****

**DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING**

**EEL5741-RXPE-1201: Advanced Microprocessor Systems**

**SPRING 2020**

**Benchmark Profiling and Processor Architecture**

**Simulation Project 1**

**EEL5741\_4**

Elvis Fernandez 5866280 efern207@fiu.edu

Kevin Lopez 4054391 klope108@fiu.edu

Narek Vanestyan 6227578 nvane007@fiu.edu

**Submitted to:**

**Professor: Dr. Gustavo Chaparro**

**February 16th, 2020**

## Table of Contents

[**Table of Contents**](#_4ye3bfgeecf8) **2**

[**Introduction**](#_nzo305eee1m0) **3**

[Part I](#_4a5iwwl8qi8r) 3

[Getting Set Up](#_k10d1pd3zvmk) 3

[Quick Notes on SimpleScalar](#_2mxsfvulth3s) 4

[SimpleScalar tool set](#_7wfkypiymxrx) 4

[Instruction sets that can be emulated](#_cm6xk511u7fi) 4

[Platforms](#_uqgd7jns7vtq) 4

[Working Setup (SimpleScalar commands)](#_987v4giiwhud) 5

[Part II](#_g9nx7n3hi61p) 9

[Request:](#_rcjhba8y3hbm) 9

[Answer:](#_qad8wa6vzvhz) 9

[Part III](#_sdkrnb8frfnc) 15

[Request:](#_167lupq4ozf1) 15

[Answer:](#_l02muc33kxes) 15

[Conclusion:](#_s2snf3bk8kmo) 17

[Part IV](#_iffrnwblmro9) 18

[Request:](#_rr2cqho44prv) 18

[Answer:](#_ieo4o42l3kft) 19

[Part a)](#_qr3lrplpw36y) 19

[Part b)](#_wlk2e77zz4m2) 20

[Summary](#_7d5v0romfzn6) 21

## Introduction

The goal of this simulation project is to use the SimpleScalar simulator to profile the executions of different benchmark programs from SPEC95 so that we can analyze the performance of different computer systems and to examine the roles of the compiler.

### Part I

Read the SimpleScalar Tool Set Document and understand the SimpleScalar architecture. Download and install SimpleScalar 3.0 software package on your computer. Study the usage of SimpleScalar simulation commands **sim-fast** and **sim- profile**. Please check the references at the end of this document.

##### **Getting Set Up**

In preparation for parts 2, 3, and 4, our team worked on getting the proper setup and reviewed available documentation for SimpleScalar and SPEC95 on the web. The following list shows the actions to get the set up working:

1. Download and Installed VirtualBox form <https://www.virtualbox.org/>
   1. The team found the following link helpful on how to install VirtualBox: <https://www.youtube.com/watch?v=63_kPIQUPp8>
2. Download and Installed Ubuntu 16.04 LT OS for Desktops from: <http://releases.ubuntu.com/16.04/>
   1. The team found the following link helpful on how install Ubuntu in VirtualBox: <https://www.youtube.com/watch?v=VTY_fsY0m0c>
3. Download SimpleScalar files from <http://www.simplescalar.com/>
   1. GitHub:
      1. <https://github.com/sdenel/How-to-install-SimpleScalar-on-Ubuntu>
   2. The team found the following links helpful on how to install SimpleScalar:
      1. <https://www.youtube.com/watch?v=kNJHX7vyKs4&t=448s>
      2. <https://www.youtube.com/watch?v=H1Xhggu4h4c&t=132s>
      3. <https://www.youtube.com/watch?v=VM0iT1URnis>
4. Download and Cross-Compile SPEC95 with SimpleScalar.
   1. Team found the following Github instructions helpful: <https://github.com/priyankarroychowdhury3/Simplescalar>
   2. SPEC95 commands help can be found here: <http://hamblen.ece.gatech.edu/4100/course/simplescalar/Spec95%20Benchmark%20Command%20Lines.htm>

##### **Quick Notes on SimpleScalar**

Information regarding SimpleScalar simulator can be found at its [site](http://www.simplescalar.com/); however, the team extracted the next three sections and it is sharing them here for information.

