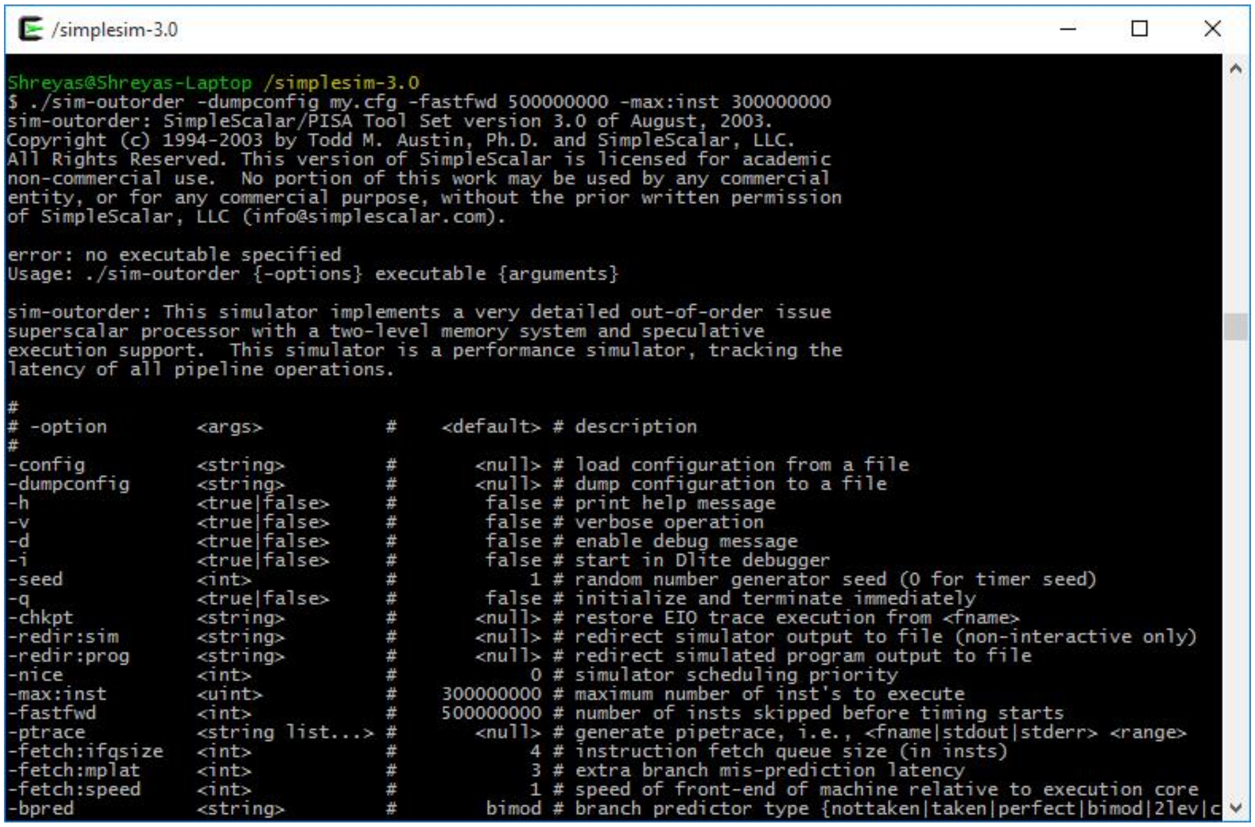
Shreyas Gaonkar
UIN: 657613409

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 Cygwin software 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 my.cfg -fastfwd 500000000 -max:inst 300000000



```
Shreyas@Shreyas-Laptop /simplesim-3.0
$ ./sim-outorder -dumpconfig my.cfg -fastfwd 500000000 -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 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>            #    300000000 # maximum number of inst's to execute
-fastfwd         <int>             #    500000000 # number of insts skipped before timing starts
-ptrace          <string list...>  #       <null> # generate pipetrace, i.e., <fname|stdout|stderr> <range>
-fetch:ifqsize   <int>             #            4 # instruction fetch queue size (in insts)
-fetch:mplat     <int>             #            3 # extra branch mis-prediction latency
-fetch:speed     <int>             #            1 # speed of front-end of machine relative to execution core
-bpred           <string>          #        bimod # branch predictor type {nottaken|taken|perfect|bimod|2lev|c
```

