SimpleScalar

This document should act as a quick reference to the SimpleScalar out-of-order issue processor simulator that we will use throughout much of this course.

What is it?

SimpleScalar is a suite of processor simulators and supporting tools. The simulation architecture is called PISA, and is similar to the MIPS architecture studied in CSE378.

Sim-outorder is an instruction-level simulator of an out-of-order issue superscalar processor. The memory system is two-level and there is speculative execution support. This is a performance simulator, tracking the latency of all pipeline operations. In order to track latency and contention for resources, sim-outorder does a lot of work. This makes it slow. It executes perhaps half a million instructions every second, while the machine you run it on executes maybe 2 billion instructions. This makes sim-outorder around 4000 times slower than an actual processor. So remember, a program that took 1 second to run on a real computer would take over half an hour to run in sim-outorder.

The toolkit also contains several other simulators. Sim-fast and sim-safe simulate the execution of instructions, but do not model any processor internals. There is no pipeline, one instruction is fetched, executed and completed each "cycle." They run 4-8 times faster than sim-outorder, but provide no detail about what happened during execution.

What isn't it?

SimpleScalar doesn't have a graphical front-end like xspim or pcspim. It does not simulate an operating system, though a limited number of system calls are supported with the help of the host operating system.

Finding out more

You can learn more about SimpleScalar and find documentation at http://www.simplescalar.com. The documentation on the website matches the versions of the tools available on the instructional machines. The user's guide and hacker's tutorial may both be useful.

Running sim-outorder

You can find the sim-outorder binary in the directory
/cse/courses/cse471/06sp/simplescalar/bin/
Optionally you can add this path to your PATH environment variable.

To invoke the simulator, type:
sim-outorder {simulator-options} *simulated-program* {program-arguments}

It is often handy to redirect a file to standard input by using < *input file name*.

When the simulator is running, it produces no output at all for sometimes minutes on end. This is normal, if frustrating behavior.

General Options

Option	Arguments	Default

-config <string> <none>
Load the configuration parameters from a file (one option per line).

-dumpconfig <string> <null>
Dump the configuration parameters to a file.

-h <true|false> false
Print help message.

-v <true|false> false
Verbose operation.

-d <true|false> false
Enable debug messages.

-i <true|false> false
Start in Dlite debugger.

-seed <int> 1
Random number generator seed (0 for timer seed).

-q <true|false> false
Initialize and terminate immediately.

-chkpt <string> <null>
Restore EIO trace execution from a file.

-redir:sim <string> <null>
Redirect simulator output to file (non-interactive only).

-redir:prog <string> <null>
Redirect simulated program output to file.

-nice <int> 0
Simulator scheduling priority.

-max:inst <uint> 0
Maximum number of instructions to execute.

-fastfwd <int> 0
Number of instructions skipped before timing starts.

-ptrace <string list...> <null>
Generate pipetrace <fname|stdout|stderr> <range> (see below).

-pcstat <string list...> <null>
Profile stat(s) against text addresses (multiple uses ok).

-bugcompat <true|false> false
Operate in backward-compatible bugs mode (for testing only).

Pipetrace range arguments are formatted as follows:

{{@|#}<start>}:{{@|#|+}<end>}

Both ends of the range are optional, if neither are specified, the entire execution is traced. Ranges that start with a `@' designate an address range to be traced, those that start with an `#' designate a cycle count range. All other range values represent an instruction count range. The second argument, if specified with a `+', indicates a value relative to the first argument, e.g., 1000:+100 == 1000:1100. Program symbols may be used in all contexts.

Examples:
-ptrace FOO.trc #0:#1000
-ptrace BAR.trc @2000:
-ptrace BLAH.trc :1500
-ptrace UXXE.trc :
-ptrace FOOBAR.trc @main:+278

Processor Configuration Options

Option	Arguments	Default

-fetch:ifqsize <int> 4
Instruction fetch queue size (instructions).

-fetch:mplat <int> 3
Extra branch mis-prediction latency.

-fetch:speed <int> 1
Speed of front-end of machine relative to execution core.

-decode:width <int> 4
Instruction decode bandwidth (instructions/cycle)

-issue:width <int> 4
Instruction issue bandwidth (instructions/cycle)

-issue:inorder <true|false> false
Run pipeline with in-order issue.

-issue:wrongpath <true|false> true
Issue instructions down wrong execution paths.

-commit:width <int> 4
Instruction commit bandwidth (instructions/cycle).