###### SimpleScalar tool set

The SimpleScalar tool set is a system software infrastructure used to build modeling applications for program performance analysis, detailed microarchitectural modeling, and hardware-software co-verification. Using the SimpleScalar tools, users can build modeling applications that simulate real programs running on a range of modern processors and systems. The tool set includes sample simulators ranging from a fast functional simulator to a detailed, dynamically scheduled processor model that supports non-blocking caches, speculative execution, and state-of-the-art branch prediction. In addition to simulators, the SimpleScalar tool set includes performance visualization tools, statistical analysis resources, and debug and verification infrastructure.

###### Instruction sets that can be emulated

SimpleScalar simulators can emulate the Alpha, PISA, ARM, and x86 instruction sets. The tool set includes a machine definition infrastructure that permits most architectural details to be separated from simulator implementations. All of the simulators distributed with the current release of SimpleScalar can run programs from any of the above listed instruction sets. Complex instruction set emulation (e.g., x86) can be implemented with or without microcode, making the SimpleScalar tools particularly useful for modeling CISC instruction sets.

The PISA instruction set (a.k.a. the portable instruction set architecture) is a simple MIPS-like instruction set maintained primarily for instructional use. A GNU GCC-based cross-compiler and pre-built libraries are also available for this target. The PISA target is particularly useful for computer engineering instruction as the tools can be built on a wide range of host platforms, including Linux/x86, Win2000, SPARC Solaris, and others.

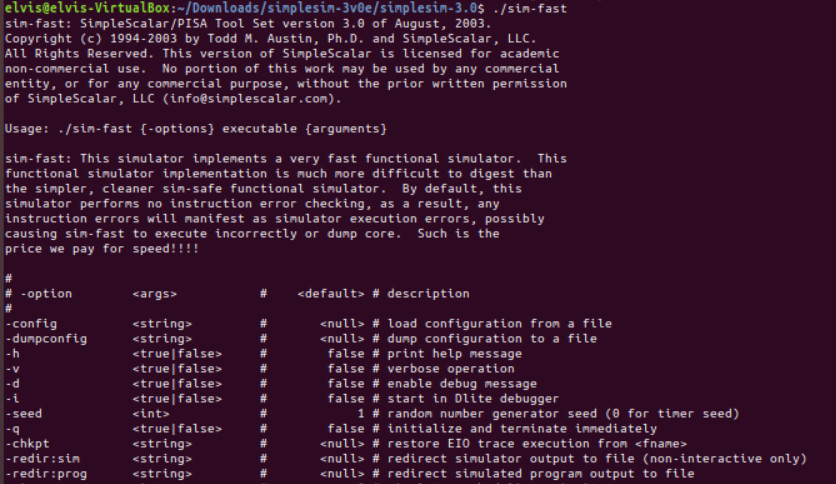
###### Platforms

SimpleScalar builds on most 32-bit and 64-bit flavors of UNIX and Windows NT-based operating systems. The internal software architecture of the tool set includes a host interface module, permitting fast porting to other host platforms. The host interface module permits cross-endian emulation, thus it is possible to use emulate a target on a host platform with a different endian, *e.g.*, running Alpha ISA emulation on a SPARC Solaris host platform. Most SimpleScalar users and developers (including SimpleScalar LLC) use SimpleScalar on Linux/x86.

##### **Working Setup (SimpleScalar commands)**

Figure 1 and 2 illustrate sim-fast and sim-profile command options.

