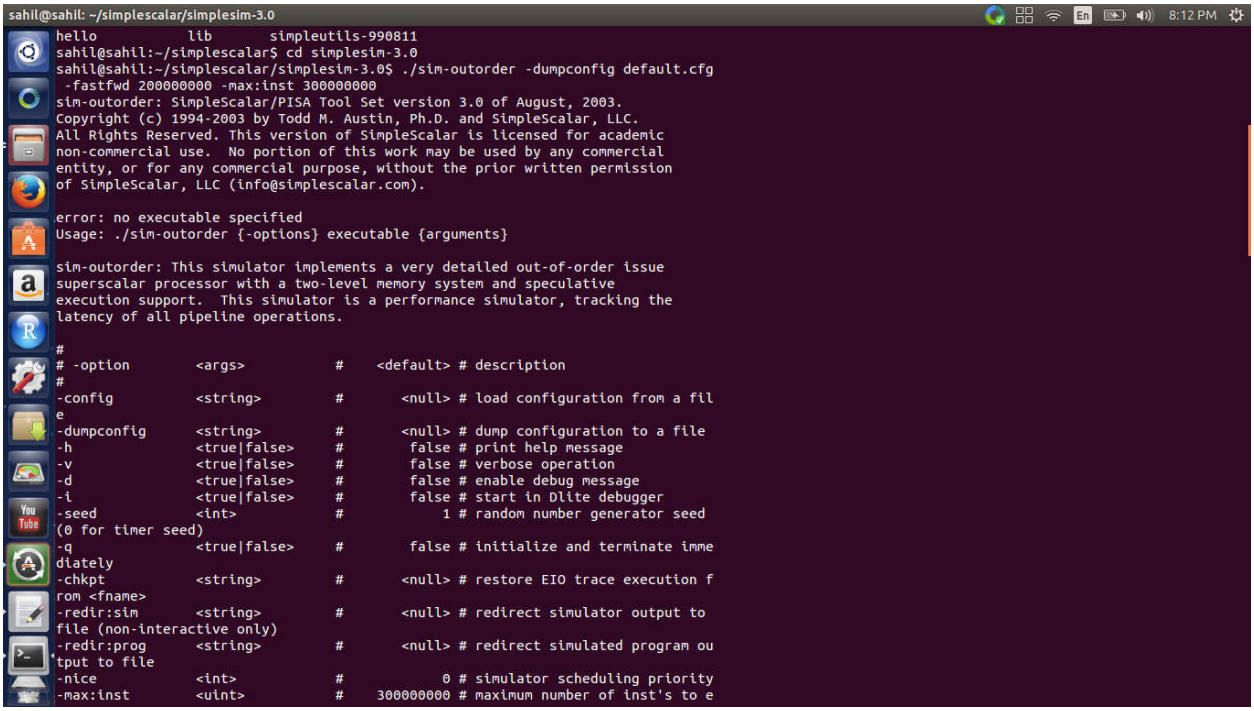
Sahil Shetye
UIN: 673274841

ECE 466 Advance Computer Architecture
Project 1

1. What is the performance of running the program, equake, under the default system setup (without changing any simulation parameters) using command: ./sim-outorder equake.ss < equake.in?

To run the Simplescalar software on a Windows machine, I have used Linux to carry out all the executions. To simplify things, I have created a configuration file which can be later modified as per the needs. The configuration file was created by –

./sim-outorder -dumpconfig default.cfg -fastfwd 200000000 -max:inst 300000000



```
sahil@sahil: ~/simplescalar/simplesim-3.0
hello           lib       simpleutils-990811
sahil@sahil:~/simplescalar$ cd simplesim-3.0
sahil@sahil:~/simplescalar/simplesim-3.0$ ./sim-outorder -dumpconfig default.cfg
 -fastfwd 200000000 -max:inst 300000000
sim-outorder: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.
All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use.  No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).

error: no executable specified
Usage: ./sim-outorder {-options} executable {arguments}

sim-outorder: This simulator implements a very detailed out-of-order issue
superscalar processor with a two-level memory system and speculative
execution support.  This simulator is a performance simulator, tracking the
latency of all pipeline operations.

#
# -option        <args>          #    <default> # description
#
-config          <string>        #       <null> # load configuration from a fil
e
-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 imme
diately
-chkpt           <string>        #       <null> # restore EIO trace execution f
rom <fname>
-redir:sim       <string>        #       <null> # redirect simulator output to
file (non-interactive only)
-redir:prog      <string>        #       <null> # redirect simulated program ou
tput to file
-nice            <int>           #            0 # simulator scheduling priority
-max:inst        <uint>          #    300000000 # maximum number of inst's to e
```