Figure 1: Creating configuration file name my.cfg

Here, the configuration file is saved under the name 'my' is saved under the same folder. Configuration file can be access and modified with any standard text editor and is pretty self-explanatory. Instruction -fastfwd 500000000 is used to fast forward 500M instructions and max:inst: 300000000 is used to execute the next 300M instructions. The program can be then run by the following command –

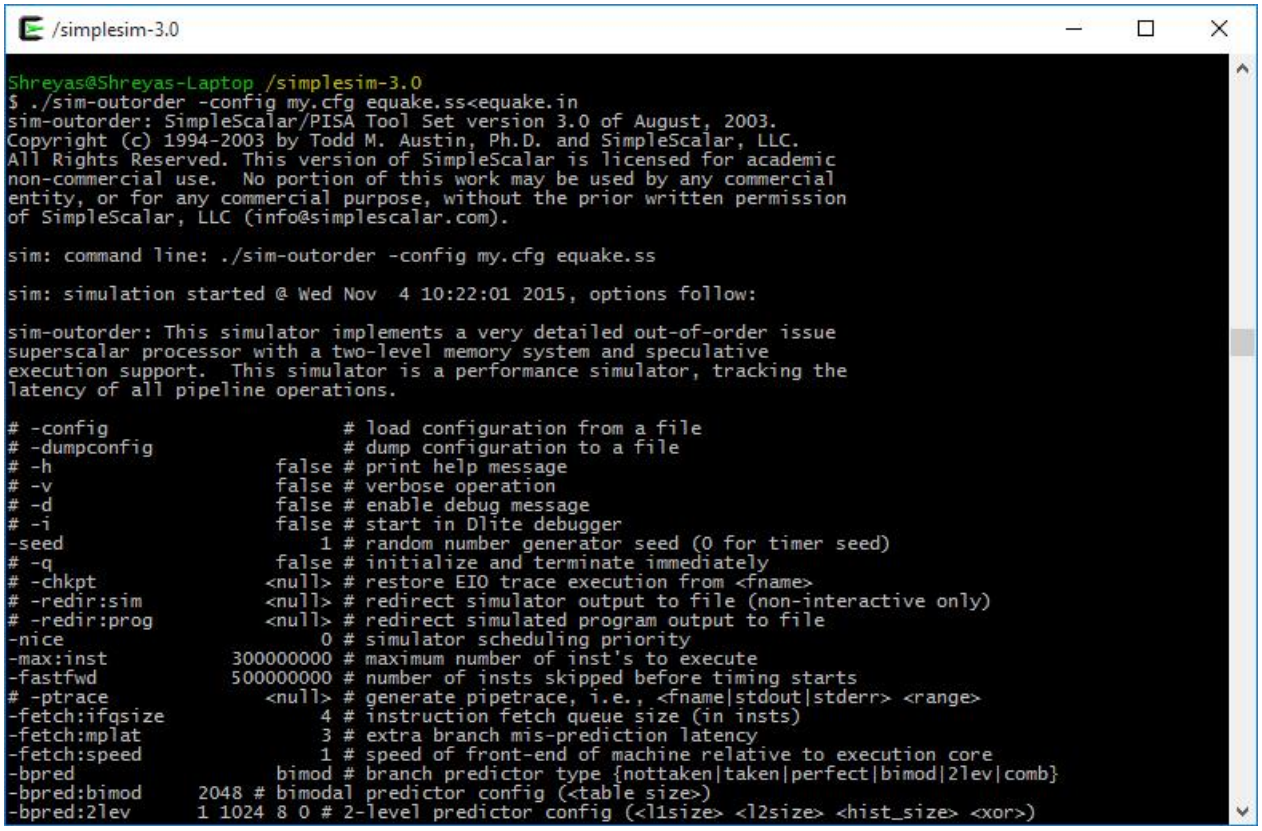
./sim-outorder -config my.cfg equake.ss<equake.in

```
sim: ** simulation statistics **
sim_num_insn              300000001 # total number of instructions committed
sim_num_refs               97642664 # total number of loads and stores committed
sim_num_loads              67983535 # 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                169 # total simulation time in seconds
sim_inst_rate          1775147.9349 # simulation speed (in insts/sec)
sim_total_insn            318884757 # total number of instructions executed
sim_total_refs            103448882 # total number of loads and stores executed
sim_total_loads            72286691 # total number of loads executed
sim_total_stores      31162191.0000 # total number of stores executed
```

```
sim_total_branches          83300410 # total number of branches executed
sim_cycle                  180472212 # total simulation time in cycles
sim_IPC                       1.6623 # instructions per cycle
sim_CPI                       0.6016 # 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                       1.6623 # instructions per cycle
sim_CPI                       0.6016 # cycles per instruction
```



```
Shreyas@Shreyas-Laptop /simplesim-3.0
$ ./sim-outorder -config my.cfg equake.ss<equake.in
