace vs Execution-Driven Simulation

- Trace-based simulation
  - Reads a trace of insts. saved from previous execution
  - Easy to implement, no functional component needed
  - No feedback into trace (e.g., misspeculation)
- Execution-driven simulation
  - Runs program generating stream dynamically
  - More difficult to implement
  - Direct execution: instrumented program runs on the host

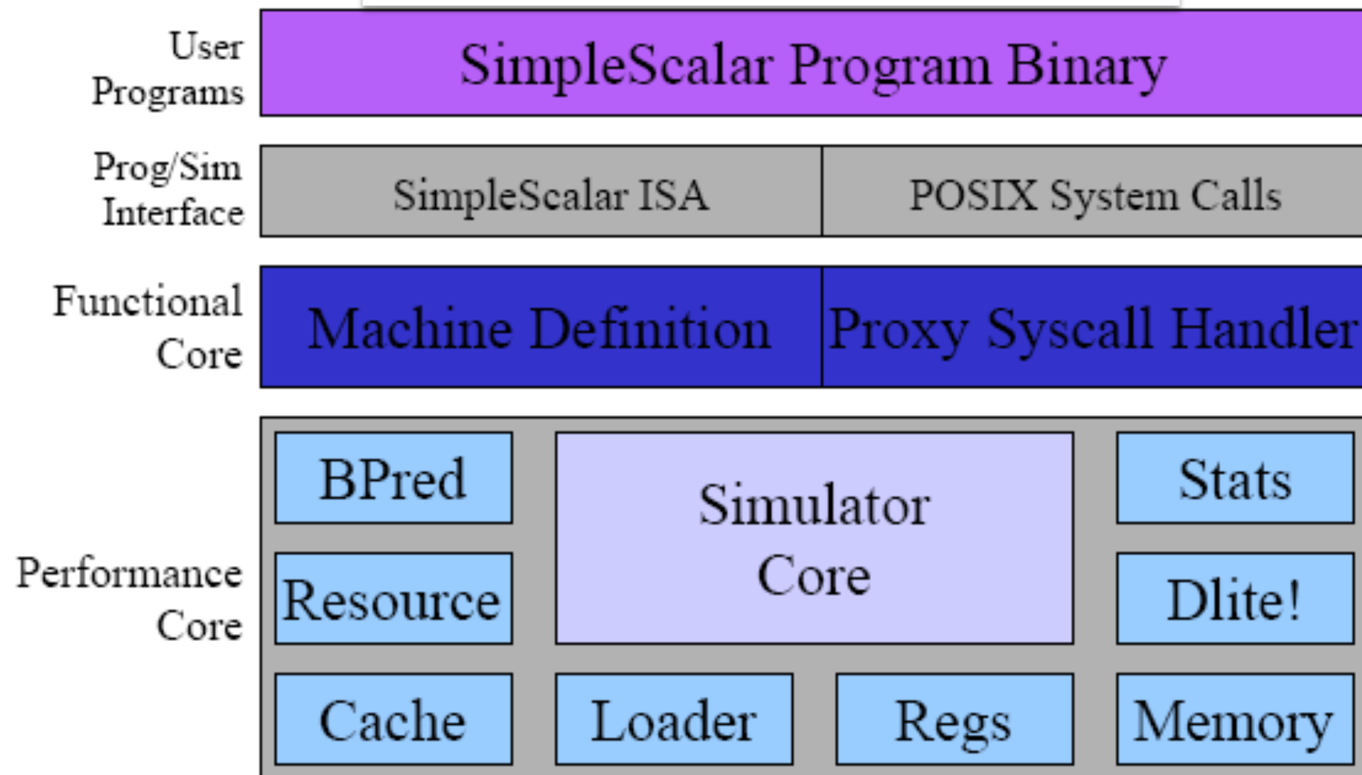
SimpleScalar

- Developed by Todd Austin with Doug Burger at UW, 1994-1996
- Execution-driven
- Collection of simulators that emulate microprocessor at different levels
  - Functional, functional+cache/bpred, out-of-order cycle-timer, etc
- Tools:
  - C/Fortran compiler, assembler, linker
  - Dlite: target-machine-level debugger
  - Pipeline trace viewer