Figure 1: Creating configuration file name my.cfg

Here, the configuration file is saved under the name 'default' is saved under the same folder. Configuration file can be access and modified with any standard text editor and is very convenient to implement simulations by setting parameters under this configuration file. Instruction -fastfwd 200000000 is used to fast forward 500M instructions and max:inst: 300000000 is used to execute the next 300M instructions as per the first requirement of the question. The program can be then run by the following command –

./sim-outorder -config my.cfg equake.ss<equake.in

```
sim: ** simulation statistics **
sim_num_insn              300000001 # total number of instructions committed
sim_num_refs               98046368 # total number of loads and stores committed
sim_num_loads              68339475 # total number of loads committed
sim_num_stores         29706893.0000 # total number of stores committed
sim_num_branches           78914237 # total number of branches committed
sim_elapsed_time                431 # total simulation time in seconds
sim_inst_rate           696055.6868 # simulation speed (in insts/sec)
sim_IPC                       1.6747 # instructions per cycle
sim_CPI                       0.5971 # cycles per instruction
```

Above is a brief part of the simulation results and we can observe several important results like number of instructions, elapsed time and most importantly, the IPC(Instructions per cycle) and CPI(Cycles per Instructions) values.

```
sim_IPC                          1.6747 # instructions per cycle
sim_CPI                          0.5971 # cycles per instruction
```

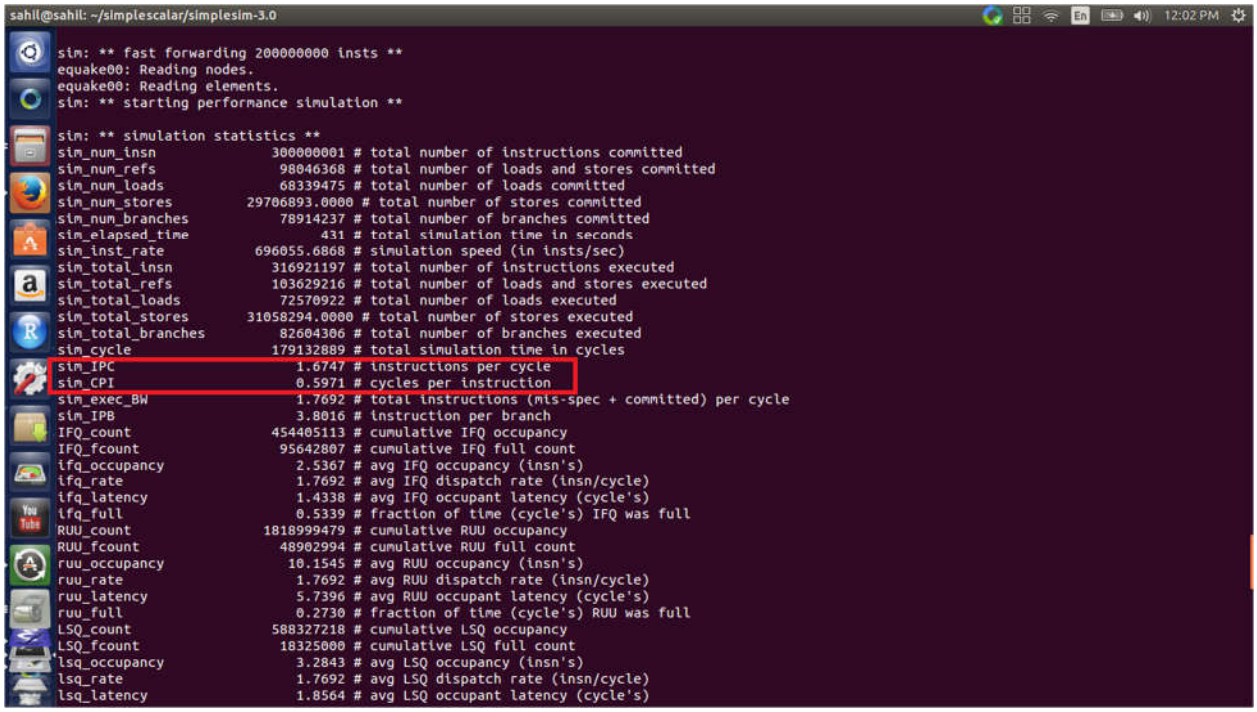


Figure 2: using configuration file for finding out-of-order execution

2. How much is the performance loss if the processor uses in-order execution instead of the default out-of-order execution for running the program?