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).

sim: command line: ./sim-outorder -config my.cfg equake.ss

sim: simulation started @ Wed Nov  4 10:22:01 2015, options follow:

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.

# -config                     # load configuration from a file
# -dumpconfig                 # dump configuration to a file
# -h                    false # print help message
# -v                    false # verbose operation
# -d                    false # enable debug message
# -i                    false # start in Dlite debugger
-seed                       1 # random number generator seed (0 for timer seed)
# -q                    false # initialize and terminate immediately
# -chkpt               <null> # restore EIO trace execution from <fname>
# -redir:sim           <null> # redirect simulator output to file (non-interactive only)
# -redir:prog          <null> # redirect simulated program output to file
-nice                       0 # simulator scheduling priority
-max:inst           300000000 # maximum number of inst's to execute
-fastfwd            500000000 # number of insts skipped before timing starts
# -ptrace              <null> # generate pipetrace, i.e., <fname|stdout|stderr> <range>
-fetch:ifqsize              4 # instruction fetch queue size (in insts)
-fetch:mplat                3 # extra branch mis-prediction latency
-fetch:speed                1 # speed of front-end of machine relative to execution core
-bpred                  bimod # branch predictor type {nottaken|taken|perfect|bimod|2lev|comb}
-bpred:bimod     2048 # bimodal predictor config (<table size>)
-bpred:2lev      1 1024 8 0 # 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
```