***./sim-fast***

******

**Fig 1. SimpleScalar ./sim-fast <options>**

*Usage: ./sim-fast {-options} executable {arguments}*

*sim-fast: This simulator implements a very fast functional simulator. This*

*functional simulator implementation is much more difficult to digest than*

*the simpler, cleaner sim-safe functional simulator. By default, this*

*simulator performs no instruction error checking, as a result, any*

*instruction errors will manifest as simulator execution errors, possibly*

*causing sim-fast to execute incorrectly or dump core. Such is the*

*price we pay for speed!!!!*

*#*

*# -option <args> # <default> # description*

*#*

*-config <string> # <null> # load configuration from a file*

*-dumpconfig <string> # <null> # dump configuration to a file*

*-h <true|false> # false # print help message*

*-v <true|false> # false # verbose operation*

*-d <true|false> # false # enable debug message*

*-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*

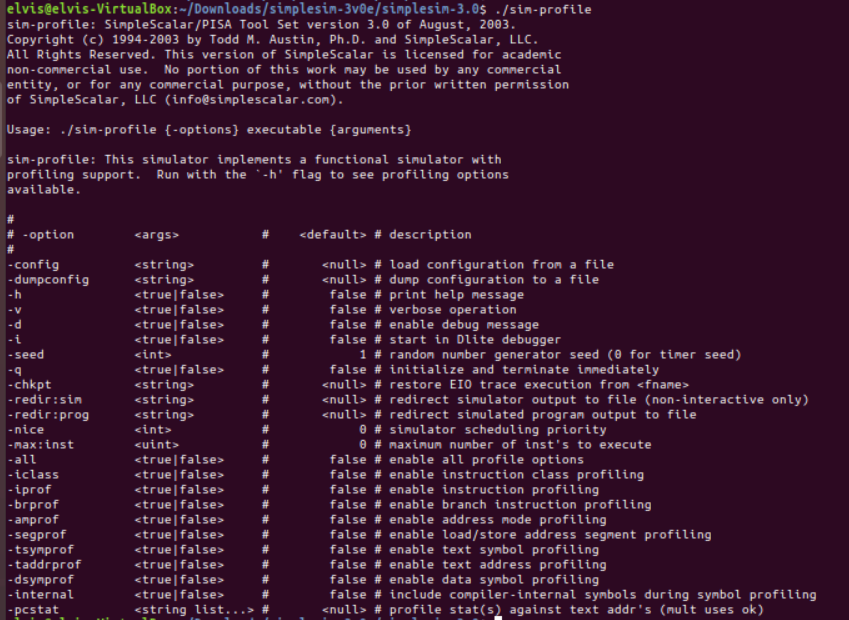
*-chkpt <string> # <null> # restore EIO trace execution from <fname>*

*-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*

***./sim-profile***

**

**Fig 2. SimpleScalar ./sim-profile <options>**

*Usage: ./sim-profile {-options} executable {arguments}*

*sim-profile: This simulator implements a functional simulator with*

*profiling support. Run with the `-h' flag to see profiling options*

*available.*

*#*

*# -option <args> # <default> # description*

*#*

*-config <string> # <null> # load configuration from a file*

*-dumpconfig <string> # <null> # dump configuration to a file*

*-h <true|false> # false # print help message*

*-v <true|false> # false # verbose operation*

*-d <true|false> # false # enable debug message*

*-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 <fname>*

*-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 inst's to execute*

*-all <true|false> # false # enable all profile options*

*-iclass <true|false> # false # enable instruction class profiling*

*-iprof <true|false> # false # enable instruction profiling*

*-brprof <true|false> # false # enable branch instruction profiling*

*-amprof <true|false> # false # enable address mode profiling*

*-segprof <true|false> # false # enable load/store address segment profiling*

*-tsymprof <true|false> # false # enable text symbol profiling*

*-taddrprof <true|false> # false # enable text address profiling*

*-dsymprof <true|false> # false # enable data symbol profiling*

*-internal <true|false> # false # include compiler-internal symbols during symbol profiling*

*-pcstat <string list...> # <null> # profile stat(s) against text addr's (mult uses ok*

### 

### Part II

#### **Request:**

For each of the following five precompiled benchmarks, namely, ***go.ss, anagram.ss,***

***apsi.ss, cc1.ss, and compress95.ss***. Check the outputs. The command lines to use these files are listed as follows:

a) /simplesim-3.0/ ./sim-profile -iclass BenchMarks\_Little/go.ss 50 9 2stone9.in

b) /simplesim-3.0/ ./sim-profile -iclass BenchMarks\_Little/anagram.ss words <BenchMarks\_Little/anagram.in

c) /simplesim-3.0/ ./sim-profile -iclass BenchMarks\_Little/apsi.ss

d) /simplesim-3.0/ ./sim-profile -iclass BenchMarks\_Little/cc1.ss -O 1smt.i

e) /simplesim-3.0/ ./sim-profile -iclass BenchMarks\_Little/compress95.ss <BenchMarks\_Little/compress95.in