We have created the configuration file, it makes our job much simple by changing the default configuration as per the needs. Here, we need to execute the program with in-order execution as compared to the out-of-order. This can be done by changing the following line in the configuration file –

 # run pipeline with in-order issue
-issue:inorder true

When the **-issue:inorder** is set to true, in-order execution takes places. By default, it is false, for out-of-order execution. I create a config file called inorder.cfg to implement question 2

And we can re-run the simulation by the same command as above -
./sim-outorder -config inorder.cfg equake.ss<equake.in

```
sim: ** simulation statistics **
sim_num_insn                300000000 # total number of instructions committed
sim_num_refs                 97642663 # total number of loads and stores committed
sim_num_loads                67983534 # total number of loads committed
sim_num_stores          29659129.0000 # total number of stores committed
sim_num_branches             79161917 # total number of branches committed
sim_elapsed_time                  158 # total simulation time in seconds
sim_inst_rate            1898734.1772 # simulation speed (in insts/sec)
```

```
s
sim_IPC                          0.7756 # instructions per cycle
sim_CPI                          1.2894 # cycles per instruction
```

From the above simulation we can notice that the values of IPC and CPI has been changed –

```
sim_IPC                          0.7756 # instructions per cycle
sim_CPI                          1.2894 # cycles per instruction
```

As compared with the default out-of-order execution, we observe 2.159 times execution time or 46.31% execution speed in case of in-order execution.

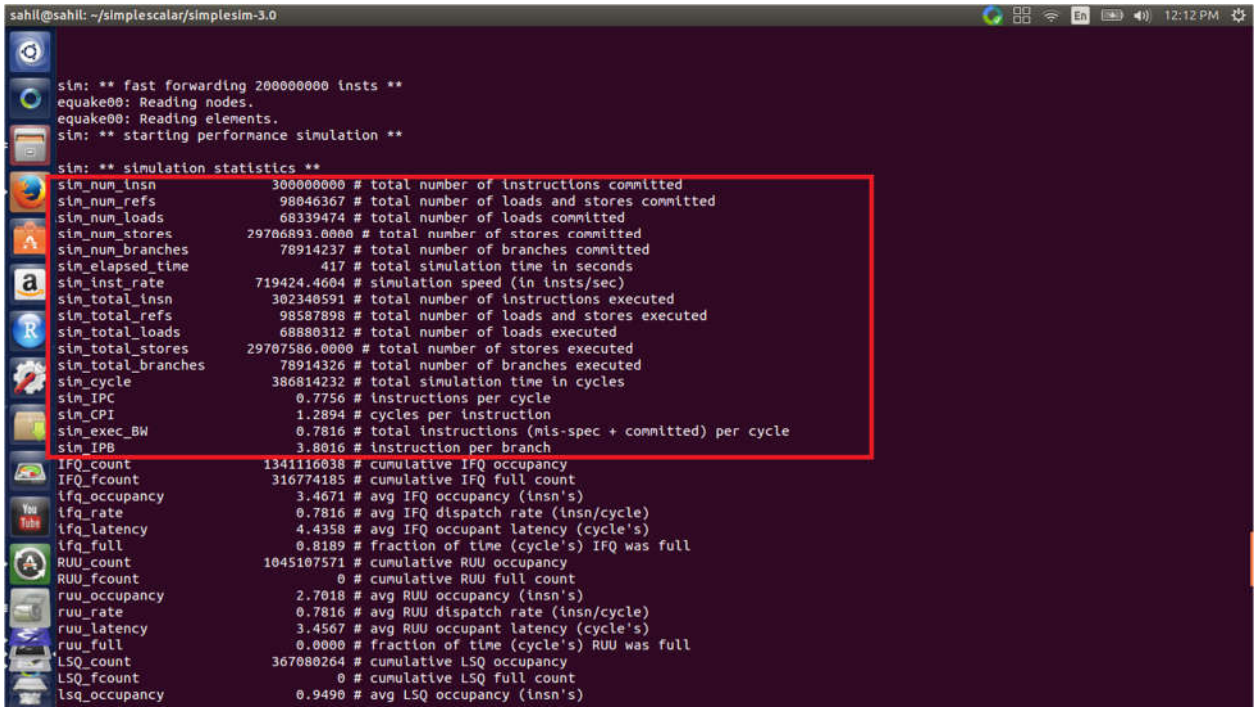


Figure 3: Simulated results for in-order execution

3. The above experiments only perform detailed simulation on 300 million instructions. Based on the simulator running time in Question 1, estimate how long it would take to run the program on a real machine with 3GHz processor and the same IPC value as in Question 1; and then estimate how long it would take to simulate the program's execution in details from beginning to end using the default configuration. Note: Do not run the detailed simulation from beginning to end. It may take days to finish.

The value of IPC from the first question was observed to be **1.6745**

Therefore, the numbers of cycles required for simulation of 300 million instructions are:

Number of Cycles = 300000000/1.6745

= 179157958

And for a 3GHz machine,

Time = No. of cycles X Processor Clock Speed

(Processor Clock Speed = 1/CPU Frequency)

Time = $179579588/3x10^9$

Time = 0.05972 Seconds

With a 3GHz machine, time required would be **0.05972 seconds** or 59.72 msecs to implement 300 million instruction.

