University of Southern California

Department of Electrical Engineering

EE557 Fall 2022

Simulation Assignment 2

Instructor: John Paul Walters

Due: 11:59 PM., Tuesday, October 11th

TOTAL SCORE: / 5

I. Simulation Basics

The purpose of this project is for you to gain experience exploring the design space of a typical microarchitecture enhancement to an existing processor design. Computer architects in industry use software based microarchitectural simulation tools to analyze bottlenecks, rapid prototyping of new ideas, and to compare the relative merits of different ideas.

In the analysis phase, they use simulations to understand where existing performance/power (anything else of importance) bottlenecks are in a given microarchitecture. Most, if not all, simulators take as one of the inputs a configuration file that describes the underlying microarchitecture of the machine using several knobs. For instance, the size of L1 cache, number of register read ports etc. are specified in this configuration file. Another important input to the simulator is a benchmark or trace file. The simulator reads the microarchitectural configuration file and configures a CPU with the specified parameters. The CPU simulator runs the benchmark/trace file and collects detailed statistics on various components of interest. For instance, the simulator may measure the number of cycles wasted during a benchmark execution due to the unavailability of register read ports. Such detailed statistics give insights into where the most significant bottlenecks are.

Once the bottleneck analysis is done, then architects put their knowledge into practice. In this phase they think of new microarchitectural enhancements that will alleviate the observed bottlenecks. For instance, a new dynamic instruction scheduling heuristic that predicts the potential register read port usage in the near future and prioritizes scheduling to reduce the peak read port usage. Once the idea is hatched, they have to show how effective the proposed idea is (remember all ideas need to be justified). For this purpose, the proposed idea is implemented as a new enhancement in the simulator. During the rapid prototyping phase the simulator is hacked where architects not only implement the proposed idea but also add code to measure activity factors in the new approach. The same set of benchmarks as were used in the first phase is simulated under the newly enhanced microarchitecture simulator.

In the final phase detailed statistics from the original simulation are compared with the enhanced simulator results to judge the improvement (performance/power/area…), if any, of the proposed idea. Then based on new findings the proposed idea may be refined further and Phases 2 and 3 may be repeated multiple times.

In this project you will get a taste of the analysis phase, and gain experience analyzing several design choices and the impact of these choices on the performance of an out-of-order superscalar processor.

II. Project Description

In this project we will use an existing out-of-order superscalar processor simulator called SimpleScalar as the simulation tool. This simulator, like many other processor simulators, reads a machine configuration file and generates a processor model that matches the requirements specified in the configuration file. The configuration file specifies parameters such as the size and associativity of caches, number of Reservation Stations, number of functional units etc. In EE557 we learned that Reservation Stations and Reorder Buffer (ROB) are implemented as two separate entities in a processor. However, in SimpleScalar simulator the merge the functionality of Reservation Station and ROB into a single entity called Register Update Unit (RUU). For our class purposes treat RUU as a combined Reservation Station and ROB. Other than this one quirk the remaining components of SimpleScalar match what we learned about out-of-order processors in class.

The second tool you will use in this class is Cacti. Cacti is a tool that estimates the access time and power consumption of any structure that stores data, such as RUU and cache structure. The input to Cacti is the size of the structure, Associativity and process technology used in building the structure. We will use Cacti to compute the latency of RUU and caches in this project.

Finally, you will use a simple area estimation tool that is provided to you as an Excel spreadsheet. The hardware complexity estimation tool can estimate the transistor count and chip space based on the SimpleScalar configuration parameters. The tool is implemented as an MS Excel spreadsheet. You can map sim-outorder's configuration into this tool and estimate the silicon area in terms of λ (which is equivalent to a half of the minimum feature size) of a processor. You can also estimate the transistor count of a processor. The estimation tool and its documentation are available on the class web page.

We will use all these three tools in this project. We provide some useful reading materials helping you get familiar with those tools. Please download the materials.zip from OneDrive.

Basic Project Steps

Here are the steps for doing the project:

1. In this first step, you will create a baseline configuration file using the parameters specified in Section III.d below. The baseline configuration file specifies the size and latencies of many structures, except the cache latencies that will be estimated in the following steps.