#### **Answer:**

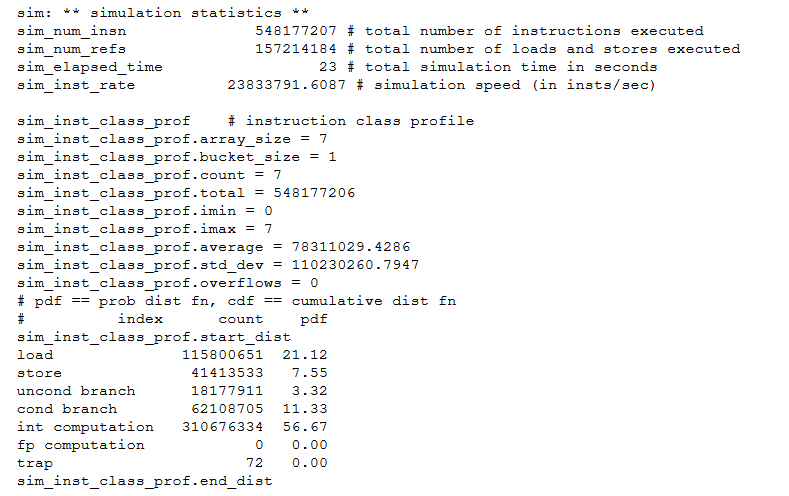
***a) ./sim-profile -iclass BenchMarks\_Little/go.ss 50 9 BenchMarks\_Little/2stone9.in***

Where

* go.ss is the SimpleScalar binary for the "go" benchmark that simulates a play of the board game “Go.”
* "50 9" are the play quality and the board size for the go simulation.
* "2stone9.in" is a file that specifies the starting board position.

Link to output file:

<https://drive.google.com/drive/u/0/folders/1DlL9yW3MA18Sj552n-kn9tPbOzS1ul8E>



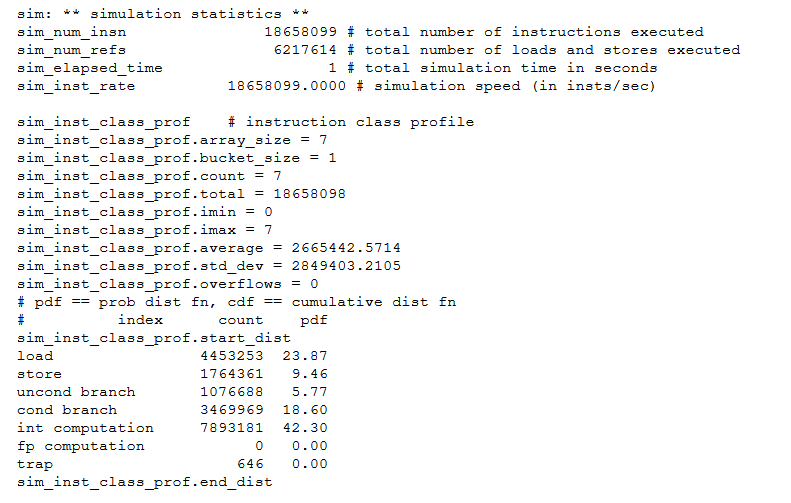
**Fig.3 Simulation Statics form go.ss**

***b) ./simplesim-3.0$ ./sim-profile -iclass BenchMarks\_Little/anagram.ss words < BenchMarks\_Little/anagram.in***

Where:

* anagram.ss is the SimpleScalar binary for the "anagram" benchmark. A program for finding anagrams for a phrase, based on a dictionary
* “words” is the word used by the program to create the anagrams
* anagram.in is the dictionary fed to the anagram.ss benchmark

Link to output file: <https://drive.google.com/drive/u/0/folders/1DlL9yW3MA18Sj552n-kn9tPbOzS1ul8E>



**Fig.4 Simulation statics from anagram.ss**

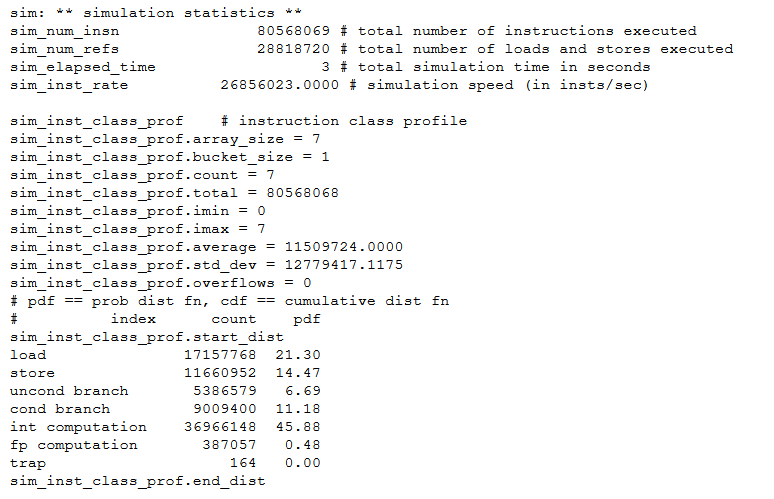
***c)./sim-profile -iclass spec95-little/apsi.ss***