So to implement 165 billion instruction we require = 656433162265*0.0601/300000000

To execute the program from the start, we use the command –

./sim-safe equake.ss<equake.in

Here, the observed results were –

```
sim: ** simulation statistics **
sim_num_insn           165643487723 # total number of instructions executed
sim_num_refs            78603314990 # total number of loads and stores executed
sim_elapsed_time               1527 # total simulation time in seconds
sim_inst_rate         108455510.8835 # simulation speed (in insts/sec)
ld_text_base             0x00400000 # program text (code) segment base
ld_text_size                 132784 # program text (code) size in bytes
ld_data_base             0x10000000 # program initialized data segment base
ld_data_size                  16384 # program init'ed `.data' and uninit'ed `.bss'
size in bytes
ld_stack_base            0x7fffc000 # program stack segment base (highest address
in stack)
ld_stack_size                 16384 # program initial stack size
ld_prog_entry            0x00400140 # program entry point (initial PC)
ld_environ_base          0x7fff8000 # program environment base address address
ld_target_big_endian              0 # target executable endian-ness, non-zero if
big endian
mem.page_count                10410 # total number of pages allocated
mem.page_mem                 41640k # total size of memory pages allocated
mem.ptab_misses             3253587 # total first level page table misses
mem.ptab_accesses      906492850614 # total page table accesses
mem.ptab_miss_rate           0.0000 # first level page table miss rate
```

Total number of Instructions were – **16564348772** instructions (~165.6B) and using the value of IPC from the result of the first question, IPC = **1.6745**.

Therefore,

Number of Cycles = Total Instructions/IPC

= 1656448772/1.6745

= 98921989.85

Which is also equal to the Simulation time for the simulator

But, Simulation Time = Number of Instructions / Simulation Speed

= 165643487723 / 696855.6868

= 237701.27 seconds

= 3961.23 minutes

= 66.02 Hours

It would take almost **66 hours** to complete the entire program from start to finish when executed with default configuration.

