

ECE 5752 - Spring '05

Programming Exercise #2

Revision B

Various tools & Modifying the source code in SimpleScalar 3.0d

Posted: Monday 2/14

Due: Monday 2/28

All work should be done individually, no group work allowed for this assignment

Purpose:

The purpose of this exercise is to learn more about the SimpleScalar environment and use this environment in a productive manner to evaluate multiple microprocessor architectures. The student should complete following objectives to satisfy the requirements of this assignment.

- Vary cache parameters to a simulation performed by sim-cache
- Vary branch prediction methods to a simulation performed by sim-bpred
- Vary functional unit allocations in a simulation performed by a modified sim-outorder
- Add the parameter corresponding to a cache enhancement to the-sim-outorder tool
- Generate a histogram of accesses to the off-chip memory space

This assignment will lead up to the subsequent assignments in which you will augment the `mem_access_latency()` function to more accurately reflect the latency of the main memory system.

Problem Specification:

For completion of this assignment, each student will be required to run three separate simulation engines {sim-cache, sim-bpred, and a modified sim-outorder} each upon on three separate architectural configurations. Each of these simulations should reflect the execution of the same (4) benchmarks and input pairings. Once the results have been acquired, comparisons should be made between the results off the 36 simulation tool and configuration pairings.

Modifications to the source code of sim-outorder are required. These changes are focussed upon the addition of parameters for a cache extension and a statistical examination of the address stream.

It is recommended that you follow the suggested procedure, but if you are able to complete the requirements of the problem, the procedure is secondary.

Suggested Procedure:

It is suggested that the following procedure be followed on one of the COLOR machines in the btdpool, or any available Linux machine. Beware, the btdpool is not designed for, and is rarely used in a classroom environment - if there are any machine failures or other issues to report, please direct them to *btdavis@mtu.edu*.

1. Using the sim-* binaries generated in assignment # 1, generate the default configuration files for each of the three simulation tools (sim-cache, sim-bpred, sim-outorder) which will be used for this assignment. Use the -dumpconfig option of the simulators to do this.
2. Copy and modify the default configuration files for these three simulators to generate configuration files which correspond to the configurations given in the table below.

Simulation Engine	Configuration Requirements
sim-cache	I-side L1 : 8KB direct-mapped cache w/ 32 byte line size using LRU replacement D-side L1 : 8KB direct-mapped cache w/ 32 byte line size using LRU replacement Unified L2 : 256KB 4-way set associative cache w/ 64 byte line size using LRU replacement
	Unified L1 : 32KB 2-way set associative cache w/ 32 byte line size using LRU replacement Unified L2 : 1MB 4-way set associative cache w/ 128 byte line size using LRU replacement
	I-side L1 : 32KB 2-way set associative cache w/128 byte line size using LRU replacement D-side L1 : 64KB 2-way set associative cache w/32 byte line size using LRU replacement I-side L2 : 256KB 4-way set associative cache w/128 byte line size using LRU replacement D-side L2 : 256KB 2-way set associative cache w/128 byte line size using LRU replacement
sim-bpred	Two-bit (bimodal) prediction based upon a 2048 entry tagless table indexed by the branch address, a return address stack with 8 entries, and a 4-way set associative BTB with 2048 entries
	Two-level adaptive GAg prediction with a 12-bit history register, a return address stack with 8 entries, and a 2-way set associative BTB with 2048 entries
	Hybrid prediction based upon a 1024 entry tagless bimodal table, a two-level gshare scheme with a 10 bit history register, combined using a meta-predictor with 512 entries, a return address stack with 8 entries, and a 4-way set associative BTB with 2048 entries