Where:

* apsi.ss is the SimpleScalar binary for the “apsi” benchmark. It reads a 112x112x112 area array of data and iterates over 70 timesteps.
* Program to solve for the mesoscale and synoptic variations of potential temperature, U AND V wind components, and the mesoscale vertical velocity W pressure and distribution of pollutants C having sources Q. The synoptic scale components are in quasi-steady state balance, while the mesoscale pressure and velocity W are found diagnostically.
* The outcome prints the diagonal elements of the velocity field.

Link to output file:

<https://drive.google.com/drive/u/0/folders/1DlL9yW3MA18Sj552n-kn9tPbOzS1ul8E>



**Fig.5 Simulation statistics from apsi.ss**

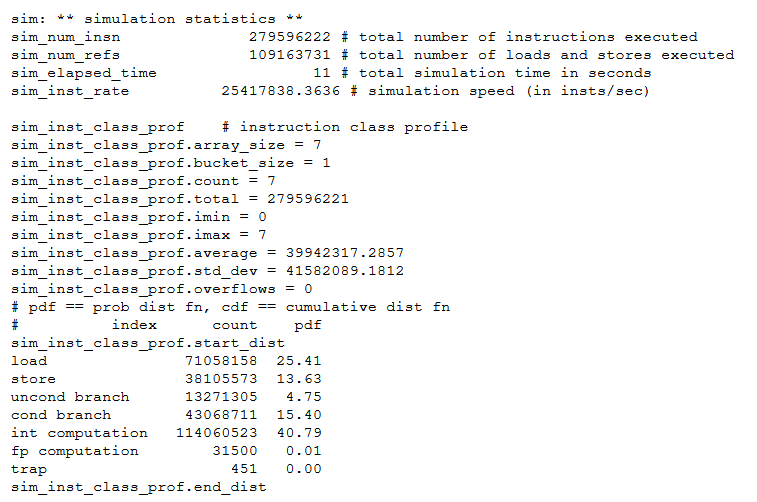
***d) ./sim-profile -iclass BenchMarks\_Little/cc1.ss -O BenchMarks\_Little/Input/1stmt.i***

Where:

* cc1.ss is the SimpleScalar binary for the “cc1” benchmark. It is a limited version of gcc. It compiles an already pre-processed input file, producing an assembly file as output.
* The parameter “-o 1stmt.i” causes the compiler to write its results to the 1stmt.i file.

Link to output file:

<https://drive.google.com/drive/u/0/folders/1DlL9yW3MA18Sj552n-kn9tPbOzS1ul8E>



**Fig.6 Simulation statistics from cc1.ss**

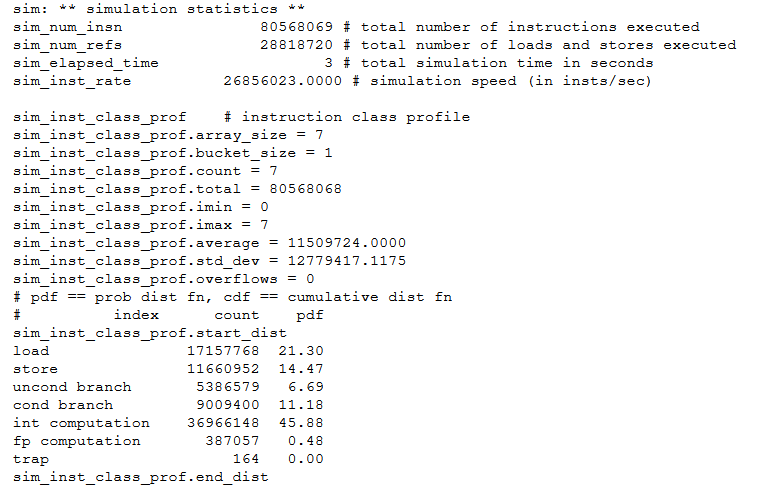
***e) ./sim-profile -iclass BenchMarks\_Little/compress95.ss < BenchMarks\_Little/Input/compress95.in***

Where:

* compress95.ss the SimpleScalar binary for the "compress" benchmark. It compresses and decompresses a file in memory
* compress95.in is the input file for the benchmark

Link to output file:

<https://drive.google.com/drive/u/0/folders/1DlL9yW3MA18Sj552n-kn9tPbOzS1ul8E>



**Fig.7 Simulation statistics from compress95.ss**

### 

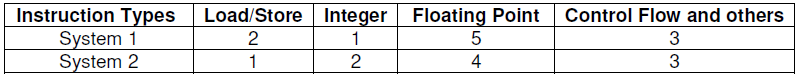
### Part III

#### **Request:**

Assuming you have the following two machines with the same cycle time but different

cycles-per-instruction (CPI) for different instruction groups as follows:

Table 1. CPIs for different types of instructions



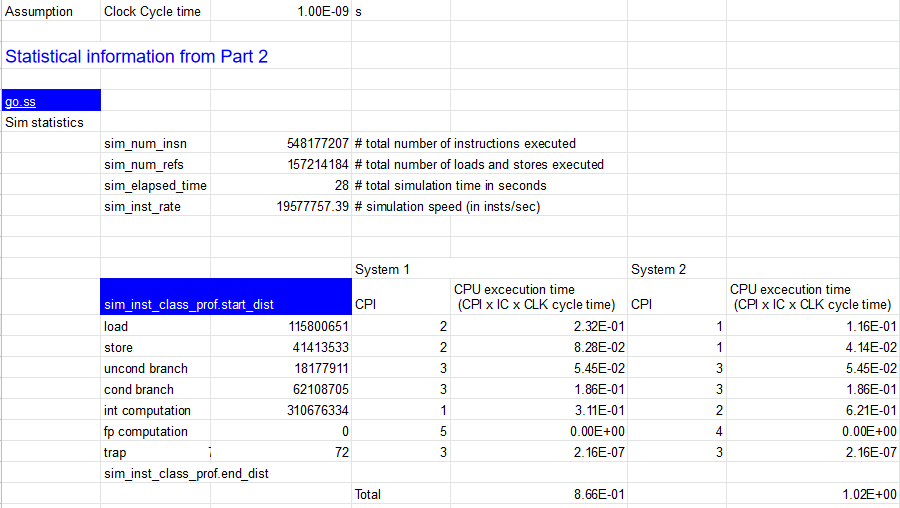
First, let System 1 be the reference machine. With the data you can collect from step 2, compare the performance of these two systems normalized geometric mean. Use System 2 as the reference machine and redo the comparison. Discuss your results.

#### **Answer:**

Link to analysis

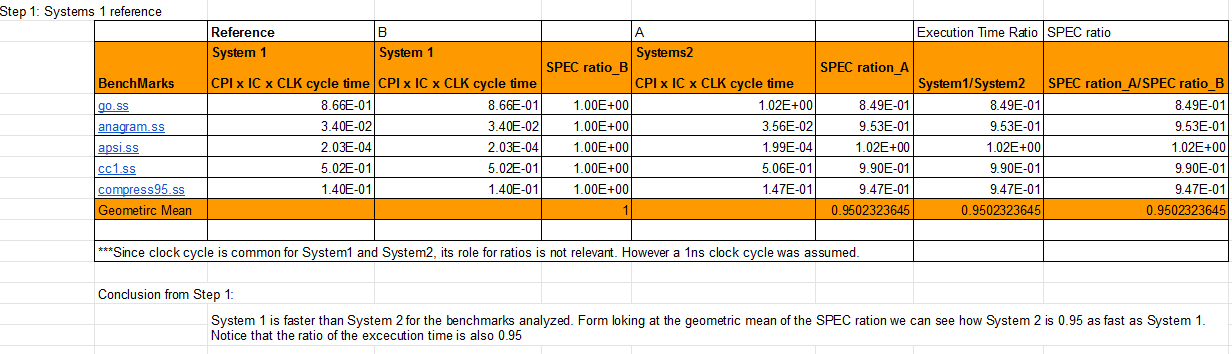
<https://docs.google.com/spreadsheets/d/1c5TB2TsFCzocje3jEPGeXkVJaHiLVlndVtpx3LDaonU/edit#gid=2019272544>

Statistical information was gathered from part 2 for all 5 programs. CPU execution time was calculated for Systems 1 and 2 (eg. for the go.ss - figure 8)