2. Using the Real Estate Estimator tool and the configuration file obtained in step 1 above, determine parameters such as port numbers that are used as inputs to the Cacti tool to complete step 3 and 4 below.
3. Compute the access time of RUU using Cacti 5.3's Pure RAM Interface*****. We assume that the RUU update is on the critical path, and the operating frequency of a processor is determined by this latency. **Round up to the nearest hundredth.**
4. Estimate the access time for the caches using Cacti5.3 Detailed Interface*****. You must convert the access time into the number of processor clock cycle. For example, if you get access time of 1 ns for RUU and 10 ns for the DL1, DL1 will have a latency of 10 cycles.
   ***** Refer to section III.b below to set parameters in the Cacti tool.
5. Use the RUU and Cache latencies estimated above to complete the baseline configuration file.
6. After completing the configuration file you are now ready to simulate the baseline processor configuration. Simulate the baseline configuration using two benchmarks provided to you and described in Section III.a below.
7. In this step you will explore how machine width affects the performance of a superscalar dynamically scheduled processor. You will change the machine width (fetch/decode/issue/commit) width in the configuration file. To explore the design space, you will first fix the fetch, decode and issue width to be 4, and explore the commit width 1, 2, 4 and 8. Then, fix the commit width to be 4, and explore the fetch/decode/issue width 1, 2, 4 and 8. Lastly, change fetch/decode/issue/commit width to be the same from 1, 2, 4 and 8.
8. Finally, you will generate the results, plot them and analyze them to show your understanding of this design space exploration.
   - Plot the MIPS vs. machine widths and explain these curves by observing various metrics such as the occupancy of varied buffers (instruction fetch queue, register update unit, and load/store queue), the dynamic frequency of branch instructions, the cache(s) hit rates, and etc.
   - In order to obtain the MIPS, you have to know the operating frequency of a processor. Use the access time of the RUU to determine the frequency.
   - Compare and discuss the areas (RUU and the L1, L2 caches – data area + tag area from Cacti; and areas from Real Estate Estimator) given by the estimation tool and the areas estimated by the Cacti. In order to do this, you have to convert λ to proper feature size (16 nm, half of the min. size, which is 32nm). Round off to the nearest hundredth.
   - Plot the total area (from Real Estate Estimator) vs. machine widths and discuss about it.
   - Plot the total transistor count (from Real Estate Estimator) vs. machine widths and discuss about it.

III. Project Environment

a. SimpleScalar Simulator and Benchmarks

SimpleScalar (http://www.simplescalar.com) is a suite of several simulators, which simulate the machine at different levels of detail. We will use sim-outorder, the most detailed one. It can simulate, cycle by cycle, a superscalar processor with dynamic scheduling, branch prediction, speculative execution, caches, etc.

The SimpleScalar simulator has been installed on the virtual machine we provided. The location of the simulator is **'/opt/SimpleScalar/simplesim-3.0/'**. The user's guide for the simulator can be obtained from the den class website.

We have setup the path so that you can run SimpleScalar from your personal directory. A configuration file can be generated by running "sim-outorder -dumpconfig <file>" (replace <file> with your own configuration file name.)

We will be using some benchmark program to test the performance of our machine. The two benchmarks we are using is a perl test scrip and a compression program. To run these programs in Simplescalar, first download the input files: **perl-tests.pl** and **compress95.in** and put them in the **~/Desktop/p2p3** folder. We provide download link for **input.zip** on OneDrive. Then run these benchmarks in Simplescalar from the **p2p3** directory:

```
1. sim-outorder -config test.config -redir:sim perl.out spec95-
   little/perl.ss perl-tests.pl
2. sim-outorder -config test.config -redir:sim compress.out spec95-
   little/compress95.ss < compress95.in
```

As detailed simulation is slow with many instructions, we will limit the number of instructions in these two benchmarks and skip some initial instructions, which can be done by modifying the **max:inst** and **fastfwd** in test.config. For **perl.ss**, set