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?

Since 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.

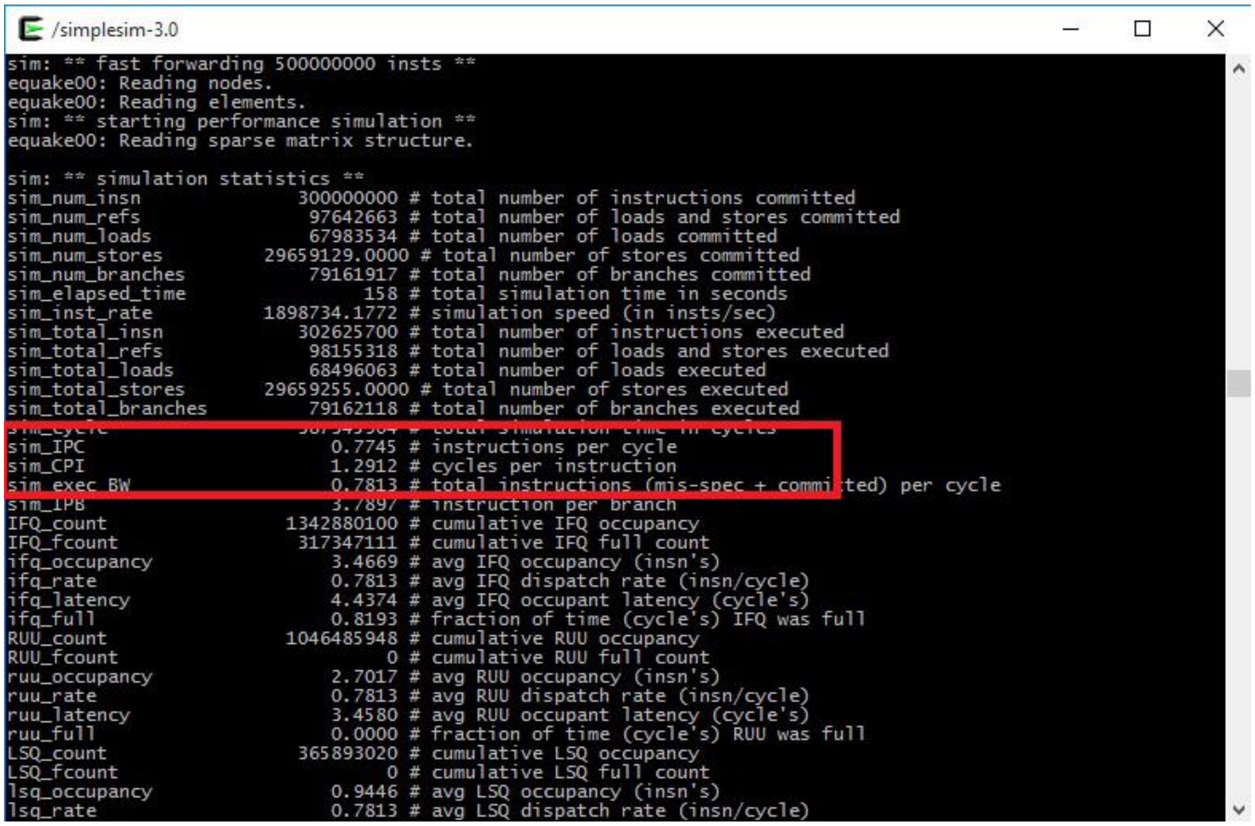
And we can re-run the simulation by the same command as above -
./sim-outorder -config myconfig.cfg equake.ss<equake.in

```
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)
sim_total_insn            302625700 # total number of instructions executed
sim_total_refs             98155318 # total number of loads and stores executed
sim_total_loads            68496063 # total number of loads executed
sim_total_stores      29659255.0000 # total number of stores executed
sim_total_branches         79162118 # total number of branches executed
sim_cycle                 387345904 # total simulation time in cycles
sim_IPC                      0.7745 # instructions per cycle
sim_CPI                      1.2912 # cycles per instruction
```

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

```
sim_IPC                      0.7745 # instructions per cycle
sim_CPI                      1.2912 # cycles per instruction
```

As compared with the default out-of-order execution, we observe 2.1462 times execution time or 46.5939% execution speed in case of in-order execution.



```
/simplesim-3.0
sim: ** fast forwarding 500000000 insts **
equake00: Reading nodes.
equake00: Reading elements.
sim: ** starting performance simulation **
equake00: Reading sparse matrix structure.

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)
sim_total_insn            302625700 # total number of instructions executed
sim_total_refs             98155318 # total number of loads and stores executed
sim_total_loads            68496063 # total number of loads executed
sim_total_stores      29659255.0000 # total number of stores executed
sim_total_branches         79162118 # total number of branches executed
sim_cycle                 387349504 # total simulation time in cycles
sim_IPC                      0.7745 # instructions per cycle
sim_CPI                      1.2912 # cycles per instruction
sim_exec_BW                  0.7813 # total instructions (mis-spec + committed) per cycle
sim_IPB                      3.7897 # instruction per branch
IFQ_count                1342880100 # cumulative IFQ occupancy
IFQ_fcount                317347111 # cumulative IFQ full count
ifq_occupancy                3.4669 # avg IFQ occupancy (insn's)
ifq_rate                     0.7813 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                  4.4374 # avg IFQ occupant latency (cycle's)
ifq_full                     0.8193 # fraction of time (cycle's) IFQ was full
RUU_count                1046485948 # cumulative RUU occupancy
RUU_fcount                        0 # cumulative RUU full count
ruu_occupancy                2.7017 # avg RUU occupancy (insn's)
ruu_rate                     0.7813 # avg RUU dispatch rate (insn/cycle)
ruu_latency                  3.4580 # avg RUU occupant latency (cycle's)
ruu_full                     0.0000 # fraction of time (cycle's) RUU was full
LSQ_count                 365893020 # cumulative LSQ occupancy
LSQ_fcount                        0 # cumulative LSQ full count
lsq_occupancy                0.9446 # avg LSQ occupancy (insn's)
lsq_rate                     0.7813 # avg LSQ dispatch rate (insn/cycle)
```