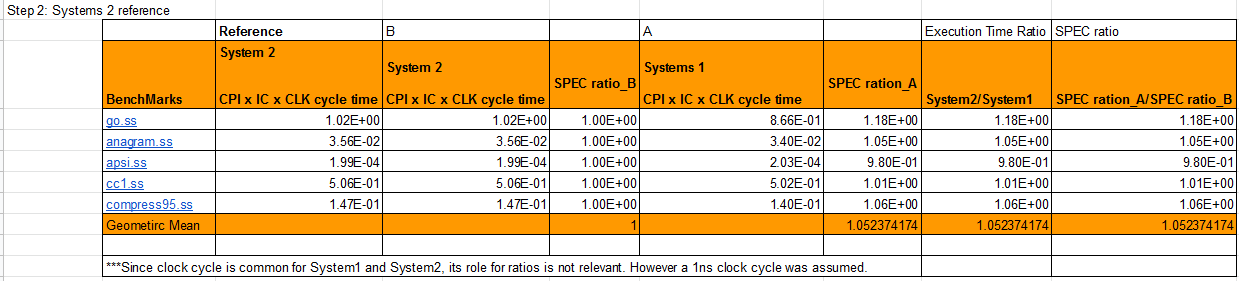
**Fig.8 CPU execution time for System 1 and 2 from go.ss program.**

System 1 was selected as reference for benchmarking both systems using the five programs.



**Fig.9 System 1 as reference.**

Secondly, System 2 was selected as reference



**Fig.10System 2 as reference..**

#### **Conclusion:**

**After swapping System 1 & 2 and analysing the results we proved that System 1 was faster than System 2 regardless of the reference system chosen. This proves that the performance of the reference system is irrelevant, as long as we know how each system performs with respect to the reference system. This will allow us to compare System 1 and System 2 together because they are both benchmarked with respect to the same “reference.” Also, we validated that the geometric means of the ratios were the same as the ratio of the geometric means.**

### 

### Part IV

#### **Request:**

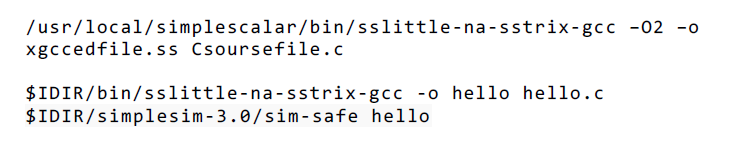
For this part use the C program for matrix multiplication, i.e., matmul.c, that computes

the production of two matrices Amxn x Bnxk. It takes three input parameters, i.e., m, n,

and k, randomly generates two matrices, and then computes their product. Compile it

with SimpleScalar compiler (i.e., /usr/local/simplescalar/bin/sslittle-na-sstrix-gcc)

with no optimization and with level 2 optimization. Here is an example:



The above command compiles C source code Csoursefile.c to the SimpleScalar binary

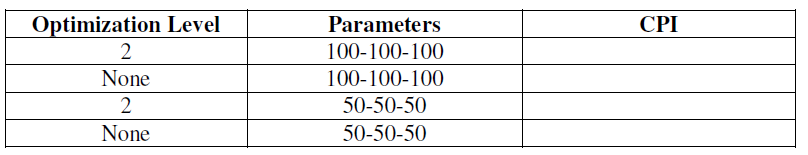
file xgccedfile.ss with the level 2 optimization option. Note that the SimpleScalar compiler has a similar input format as that of gcc. (If you are not familiar with gcc,

you should check the details using the UNIX/Linux manual command, i.e., man gcc.)

a) Let m=n=k=50, simulate the binary files and compare the total instruction counts.

b) What are the overall CPIs of the computer when running the optimized and non-

optimized code? Discuss your results.