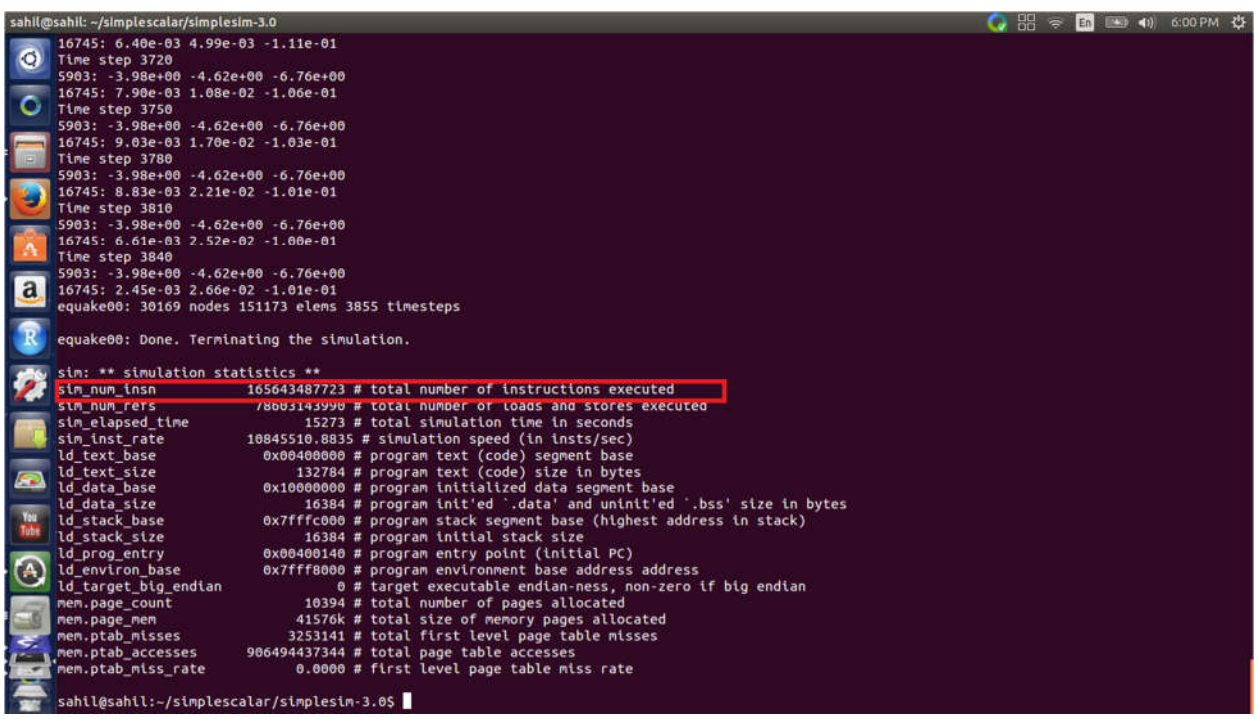


Figure 4: Completion of program

4. An advantage of using simulator is that you can vary the processor parameters to see their performance bottleneck. A widely used approach is to partition the program's execution time or CPI value into three categories: CPU execution (including L1 cache access), L2 cache access and memory access. To achieve this goal, we can run experiments under three configurations: (1) the default configuration, (2) a configuration with the memory latency equal to the L2 cache latency, which basically assumes an infinitely large L2 cache, and (3) a configuration with both the memory and L2 cache latency equal to the L1 cache latency, which basically assumes an infinitely large L1 cache. The performance under these three configurations can be used to partition a program's execution into three categories mentioned above. Report your results and discuss which portion is the performance bottleneck for this program running on the simulated machine.

This part is divided into three part. The first part was implemented in the first question itself. So from the first question we know

1. That default configuration of file has cache l1 latency= 1 cycles, l2 cache latency= 6 cycles and memory access latency of 2 cycles.

On implementing the simulation for above condition, the output IPC obtained is

```
sim_IPC                    1.6747 # instructions per cycle
```

2. The next section involves changing the memory access delay equal to l2 cache delay. For that I make a copy of default config file used in question one and modify it. I change the name to cache.cfg. And modify following parameters in the file by using gedit.
   a. Change L2 lat and mem lat to both 6 and keep L1 lat at 1 cycle and then keep L1 lat at 6 and other L2 and mem at 1.
   b. These updates are implemented by simulating from cfg file.

3. Third part involves again making a copy of cache.cfg file and renaming the new file as cache2.cfg and making following changes in the cache2.cfg file.

   a. Change l1 data, instruction latency same as l2 and memory latency. I try two combinations of latency period by applying 1,3 and 6 cycle latency to each(l1, l2 and memory latency)
      We do this by changing default cycle values in the config file.

For every modification in .cfg file. I implemented the simulation by giving following commands:

./sim-utorder –config cache2.cfg equake.ss

Observed Results:

#	Cycles L1 Lat.	Cycles L2 Lat.	Mem Lat.	IPC	CPI	Comments
1	1	6	18, 2	1.6747	0.5971	Default
2	1	6	6,6	1.6739	0.5974	Inf. L2
3	6	1	1,1	1.18	0.8458	Inf. L2
4	1	1	1,1	2.063	0.4854	Inf. L1
5	6	6	6,6	1.0815	0.9247	Inf. L1
6	3	3	3,3	1.616	0.6187	Inf. L2

Observered O/P:

#1 Same as q1

```
sahil@sahil: ~/simplescalar/simplesim-3.0
sim: ** fast forwarding 200000000 insts **
equake00: Reading nodes.
equake00: Reading elements.
sim: ** starting performance simulation **

sim: ** simulation statistics **
sim_num_insn              300000001 # total number of instructions committed
sim_num_refs               98046368 # total number of loads and stores committed
sim_num_loads              68339475 # total number of loads committed
sim_num_stores         29706893.0000 # total number of stores committed
sim_num_branches           78914237 # total number of branches committed
sim_elapsed_time                426 # total simulation time in seconds
sim_inst_rate           704225.3545 # simulation speed (in insts/sec)
sim_total_insn            316921261 # total number of instructions executed
sim_total_refs            103629216 # total number of loads and stores executed
sim_total_loads            72570922 # total number of loads executed
sim_total_stores       31058294.0000 # total number of stores executed
sim_total_branches         82604306 # total number of branches executed
sim_cycle                 179226473 # total simulation time in cycles
sim_IPC                      1.6739 # instructions per cycle
sim_CPI                      0.5974 # cycles per instruction
sim_exec_BW                  1.7683 # total instructions (mis-spec + committed) per cycle
sim_IPB                      3.8016 # instruction per branch
IFQ_count                 454770553 # cumulative IFQ occupancy
IFQ_fcount                 95734167 # cumulative IFQ full count
ifq_occupancy                2.5374 # avg IFQ occupancy (insn's)
ifq_rate                     1.7683 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                  1.4350 # avg IFQ occupant latency (cycle's)
ifq_full                     0.5342 # fraction of time (cycle's) IFQ was full
RUU_count                1820461927 # cumulative RUU occupancy
RUU_fcount                 48994322 # cumulative RUU full count
ruu_occupancy               10.1573 # avg RUU occupancy (insn's)
ruu_rate                     1.7683 # avg RUU dispatch rate (insn/cycle)
ruu_latency                  5.7442 # avg RUU occupant latency (cycle's)
ruu_full                     0.2734 # fraction of time (cycle's) RUU was full
LSQ_count                 588962514 # cumulative LSQ occupancy
LSQ_fcount                 18325016 # cumulative LSQ full count
lsq_occupancy                3.2861 # avg LSQ occupancy (insn's)
```