Advantages of SimpleScalar

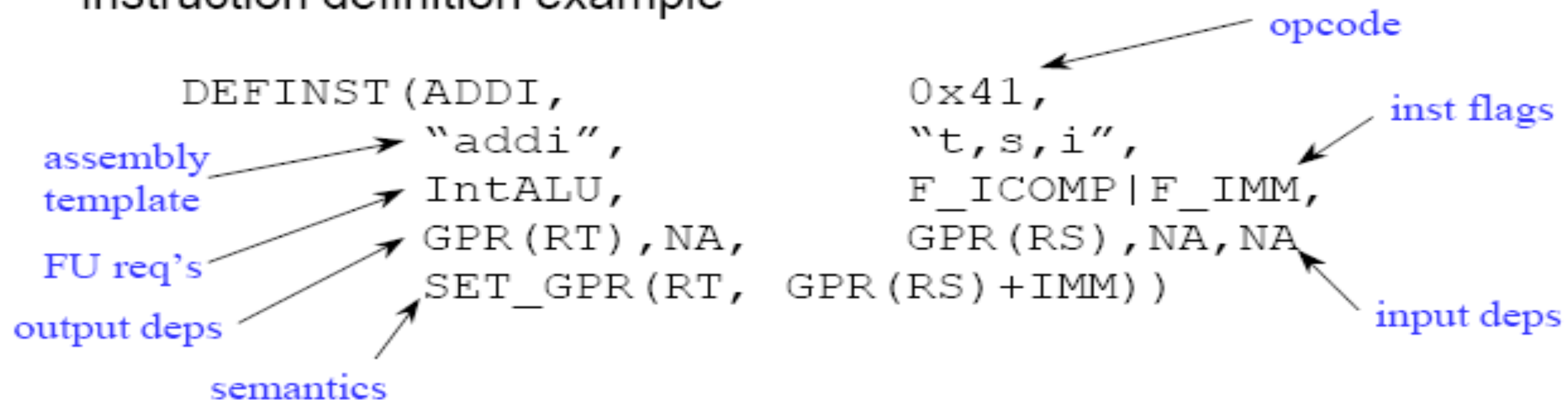
- Fast
  - Simulations won't last more than an hour
- Easy learning curve
- Modular design
- Well documented and commented code base
- Limitations summarized at the end