#### 

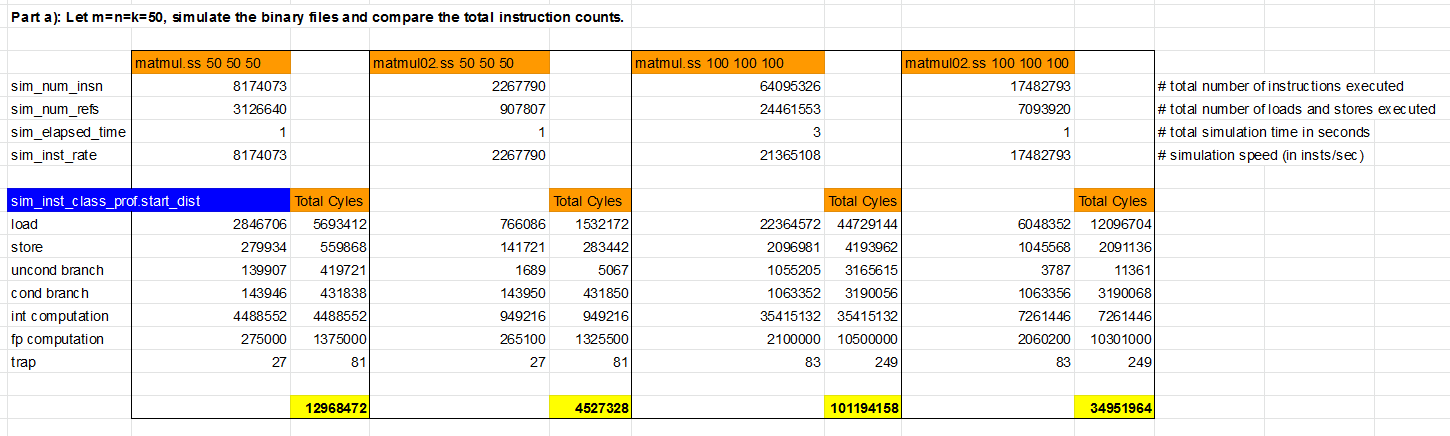
#### **Answer:**

To get Part 4 to work Team followed the below steps:

1. Download matmul.c file and save it into the simplesim-3.0 dir
2. You need to cross compile matmul.c with SimpleScalar to generate optimized and non-optimized SimpleScalar binary.ss files.
   1. For Optimized matmu02.ss file:
      1. elvis@elvis-VirtualBox:~/Downloads/How-to-install-SimpleScalar-on-Ubuntu-master/build/simplesim-3.0$ sslittle-na-sstrix-gcc -O2 -x c matmul.c -o matmul02.ss
         1. (A matmul02.ss file will be created at the root dir)
   2. Non-optimized matmul.ss file:
      1. elvis@elvis-VirtualBox:~/Downloads/How-to-install-SimpleScalar-on-Ubuntu-master/build/simplesim-3.0$ sslittle-na-sstrix-gcc -x c matmul.c -o matmul.ss
         1. (A matmul.ss file will be created at the root dir)
3. Now, you can use SimpleScalar tools, such as sim-profile, sim-fast, etc..
   1. For Optimized matmul02.ss file:
      1. elvis@elvis-VirtualBox:~/Downloads/How-to-install-SimpleScalar-on-Ubuntu-master/build/simplesim-3.0$ ./sim-profile -iclass matmul02.ss 50 50 50
   2. Non-optimized matmul.ss file:
      1. Same as part “a” with the matmul.ss file

##### Part a)

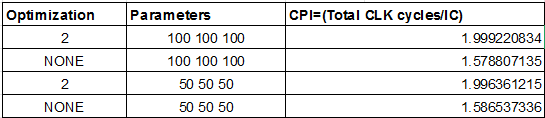
Once the matmul.ss files were complied, the team calculated the Cycle times for the 4 cases (See figure 11)



**Fig.11 Calculation of Total Clock cycles**

##### Part b)

Once we calculated the total cycle time for all 4 cases, we calculated the CPI for optimized and unoptimized programs (see figure 12)



**Fig.12 Calculated CPU for 4 cases.**

Form figure 11 and 12 we can observe how the optimized programs reduce the total number instructions, which is reflected in the reduction of the CPI magnitud.

Link to Analysis:

<https://docs.google.com/spreadsheets/d/1Qn9GsPyR5qsCM7ZQEaMCFnzLTYhnvmrwJ9NI8xH6k8w/edit#gid=0>

### Summary

We were able to run all the correct commands after some research and were able to understand what each command did. We obtained the correct outputs and were able to use the numbers in the required sections to run some calculations on the benchmark figures.

We observed how different systems can perform under the same benchmarks and that changing the options and inputs to the benchmarks could yield very different results.

Aside from benchmarks, compilers and their optimization levels were also something we were able to see in action; the outputs were generally in line with our expectations. Higher optimization levels produced fewer total number of instructions during compilation, but the CPI could be higher as the compiler has to put in more effort in order to optimize the .c file during optimization.

It is worth noting that our team learned quite a bit about running terminal commands in Linux, as well as setting up a VM. We all had some prior experience, but it was quite limited, and now we are able to run benchmarks without any GUI.