-ruu:size <int> 16
Register update unit (RUU) size.

-lsq:size <int> 8
Load/store queue (LSQ) size.

-res:ialu <int> 4
Total number of integer ALUs available.

-res:imult <int> 1
Total number of integer multiplier/dividers available.

-res:memport <int> 2
Total number of memory system ports available (to CPU).

-res:fpalu <int> 4
Total number of floating point ALUs available.

-res:fpmult <int> 1
Total number of floating point multiplier/dividers available.

Branch Predictor Configuration Options

Option Arguments Default

-bpred <string> bimod
Branch predictor type {nottaken|taken|perfect|bimod|2lev|comb}

-bpred:bimod <int> 2048
Bimodal predictor (uses a branch target buffer with 2 bit counters) table size.

-bpred:2lev <int list...> 1 1024 8 0
2-level predictor configuration (l1size l2size hist_size xor).

-bpred:comb <int> 1024
Combining predictor meta table size.

-bpred:ras <int> 8
Return address stack size (0 for no return stack).

-bpred:btb <int list...> 512 4
BTB configuration (num_sets associativity)

-bpred:spec_upde <string> <null>
Speculative predictors update in {ID|WB} (default non-speculative).

Branch predictor configuration examples for 2-level predictor:

Configurations: N, M, W, X

N	# entries in first level (# of shift register(s))
W	width of shift register(s)
M	# entries in 2nd level (# of counters, or other FSM)
X	(yes-1/no-0) xor history and address for 2nd level index

The predictor `comb' combines a bimodal and a 2-level predictor.

<u>Memory Subsystem Configuration Options</u>

<u>Option</u> <u>Arguments</u> <u>Default</u>

-cache:dl1 <string> dl1:128:32:4:l
L1 data cache configuration {<config>|none} (see below).

-cache:dl1lat <int> 1
L1 data cache hit latency (cycles).

-cache:dl2 <string> ul2:1024:64:4:l
L2 data cache configuration {<config>|none} (see below).

-cache:dl2lat <int> 6
L2 data cache hit latency (cycles).

-cache:il1 <string> il1:512:32:1:l
L1 inst cache configuration {<config>|dl1|dl2|none} (see below).

-cache:il1lat <int> 1
L1 instruction cache hit latency (cycles).

-cache:il2 <string> dl2
L2 instruction cache configuration {<config>|dl2|none} (see below).

-cache:il2lat <int> 6
L2 instruction cache hit latency (cycles).

-cache:flush <true|false> false
Flush caches on system calls.

-cache:icompres <true|false> false
Convert 64-bit inst addresses to 32-bit inst equivalents.

-mem:lat <int list...> 18 2
Memory access latency (<first_chunk> <inter_chunk>).

-mem:width <int> 8
Memory access bus width (bytes).

-tlb:itlb <string> itlb:16:4096:4:l
Instruction TLB configuration {<config>|none} (see below).

-tlb:dtlb <string> dtlb:32:4096:4:l
Data TLB configuration {<config>|none} (see below).

-tlb:lat <int> 30
Inst/data TLB miss latency (cycles).

The cache configuration parameter <config> has the following format:

<name>:<nsets>:<bsize>:<assoc>:<repl>

<name>	name of the cache being defined
<nsets>	number of sets in the cache
<bsize>	block size of the cache
<assoc>	associativity of the cache
<repl>	block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random

Examples:

-cache:dl1 dl1:4096:32:1:l
-dtlb dtlb:128:4096:32:r

Cache levels can be unified by pointing a level of the instruction cache hierarchy at the data cache hierarchy using the "dl1" and "dl2" cache configuration arguments. Most sensible combinations are supported, e.g.,

A unified l2 cache (il2 is pointed at dl2):

-cache:il1 il1:128:64:1:l -cache:il2 dl2
-cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l

Or, a fully unified cache hierarchy (il1 pointed at dl1):

-cache:il1 dl1
-cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l

Simulation Outputs

General Simulation Statistics

sim_num_insn	total number of instructions committed
sim_num_refs	total number of loads and stores committed
sim_num_loads	total number of loads committed
sim_num_stores	total number of stores committed
sim_num_branches	total number of branches committed
sim_elapsed_time	total simulation time (seconds)
sim_inst_rate	simulation speed (instructions/second)
sim_total_insn	total number of instructions executed
sim_total_refs	total number of loads and stores executed
sim_total_loads	total number of loads executed
sim_total_stores	total number of stores executed
sim_total_branches	total number of branches executed
sim_cycle	total simulation time (cycles)
sim_IPC	instructions per cycle
sim_CPI	cycles per instruction
sim_exec_BW	total instructions (mis-speculated + committed) per cycle
sim_IPB	instructions per branch