Simulation Engine	Configuration Requirements
sim-outorder	out-of-order microprocessor with 4-deep fetch queue, 4-wide decode, 4-wide issue, 4-wide commit, 4 integer ALU's, 1 integer mult/div unit, 2 load-store pipelines, 4 floating-point ALU's, 1 floating-point mult/div units, an 8 entry load/store queue, initial DRAM latency of 18 cycles, with subsequent bus-retrievals every 2 cycles, and a 64bit DRAM bus
	in-order microprocessor with 4-deep fetch queue, 4-wide decode, 4-wide issue, 4-wide commit, 4 integer ALU's, 1 integer mult/div unit, 2 load-store pipelines, 4 floating-point ALU's, 1 floating-point mult/div units, an 8 entry load/store queue, initial DRAM latency of 60 cycles, with subsequent bus-retrievals every 10 cycles, and a 256bit DRAM bus
	out-of-order microprocessor with 16-deep fetch queue, 8-wide decode, 8-wide issue, 8-wide commit, 6 integer ALU's, 2 integer mult/div unit, 4 load-store pipelines, 4 floating-point ALU's, 2 floating-point mult/div units, a 32 entry load/store queue, initial DRAM latency of 45 cycles, with subsequent bus-retrievals every 5 cycles, and a 64bit DRAM bus

Parameters not mentioned in the table may be any value you wish, but must be consistent between simulations.

- Execute the appropriate simulation tool on each of the 6 configuration files for the sim_bpred and sim_cache simulation engines you have generated. The benchmark and input set to be used is up to your discretion, but must execute at least 500 Million instructions. Similarly, the same benchmarks and input sets must be used for all (9) simulator and configuration pairings.
- Compare the three cache configurations and the three branch prediction configurations. Order the performance of the three configurations and hypothesize why the performance may be in the order of the results based upon the configurations. Include this work and thought processes in your report conclusions.
- Generate the graphs required in the deliverables and solely dependant upon the results from the sim_bpred and sim_cache simulations. Namely graphs showing bpred_dir_rate and bpred_addr_rate for sim_bpred results and average_access_time for sim_cache results. The

Cache Size	8KB	32KB	64KB	256KB	256KB	1MB
Associativity	Direct Mapped	Two-way	Two-way	Two-way	Four-way	Four-way
Latency	2nS	4nS	5nS	6nS	8nS	10nS

average access time should be calculated based upon the latencies given in the table above, and the miss rates from the simulation results. In a split memory hierarchy base average_access_time upon data side references.

The functions sim_reg_stats() and sim_reg_options() are common to all of the SimpleScalar tools. These functions are included in the primary file differentiating the various tools, namely sim-*.c. In these functions, you can add statistics to be reported at completion of the simulation, or input parameters (options) describing the behavior of the simulation. Before modifying the code you have for these simulation tools, it may be wise to create a "working" copy of the simulator being modified. This can be done by (a) copying sim-outorder.c to sim-[unique_identifier].c; then

modifying the makefile to compile this unique simulator or (b) copying the source directory and maintaining a working directory. In either case you should continue to use benchmark & intpu sets from the /Public directory, only replcicae source code.

6. REG_OPTIONS: The parameter (option in SimpleScalar) added for this assignment should be the specification of a cache extension covered in the class, using when possible the same format as existing caches. It is expected that the specification of a cache extension will require multiple variables such as the specification of "-cache:il1" which requires the use of the `opt_reg_str()` function. It is suggested that you add your code at the beginning of `sim_reg_options()`. All code additions should be well commented. Your code should also include at least a single conditional in `sim_check_options()` which verifies a correct specification. This parsing and checking of the string can go so far as to save the individual components into a representative data object if desired. The implementation of the cache extension need not be completed for this assignment.
7. REG_STATS: The output statistic should be a histogram of addresses passing through the function `mem_access_latency()`. Currently the function `mem_access_latency()` is not very accurate and returns the `memlat + (block_size/word_width)*clock_period`. This does not incorporate the any consequences of address being accessed. It will be necessary to add the address (`baddr`) to the `blk_sz` parameter and pass that from the (4) functions which call `mem_access_latency()`. Then, in `mem_access_latency` divide the addresses into 32 equal divisions of the addressing range. The divisions should be based upon bits 31-27 of `baddr`. This is dividing the addressing space into a histogram of 32 bits. The data corresponding to this histogram can be stored in a `stat_stat_t` structure which is dynamically instantiated by the `stat_reg_dist()` function. This invocation of `stat_reg_dist()` should be performed in `sim_reg_stats()`. Look at `stats.c` and the invocations of the `stat_reg_*` functions in `sim_reg_stats()` for examples on how this is done. As each reference is made to main memory, as observed by the `mem_access_latency()` function, the histogram bucket corresponding to the division of the addressing space to which the reference is being made can be incremented. The histogram buckets in the `stat_stat_t` structure should be incremented using the `stat_add_samples()` function.
8. Generate the graphs required for the deliverables which are based upon the results generated by the modified `sim-outorder`. This includes one graph for `sim_cycle` and one graph for `sim_CPI`.
9. Now compare between the three simulation tools, the simulated instructions per second are reported as `sim_inst_rate` by `simplescalar`. Generate a graph showing the `sim_inst_rate` for each of the (12) simulation tool & benchmark pairings, averaged accross configuration files.