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 **1.6623**

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

Number of Cycles = 300000000/1.6623

= 180310133.4

And for a 3GHz machine,

Time = No. of cycles X Processor Clock Speed

(Processor Clock Speed = 1/CPU Frequency)

Time = 180310133.4/3x10^9

Time = 0.06010 Seconds

With a 3GHz machine, time required would be **0.0601 seconds** or 60.10 msecs

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          165643162265 # total number of instructions executed
sim_num_refs           78602999410 # total number of loads and stores executed
sim_elapsed_time              5670 # total simulation time in seconds
sim_inst_rate        29213961.5988 # 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 – **165643162265** instructions (~165.6B) and using the value of IPC from the result of the first question, IPC = **1.6623**.

Therefore,

Number of Cycles = Total Instructions/IPC

= 165643162265/1.6623

= 99646972426.75

Which is also equal to the Simulation time for the simulator

But, Simulation Time = Number of Instructions / Simulation Speed

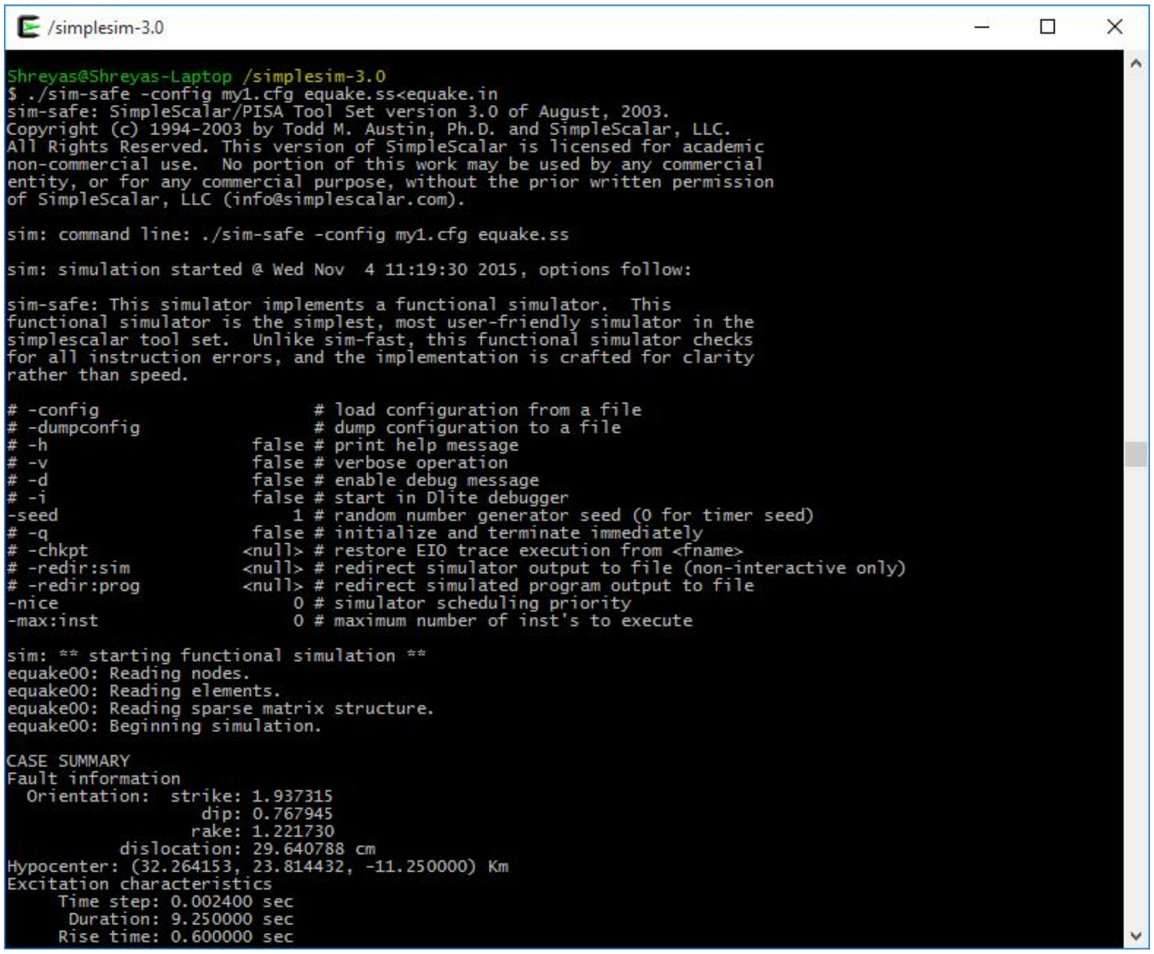
= 165643162265 / 696055

= 237974.24 seconds

= 3966.23 minutes

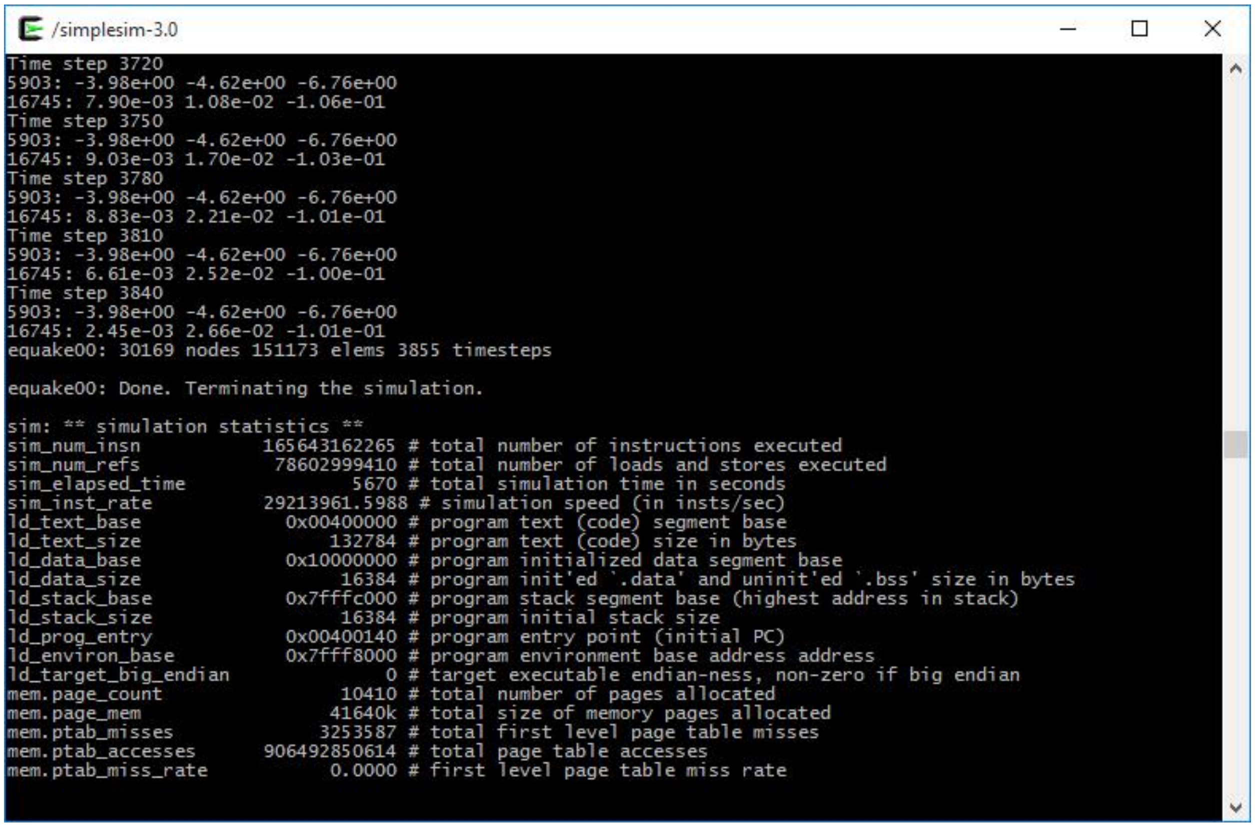
= 66.10 Hours

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


/simplesim-3.0

```
Shreyas@Shreyas-Laptop /simplesim-3.0
$ ./sim-safe -config my1.cfg equake.ss<equake.in
sim-safe: 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).