Instruction Fetch Queue (IFQ) Statistics

IFQ_count	cumulative IFQ occupancy
IFQ_fcount	cumulative IFQ full count
ifq_occupancy	average IFQ occupancy (instructions)
ifq_rate	average IFQ dispatch rate (instructions/cycle)
ifq_latency	average IFQ occupant latency (cycles)
ifq_full	fraction of time (cycles) IFQ was full

Register Update Unit (RUU) Statistics – a.k.a. commit unit

RUU_count	cumulative RUU occupancy
RUU_fcount	cumulative RUU full count
ruu_occupancy	average RUU occupancy (instructions)
ruu_rate	average RUU dispatch rate (instructions/cycle)
ruu_latency	average RUU occupant latency (cycles)
ruu_full	fraction of time (cycles) RUU was full

Load/Store Queue (LSQ) Statistics

LSQ_count	cumulative LSQ occupancy
LSQ_fcount	cumulative LSQ full count
lsq_occupancy	average LSQ occupancy (instructions)
lsq_rate	average LSQ dispatch rate (instructions/cycle)
lsq_latency	average LSQ occupant latency (cycles)

lsq_full	fraction of time (cycles) LSQ was full

Instruction Issue and Retirement (Commit) Statistics

sim_slip	total number of slip cycles
avg_sim_slip	the average slip between issue and retirement

Branch Predictor Statistics

JR = Jump Register instruction

bpred_bimod.lookups	total number of branch predictor lookups
bpred_bimod.updates	total number of updates
bpred_bimod.addr_hits	total number of address-predicted hits
bpred_bimod.dir_hits	total number of direction-predicted hits (includes addr_hits)
bpred_bimod.misses	total number of misses
bpred_bimod.jr_hits	total number of address-predicted hits for JRs
bpred_bimod.jr_seen	total number of JRs seen
bpred_bimod.jr_non_ras_hits.PP	total number of address-predicted hits for non-return address stack (RAS) JRs
bpred_bimod.jr_non_ras_seen.PP	total number of non-RAS JRs seen
bpred_bimod.bpred_addr_rate	branch address-prediction rate (address-hits/updates)
bpred_bimod.bpred_dir_rate	branch direction-prediction rate (all-hits/updates)
bpred_bimod.bpred_jr_rate	JR address-prediction rate (JR addr-hits/JRs seen)
bpred_bimod.bpred_jr_non_ras_rate.PP	non-RAS JR address-prediction rate (non-RAS JR hits/JRs seen)
bpred_bimod.retstack_pushes	total number of address pushed onto RAS
bpred_bimod.retstack_pops	total number of address popped off of RAS
bpred_bimod.used_ras.PP	total number of RAS predictions used
bpred_bimod.ras_hits.PP	total number of RAS hits
bpred_bimod.ras_rate.PP	RAS prediction rate (RAS hits/used RAS)

Cache Statistics

These are gathered appropriately for il1, dl1, il2, dl2, ul1, ul2, itlb and dtlb:

cache.accesses	total number of accesses
cache.hits	total number of hits
cache.misses	total number of misses
cache.replacements	total number of replacements
cache.writebacks	total number of writebacks
cache.invalidations	total number of invalidations
cache.miss_rate	miss rate (misses/reference)
cache.repl_rate	replacement rate (replacements/reference)
cache.wb_rate	writeback rate (wrbks/ref)
cache.inv_rate	invalidation rate (invs/ref)

Miscellaneous Statistics

sim_invalid_addrs	total non-speculative bogus addresses seen (debug variable)

ld_text_base	program text (code) segment base
ld_text_size	program text (code) size (bytes)
ld_data_base	program initialized data segment base
ld_data_size	program initialized `.data' and uninitialized `.bss' size (bytes)
ld_stack_base	program stack segment base (highest address in stack)
ld_stack_size	program initial stack size
ld_prog_entry	program entry point (initial PC)
ld_environ_base	program environment base address address
ld_target_big_endian	target executable endian-ness, non-zero if big endian
mem.page_count	total number of pages allocated
mem.page_mem	total size of memory pages allocated
mem.ptab_misses	total first level page table misses
mem.ptab_accesses	total page table accesses
mem.ptab_miss_rate	first level page table miss rate