Simulator Structure

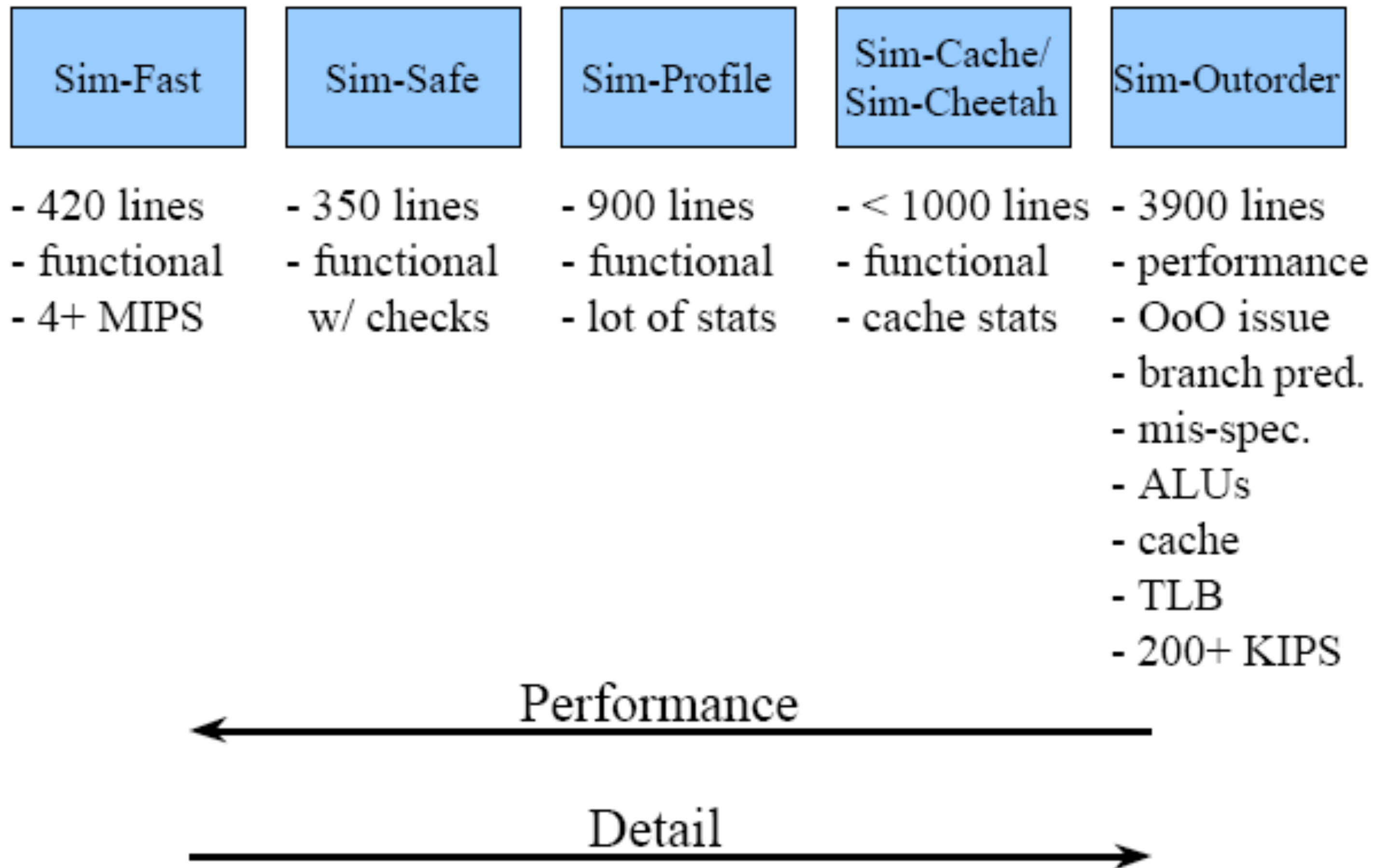


Machine Definition

- a single file describes all aspects of the architecture
  - used to generate decoders, dependency analyzers, functional components, disassemblers, appendices, etc.
  - e.g., machine definition + 10 line main == functional simulator
  - generates fast and reliable codes with minimum effort
- instruction definition example



Simulator Suite Overview



Sim-Fast

- Bare functional simulator
- Does not account for the behavior of any part of the microarchitecture
- Optimize for speed

Sim-Safe

- Similar to sim-fast (slower)
- Implements some memory op safeguards
  - Memory alignment
  - Memory access permission
- Good for debugging if sim-fast crashes

Sim-Cache/Sim-Bpred

- Functional core drives the detailed model of cache/branch predictor
- Similar to trace-driven cache/bpred simulation
- Fast results for miss ratio/rates
- No timing simulation/performance impact

Cache implementation

- Block size, # sets, associativity, all customizable
- Replacement policies: random, FIFO, LRU
- 2-level cache hierarchy supported (can be easily extended)
- Unified/separate I/D L1 and L2 caches

Specifying Caches parameters

- Creation
  - -cache:dl1 <config>
  - -cache:dl2 <config>
  - -cache:il1 <config>
  - -cache:il2 <config>
  - -tlb:dtlb <config>
  - -tld:itlb <config>
- <Config>
  - <name>:<nsets>:<bsize>:<assoc>:<repl>
- Example
  - -cache:il1 il1:128:64:1:l
  - -cache:il2 dl2
- Default
  - il1:256:32:1:l
  - dl1:256:32:1:l
  - ul2:1024:64:4:l
  - itlb:16:4096:4:l
  - dtlb:32:4096:4:l

