

ECE 5900_04 - Spring '01

Laboratory Exercise #2

Using SimpleScalar 3.0 for Architectural Analysis

Assigned: 2/1/01

Due: 2/15/01

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.

- Generate and use configuration files within the simplescalar environment
- 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 sim-outorder
- Add one new parameter and one new statistics to a SimpleScalar tool

Problem Specification:

For completion of this assignment, each student will be required to run three separate simulation engines each upon on three separate architectural configurations. Once results have been acquired for each of these simulations, comparisons should be made between the results off the three different configurations for each

In addition, minor changes to the source code to add an parameter and a statistic are also required. These additions may have no significant usage at this point, but are intended to introduce the student to modification of the source code, which will be the focus of subsequent assignments.

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 “kirchhoff#” machines in the 7th floor lab.

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. Parameters

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

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 9 configuration files you have generated. The benchmark and input set to be used is up to your discretion, but must execute at least 150 Million instructions. Similarly, the same benchmark and input set must be used for all configurations run by the same simulation tool, but may be different between simulation tools.
- Compare the results of your simulations which use the same simulation tool. (i.e. compare the three cache configurations, the three branch prediction configurations, and the three pipeline 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.

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 output by the simulation, or input parameters to the simulation respectively. Before modifying the code you have for these simulation tools, it may be wise to create a “working” directory in which to edit code.

- For the final component of this assignment, it is required that you add both an option, and a statistic to one of the simulation tools. My recommendation is that you modify `sim-outorder.c`, but that is a recommendation only.
- Add an option to the simulator chosen, in `sim_reg_options()` - this option need not perform any function as of yet, and it is my recommendation that you add a boolean “`my_debug`” flag which you will use for later assignments.
- Add a statistic, this statistic may be a constant value which is never changed in the source code, but must be a defined variable that COULD be changed with further code enhancements. Alternatively you could generate a statistic from variables already present in the simulation environment. An example of the former would be a floating point value which COULD keep track of the average memory latency time of a realistic DRAM simulation. An example of the

later would be a calculation which determines the number of simulated cycles per DRAM access (or L2 miss).

8. Verify that after your source-code changes the simple-scalar 3.0 toolset still builds properly. Run your modified simulation tool on a single configuration file - use the same tool & configuration file for which you have results from the first portion of this assignment. Verify that your source-code changes generate modified output from the simulation execution.

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. Compare the execution speed (`sim_inst_rate`) of the three simulation engines. Why does one have a significantly lower rater (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?
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 3.0 source code, do you have any opinion on the organization of the code? Were the modifications you made intuitive or difficult?

Deliverables

1. Lab report, format specified in lab guidelines, Conclusions should include the performance comparisons between each of the three configurations examined for each simulation tool.
2. Configuration files corresponding to the 9 configurations given in the Procedure section.
3. Output from the (3) `sim-outorder` simulations.
4. Output from the modified simulation tool with new parameter/statistic.