Figure 5: Observed results for infinite L2 cache length Result #2

```
sahil@sahil: ~/simplescalar/simplesim-3.0
equake00: Reading elements.
sim: ** starting performance simulation **

sim: ** simulation statistics **
sim_num_insn              300000001 # total number of instructions committed
sim_num_refs               98046368 # total number of loads and stores committed
sim_num_loads              68339475 # total number of loads committed
sim_num_stores         29706893.0000 # total number of stores committed
sim_num_branches           78914237 # total number of branches committed
sim_elapsed_time                479 # total simulation time in seconds
sim_inst_rate           626304.8038 # simulation speed (in insts/sec)
sim_total_insn            320156575 # total number of instructions executed
sim_total_refs            103988202 # total number of loads and stores executed
sim_total_loads            72570862 # total number of loads executed
sim_total_stores       31417340.0000 # total number of stores executed
sim_total_branches         83952748 # total number of branches executed
sim_cycle                 253730211 # total simulation time in cycles
sim_IPC                      1.1824 # instructions per cycle
sim_CPI                      0.8458 # cycles per instruction
sim_exec_BW                  1.2618 # total instructions (mis-spec + committed) per cycle
sim_IPB                      3.8016 # instruction per branch
IFQ_count                 916771002 # cumulative IFQ occupancy
IFQ_fcount                211082514 # cumulative IFQ full count
ifq_occupancy                3.6132 # avg IFQ occupancy (insn's)
ifq_rate                     1.2618 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                  2.8635 # avg IFQ occupant latency (cycle's)
ifq_full                     0.8319 # fraction of time (cycle's) IFQ was full
RUU_count                3653611678 # cumulative RUU occupancy
RUU_fcount                169846013 # cumulative RUU full count
ruu_occupancy               14.3996 # avg RUU occupancy (insn's)
ruu_rate                     1.2618 # avg RUU dispatch rate (insn/cycle)
ruu_latency                 11.4120 # avg RUU occupant latency (cycle's)
ruu_full                     0.6694 # fraction of time (cycle's) RUU was full
LSQ_count                1219500416 # cumulative LSQ occupancy
LSQ_fcount                 39831134 # cumulative LSQ full count
lsq_occupancy                4.8063 # avg LSQ occupancy (insn's)
lsq_rate                     1.2618 # avg LSQ dispatch rate (insn/cycle)
lsq_latency                  3.8091 # avg LSQ occupant latency (cycle's)
lsq_full                     0.1570 # fraction of time (cycle's) LSQ was full
sim_slip                5035324939 # total number of slip cycles
avg_sim_slip               16.7844 # the average slip between issue and retirement
```