Considerations:

1. Examine the documentation for the simulation tools. Notice it is possible to parameterize these tools based upon either a configuration file, or command line parameters. In this assignment you were required to use a configuration file, are there any advantages to either the configuration file invocation style or the command line style? Which would you chose to use for architectural simulations and why?
2. For each of the L2 cache configurations used for `sim_cache`, diagram and divide the address into it's component parts (i.e. tag, index and byte_in_line). Assuming 32-bit physical addresses, state which bits are used for each of these three component parts.

3. Calculate the space requirements (in bits or bytes) of each of the branch prediction schemes given for the sim-bpred tool.
4. After your minor modifications to the SimpleScalar source code, do you have any opinion on the organization of the code? Were the modifications you made intuitive or difficult?
5. Compare the execution speed (sim_inst_rate) of the three simulation engines. Why does one have a significantly lower rate (or slower simulation rate)? If you were going to write a simulation tool built upon the SimpleScalar tool set, which tool would you choose if you were only able to modify one?

Deliverables

Many of the deliverables - the text file outputs can be included in the report two-pages to each side of a piece of paper. The report should remain full page size text.

1. Assignment report, in an appropriate format. Including answers to the Consideration questions above. Reports should include: Purpose, Procedure - including suggested revisions or issues which caused problems, Results, Conclusions, answers to Consideration questions, all Deliverables and whatever else you feel is appropriate.
2. The results of a diff run on the (3) configuration files used for each of the (3) simulation engine. The entire configuration file should not be printed.
3. An edited (to one page) output from each of the (9) simulation runs (3 simulation engines, 3 configurations) showing initialization of the cache enhancement (for the modified sim-outorder simulations) and the following statistics when available:

sim_cycle	sim_num_insn	sim_num_refs
sim_num_loads	sim_num_stores	sim_num_branches
sim_inst_rate	sim_CPI	ruu_occupancy
bpred_dir_rate	bpred_addr_rate	bpred_*.updates
and for all caches		
*.accesses	*.hits	*.miss_rate

And anything else you feel is relevant

4. Calculations showing the how average access time was calculated from the results of each of the sim_cache simulations. Again, if caches are split, base this access time upon the data side accesses.

The X-axis should the graphs below should be a unique configuration identifier. Using this format, provide graphs for each of the following

5. A graphical representation of the results of the sim_cache simulation runs showing:
 - a). average access time
6. A graphical representation of the results of the sim_bpred simulation runs showing:
 - a). bpred_addr_rate
 - b). bpred_dir_rate

Both of these statistics can be presented in a single figure

7. A graphical representation of the results of the sim-outorder simulation runs showing:
 - a). Execution Time (sim_cycle)
 - b). Cycles per instruction (sim_CPI)Each statistic will require a unique figure.
8. A graphical representation of the simulation throughput of each of the (9) simulations runs showing:
 - a). sim_inst_rate