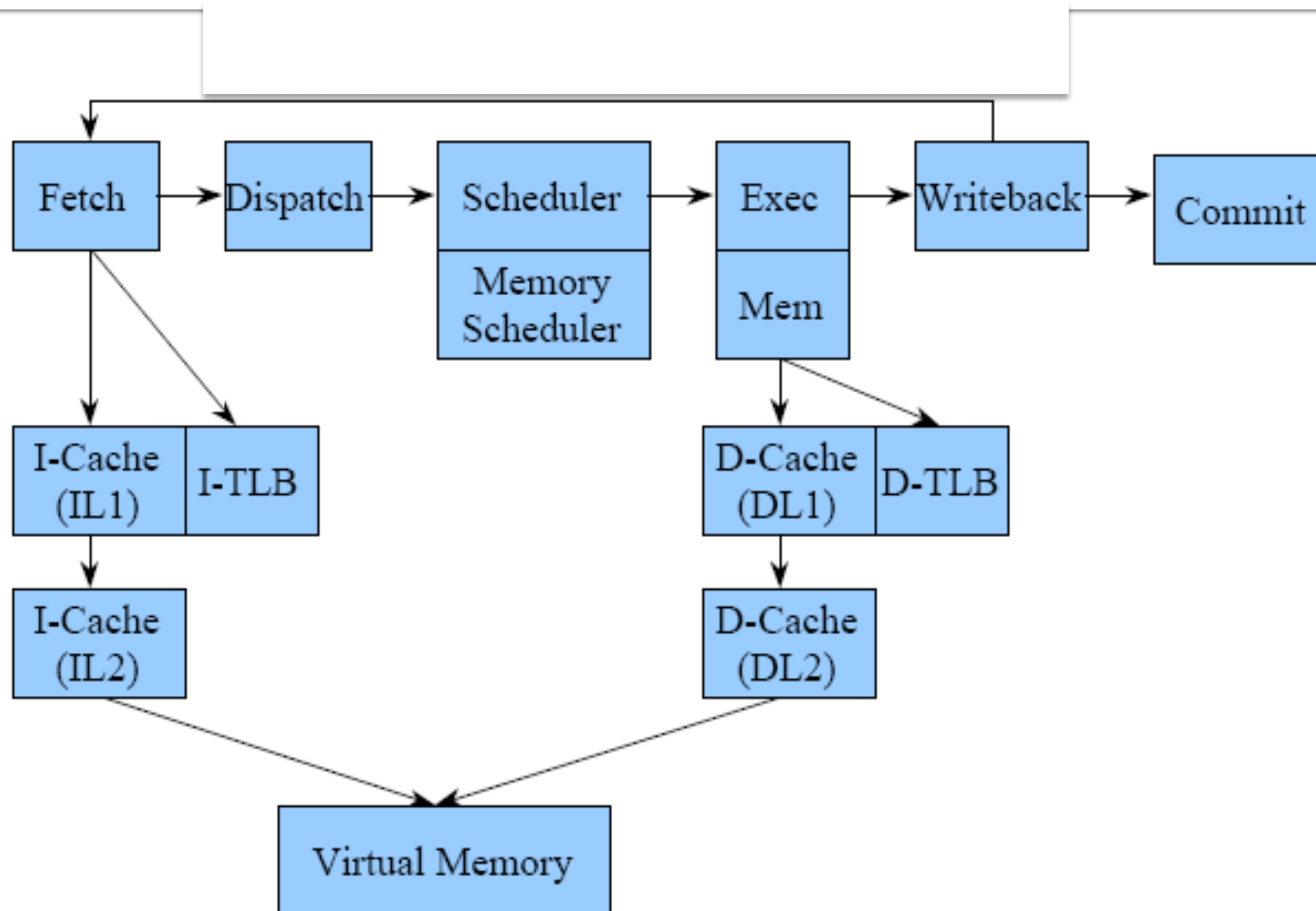
Specifying predictors

- Specifying branch predictor type
  - -bpred <type>
- <Type>
  - Nottaken
  - Taken
  - Perfect
  - Bimod <size>
  - 2lev <l1size> <l2size> <hist_size> <xor>
    - Can specify size, history bits, XOR PC is each level
    - Gag, Gap, Pag, Pap, gshare
  - Comb <size>

Sim-Outorder

- Detailed performance simulator
- Out-of-order execution core
- Register renaming, reorder buffer, speculative execution
- 2-level cache hierarchy
- Branch prediction

Out-of-order Issue Simulator



- implemented in `sim-outorder.c` and modules

Main Simulation Loop

Main Simulation Loop

```
for (;;) {
  ruu_commit();
  ruu_writeback();
  lsq_refresh();
  ruu_issue();
  ruu_dispatch();
  ruu_fetch();
}
```

- main simulator loop is implemented in `sim_main()`
- walks pipeline from Commit to Fetch
  - backward pipeline traversal eliminates relaxation problems, e.g., provides correct inter-stage latch synchronization
- loop is exited via a `longjmp()` to `main()` when simulated program executes an `exit()` system call

Stage Implementation

- Greater detail in SS hack guide and v2.0 tutorial
- Fetch (at ruu_fetch())
  - Fetch ins-s up to cache line boundary
  - Block on miss
  - Put in Fetch Queue (FQ)
  - Probe branch predictor for next cache line to fetch from

Stage Implementation

- Decode (at ruu_decode())
  - Fetch from FQ
  - Decode
  - Rename registers
  - Put into RUU and LSQ
  - Update RUU dependency lists, renaming table
  - **Sim (functional core):**
    - Execute instruction, update machine state
    - Detect branch misprediction, backup state (checkpoint)

Stage Implementation

- Issue (at ruu_issue() and lsq_refresh())
  - Ready queue -> Event queue
  - Order based on policy (see ready_queue slide)
  - Check and reserve FU's
  - Mem. Ops check memory dependences
    - No load speculation ("maybe" dependence respected)

Stage Implementation

- Writeback (at ruu_writeback())
  - Get finished instructions from ready queue
  - Wake up (put in ready queue) instructions with ready operands (use dependence list)
  - Performance core detects misprediction here and rolls back the state

Stage Implementation

- Commit (at ruu_commit())
  - Service D-TLB misses
  - Update register file (logically) and rename table
  - Retire stores to D-cache
  - Reclaim RUU/LSQ entries of retirees

Limitations

- I/O and other system calls
  - Only some limited functional simulation
- Lacks support for arbitrary speculation
  - Only branches can cause rollback
  - No speculative loads
  - Harder problem: Decoupling of functional and timing cores (for performance) complicates data speculation extensions

Limitations of Memory System

- No causality in memory system
  - All events are calculated at time of first access
- Simple TLB and virtual memory model
  - No address translation
  - TLB miss is a (small) fixed latency parallel with cache access
- Bandwidth, non-blocking
  - Modeled as n FU's (read/write ports with fixed issue delay)
  -
- **Accurate if memory system is lightly utilized**
  - SMT extensions?
- Overhaul required for multiprocessor simulation

Installation Demo

Questions ?

- Email: sridhara@cs.wisc.edu
- Subject: CS752