-max:inst 15000000
-fastfwd 5000000

For **compress95.ss**, set

-max:inst 50000000
-fastfwd 5000000

For students using CARC, please download the file **simplesim-3v0e.tgz** from OneDrive and upload to your discovery account. After download, please follow the commands below.

```
1. tar xzvf simplesim-3v0e.tgz
2. cd simplesim-3.0
3. make config-pisa (to build SimpleScalar/PISA)
4. make
5. make sim-tests
6. export PATH=<Your path of simplesim3.0>/simplesim-3.0:$PATH
```

Note that the last command add the sim-outorder command to your PATH. You may find the executable binary **sim-outorder** under the folder **simplesim-3.0**. You can now dumpconfig and run the simulator.

The other related files are included in the **project2.tar.gz**. Please download **project2.tar.gz** from the OneDrive. Then untar it using following command.

```
1. tar xzvf project2.tar.gz
```

b. Cacti Simulator

As HP doesn't service the web version, we installed the Cacti 6.5 in **~/Desktop/p2p3/cacti65**. Please go to this folder to simulate the cache performance of our system.

Configure the **cache.cfg** as follows:

Parameter	Setting for Cache	Setting for RUU
Cache size (-size) [1]	The cache size	Total RUU bits from Real Estimator[3]
Line size (-block size) [2]	The cache block size	RUU Entry bits from Real Estimator[3]
Associativity	The cache associativity	1
Read-write port	0	0
Exclusive read port	From Real Estimator	From Real Estimator
Exclusive write port	From Real Estimator	From Real Estimator
Single ended read ports	0	0
Cache type	"cache"	"ram"
Leave all the other parameters with the default values. [1]Minimum cache size is 512 bytes; set as 512 for anything less than 512. [2]Minimum line size is 32 bytes; set as 32 for anything less than 32. [3]Translate bits to bytes and round up or down to the **closest bytes**. ***Cache/RUU delay = cycle time, rounded off to the nearest hundredth.** ****Use "Data array" for RUU; "Data array" plus "Tag array" for caches, rounded off to the nearest hundredth.**		

Documentation about Cacti6 is available in the project references on the den class website.

For CARC students, you may find the cacti tool does not work on discovery cluster since it is compiled under a ubuntu OS. You need to re-compile the cacti tool using the singularity container on CARC. Here is a guide of how to build and use the container. https://carc.usc.edu/user-information/user-guides/software-and-programming/singularity.

For your convenience, we provided you a container image which already installed and compiled the Cacti 6.5 for you. Please follow the commands below:

```
1. singularity pull –arch amd64 cacti.sif
   library://superyou/default/image:latest
2. singularity shell cacti.sif
3. cd /opt/ee557project2/
4. ./cacti -infile <the direct path to your cache.cfg>
```

These commands will first pull a container image on your cluster named as **cacti.sif**. Then you will use "*singularity shell*" command to get into the container. The cacti tool is installed at the location of ***/opt/ee557project2/*** inside the container only. You have to go into that folder inside of singularity to use the executable binary cacti. You can modify your own cache.cfg file outside of the container or inside container using vim. Remember you need to include the direct path of your cache.cfg as an input when using the cacti tool.

Note: you may not be able to modify the cache.cfg under the folder ***/opt/ee557project2/*** because it is read only. You can create the cache config file anywhere outside the container and set the direct path to run the tool.

c. Transistor Count and On-chip Real Estate Estimator

The hardware complexity estimation tool can estimate the transistor count and chip space based on the SimpleScalar configuration parameters. The tool is implemented as an MS Excel spreadsheet, '**estimator_v2.xls**' that is provided on OneDrive. You can map sim-outorder's configuration into this tool by setting parameters on the 1st sheet called 'configFileParams' as the parameters in the sim-outorder configuration file, and estimate the silicon area in terms of λ (which is equivalent to a half of the minimum feature size) of a processor. You can also estimate the transistor count of a processor.