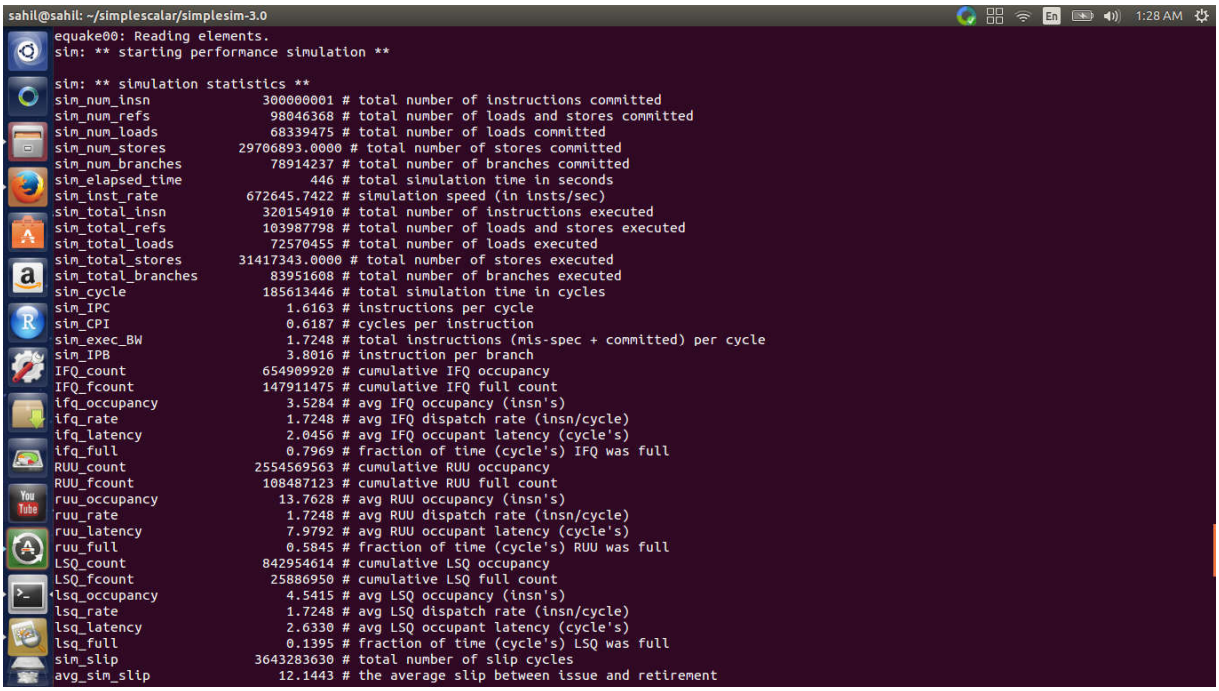
Figure 6: Observed results for infinite L2 cache length Result #3

```
sahil@sahil: ~/simplescalar/simplesim-3.0                                  1:18 AM
equake00: Reading elements.
sim: ** starting performance simulation **

sim: ** simulation statistics **
sim_num_insn              300000001 # total number of instructions committed
sim_num_refs               98046368 # total number of loads and stores committed
sim_num_loads              68339475 # total number of loads committed
sim_num_stores         29706893.0000 # total number of stores committed
sim_num_branches           78914237 # total number of branches committed
sim_elapsed_time                432 # total simulation time in seconds
sim_inst_rate           694444.4468 # simulation speed (in insts/sec)
sim_total_insn            317461070 # total number of instructions executed
sim_total_refs            103988881 # total number of loads and stores executed
sim_total_loads            72570926 # total number of loads executed
sim_total_stores       31417955.0000 # total number of stores executed
sim_total_branches         82604392 # total number of branches executed
sim_cycle                 145612215 # total simulation time in cycles
sim_IPC                      2.0603 # instructions per cycle
sim_CPI                      0.4854 # cycles per instruction
sim_exec_BW                  2.1802 # total instructions (mis-spec + committed) per cycle
sim_IPB                      3.8016 # instruction per branch
IFQ_count                 496922965 # cumulative IFQ occupancy
IFQ_fcount                106768813 # cumulative IFQ full count
ifq_occupancy                3.4126 # avg IFQ occupancy (insn's)
ifq_rate                     2.1802 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                  1.5653 # avg IFQ occupant latency (cycle's)
ifq_full                     0.7332 # fraction of time (cycle's) IFQ was full
RUU_count                1894766307 # cumulative RUU occupancy
RUU_fcount                 64937243 # cumulative RUU full count
ruu_occupancy               13.0124 # avg RUU occupancy (insn's)
ruu_rate                     2.1802 # avg RUU dispatch rate (insn/cycle)
ruu_latency                  5.9685 # avg RUU occupant latency (cycle's)
ruu_full                     0.4460 # fraction of time (cycle's) RUU was full
LSQ_count                615506792 # cumulative LSQ occupancy
LSQ_fcount                 18330095 # cumulative LSQ full count
lsq_occupancy                4.2270 # avg LSQ occupancy (insn's)
lsq_rate                     2.1802 # avg LSQ dispatch rate (insn/cycle)
lsq_latency                  1.9388 # avg LSQ occupant latency (cycle's)
lsq_full                     0.1259 # fraction of time (cycle's) LSQ was full
sim_slip                 2816801952 # total number of slip cycles
avg_sim_slip                 9.3893 # the average slip between issue and retirement
```

Figure 7: Observed results for infinite L2 cache length Result #4

```
sahil@sahil: ~/simplescalar/simplesim-3.0                                  1:07 AM
equake00: Reading elements.
sim: ** starting performance simulation **

sim: ** simulation statistics **
sim_num_insn              300000001 # total number of instructions committed
sim_num_refs               98046368 # total number of loads and stores committed
sim_num_loads              68339475 # total number of loads committed
sim_num_stores         29706893.0000 # total number of stores committed
sim_num_branches           78914237 # total number of branches committed
sim_elapsed_time                481 # total simulation time in seconds
sim_inst_rate           623700.6258 # simulation speed (in insts/sec)
sim_total_insn            319616947 # total number of instructions executed
sim_total_refs            103628535 # total number of loads and stores executed
sim_total_loads            72570854 # total number of loads executed
sim_total_stores       31057681.0000 # total number of stores executed
sim_total_branches         83952631 # total number of branches executed
sim_cycle                 277396865 # total simulation time in cycles
sim_IPC                      1.0815 # instructions per cycle
sim_CPI                      0.9247 # cycles per instruction
sim_exec_BW                  1.1522 # total instructions (mis-spec + committed) per cycle
sim_IPB                      3.8016 # instruction per branch
IFQ_count                 854805268 # cumulative IFQ occupancy
IFQ_fcount                196221228 # cumulative IFQ full count
ifq_occupancy                3.0815 # avg IFQ occupancy (insn's)
ifq_rate                     1.1522 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                  2.6745 # avg IFQ occupant latency (cycle's)
ifq_full                     0.7074 # fraction of time (cycle's) IFQ was full
RUU_count                3538880814 # cumulative RUU occupancy
RUU_fcount                147527676 # cumulative RUU full count
ruu_occupancy               12.7575 # avg RUU occupancy (insn's)
ruu_rate                     1.1522 # avg RUU dispatch rate (insn/cycle)
ruu_latency                 11.0723 # avg RUU occupant latency (cycle's)
ruu_full                     0.5318 # fraction of time (cycle's) RUU was full
LSQ_count                1181865052 # cumulative LSQ occupancy
LSQ_fcount                 39822647 # cumulative LSQ full count
lsq_occupancy                4.2606 # avg LSQ occupancy (insn's)
lsq_rate                     1.1522 # avg LSQ dispatch rate (insn/cycle)
lsq_latency                  3.6978 # avg LSQ occupant latency (cycle's)
lsq_full                     0.1436 # fraction of time (cycle's) LSQ was full
sim_slip                 4884572212 # total number of slip cycles
avg_sim_slip                16.2819 # the average slip between issue and retirement
```

Figure 8: Observed results for infinite L2 cache length Result #5



```
sahil@sahil: ~/simplescalar/simplesim-3.0
equake00: Reading elements.
sim: ** starting performance simulation **

sim: ** simulation statistics **
sim_num_insn              300000001 # total number of instructions committed
sim_num_refs               98046368 # total number of loads and stores committed
sim_num_loads              68339475 # total number of loads committed
sim_num_stores         29706893.0000 # total number of stores committed
sim_num_branches           78914237 # total number of branches committed
sim_elapsed_time                446 # total simulation time in seconds
sim_inst_rate           672645.7422 # simulation speed (in insts/sec)
sim_total_insn            320154910 # total number of instructions executed
sim_total_refs            103987798 # total number of loads and stores executed
sim_total_loads            72570455 # total number of loads executed
sim_total_stores       31417343.0000 # total number of stores executed
sim_total_branches         83951608 # total number of branches executed
sim_cycle                 185613446 # total simulation time in cycles
sim_IPC                      1.6163 # instructions per cycle
sim_CPI                      0.6187 # cycles per instruction
sim_exec_BW                  1.7248 # total instructions (mis-spec + committed) per cycle
sim_IPB                      3.8016 # instruction per branch
IFQ_count                 654909920 # cumulative IFQ occupancy
IFQ_fcount                147911475 # cumulative IFQ full count
ifq_occupancy                3.5284 # avg IFQ occupancy (insn's)
ifq_rate                     1.7248 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                  2.0456 # avg IFQ occupant latency (cycle's)
ifq_full                     0.7969 # fraction of time (cycle's) IFQ was full
RUU_count                2554569563 # cumulative RUU occupancy
RUU_fcount                108487123 # cumulative RUU full count
ruu_occupancy               13.7628 # avg RUU occupancy (insn's)
ruu_rate                     1.7248 # avg RUU dispatch rate (insn/cycle)
ruu_latency                  7.9792 # avg RUU occupant latency (cycle's)
ruu_full                     0.5845 # fraction of time (cycle's) RUU was full
LSQ_count                 842954614 # cumulative LSQ occupancy
LSQ_fcount                 25886950 # cumulative LSQ full count
lsq_occupancy                4.5415 # avg LSQ occupancy (insn's)
lsq_rate                     1.7248 # avg LSQ dispatch rate (insn/cycle)
lsq_latency                  2.6330 # avg LSQ occupant latency (cycle's)
lsq_full                     0.1395 # fraction of time (cycle's) LSQ was full
sim_slip                 3643283630 # total number of slip cycles
avg_sim_slip                12.1443 # the average slip between issue and retirement
```

Figure 9: Observed results for infinite L2 cache length Result #6

Result:

As one can see from the various combinations of the latency cycle. Latency cycle of L1 cache plays the bottlenecking affect. Even if the L2 and memory latency is less, and if L1 has higher latency, then performance is reduced (refer #3). Also when L1 cache is infinite, bottlenecking completely depends on the Latency cycles. While when in L2 infinite cache, program is further secondary bottlenecked (first being L1) by L2. Thus Bottlenecking of the performance due to increase in latency in L1 is most troublesome factor.