The estimation tool and its documentation are available on the den class website.

d. Simulator (sim-outorder) Configuration

We consider a baseline processor with the following configuration parameters (use default value for any unspecified parameters). **Do not change the other parameters!**:

4-entry instruction fetch queue
2-level GAp Branch predictor with***** 8-bit history and 4 lower bits of PC
256-entry, 2-way associative BTB
4-entry return address stack
4 instruction maximum issued per cycle
4 instruction maximum decoded per cycle
4 instruction maximum committed per cycle
8-entry register update unit
16-entry load/store queue
1 integer ALU units
1 integer multiplier/divider unit
1 floating point ALU units
1 floating point multiplier/divider unit
64-byte and 128-byte cache block sizes for L1 and L2 caches respectively******
8KB 2-way associative, L1 instruction cache, FIFO replacement
8KB 4-way associative, L1 data cache, LRU replacement
512KB 8-way associative, unified L2 cache, LRU replacement
2 memory ports
Main memory latency is 100 cycles for the first chunk and 20 cycle for the rest
100 cycles TLB miss latency

***** To figure out the parameters, refer to Figure 6 and Table 2 in users_guide_v2.pdf.

****** Refer to p.27-29 in hack_guide.pdf for details.

However, you have to estimate latencies for the caches with respect to the latency of RUU using Cacti:

L1 cache latency =
L2 cache latency =

IV. Project Submission

You must submit the followings on the den dropbox by the due date. You should combine 2 – 6 below in one pdf file, but 1 (your configuration file) should be submitted in a plain text file as being used on the simulator (**this file will be used for verification**). You should submit a .zip file containing both the pdf file and the plain text configuration file. The .zip file must be named as LastName_FirstName_Proj2.zip

1. Your baseline (machine width = 4) sim-outorder configuration file **with all the cache latencies** (LastnameFirstNameProj2.conf); outputs from running this configuration file must match the outputs in your report.
2. Estimation results given by the Cacti with corresponding inputs.

3. Explanation of how you converted the cache latencies.
4. Performance evaluation obtained from Basic Project step 8 above, and relevant discussion of the superscalar dynamically scheduled processors with different machine widths.
5. Comparison and discussion of the area (only RUU, and the L1, L2 caches) estimates generated by two different tools, Cacti and Real Estimator,
6. Plot and discuss how the total silicon area (from Real Estate Estimator) and the total transistor counts (from Real Estate Estimator) change along with the different machine widths.

V. Grading

1. Setting configuration file parameters: 0.5 pts.
2. L1-I, L1-D and L2 caches access times from Cacti, areas from Cacti and areas from Real Estimator in a table: 0.5 pt.
3. RUU access time from Cacti, area from Cacti and area from Real Estimator for the machine width, 1, 2, 4 and 8 in a table: 0.5 pt.
4. Description of how the clock cycle time, L1-I latency, L1-D latency and L2 latency were obtained for the machine width 4: 1pt
5. MIPS rates for the two applications for the different machine widths in a table and in a graph: 1 pt.
6. Total transistor counts and areas for the different machine widths in a table and in a graph: 1 pt
7. Explanations as described in section IV above: 0.5 pt.

As other assignments, this project must be done INDIVIDUALLY.

FAQ

Q: What is the cache latency calculation process?

A: For RUU, you use the "Access time (ns): " output by the cacti as the cycle time of our simulated machine. In the cacti output, "Access time (ns): " will be the same as "Data side (with Output driver) (ns): "

For Caches (L1D, L1I and L2), you use "Data side (with Output driver) (ns): " + "Tag side (with Output driver) (ns): " from cacti as the access time of caches (in nanoseconds).

Then you calcuate the latency of caches in cycles by using the ceiling of (cache access time / the cycle time).

Do *NOT* use the "Cycle time (ns):" from cacti in any circumstances.

Q: How to calculate the MIPS?

A: MIPS = **sim_num_insn** from sim-outorder output in millions (=50) / (**sim_cycle** from sim-outorder output * **ruu access time** in nanoseconds from cacti / 10^9)

Q: Some missing ConfigFileParams in the estimator tool?

A: The **BTB associativity**, **commit width**, **return address stack (RAS)** are not in the estimator tool excel file. Just ignore them when you input the simulated machine configurations.