sim: command line: ./sim-safe -config my1.cfg equake.ss

sim: simulation started @ Wed Nov  4 11:19:30 2015, options follow:

sim-safe: This simulator implements a functional simulator.  This
functional simulator is the simplest, most user-friendly simulator in the
simplescalar tool set.  Unlike sim-fast, this functional simulator checks
for all instruction errors, and the implementation is crafted for clarity
rather than speed.

# -config                        # load configuration from a file
# -dumpconfig                    # dump configuration to a file
# -h                       false # print help message
# -v                       false # verbose operation
# -d                       false # enable debug message
# -i                       false # start in Dlite debugger
-seed                          1 # random number generator seed (0 for timer seed)
# -q                       false # initialize and terminate immediately
# -chkpt                  <null> # restore EIO trace execution from <fname>
# -redir:sim              <null> # redirect simulator output to file (non-interactive only)
# -redir:prog             <null> # redirect simulated program output to file
-nice                          0 # simulator scheduling priority
-max:inst                      0 # maximum number of inst's to execute

sim: ** starting functional simulation **
equake00: Reading nodes.
equake00: Reading elements.
equake00: Reading sparse matrix structure.
equake00: Beginning simulation.

CASE SUMMARY
Fault information
  Orientation:  strike: 1.937315
                   dip: 0.767945
                  rake: 1.221730
           dislocation: 29.640788 cm
Hypocenter: (32.264153, 23.814432, -11.250000) Km
Excitation characteristics
     Time step: 0.002400 sec
      Duration: 9.250000 sec
     Rise time: 0.600000 sec
```

Figure 4: Running sim-safe command for entire program execution



```
Time step 3720
5903: -3.98e+00 -4.62e+00 -6.76e+00
16745: 7.90e-03 1.08e-02 -1.06e-01
Time step 3750
5903: -3.98e+00 -4.62e+00 -6.76e+00
16745: 9.03e-03 1.70e-02 -1.03e-01
Time step 3780
5903: -3.98e+00 -4.62e+00 -6.76e+00
16745: 8.83e-03 2.21e-02 -1.01e-01
Time step 3810
5903: -3.98e+00 -4.62e+00 -6.76e+00
16745: 6.61e-03 2.52e-02 -1.00e-01
Time step 3840
5903: -3.98e+00 -4.62e+00 -6.76e+00
16745: 2.45e-03 2.66e-02 -1.01e-01
equake00: 30169 nodes 151173 elems 3855 timesteps

equake00: Done. Terminating the simulation.

sim: ** simulation statistics **
sim_num_insn           165643162265 # total number of instructions executed
sim_num_refs            78602999410 # total number of loads and stores executed
sim_elapsed_time               5670 # total simulation time in seconds
sim_inst_rate         29213961.5988 # 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
```

Figure 5: Completion of program

4. An advantage of using simulator is that you can vary the processor parameters to see their performance impacts and find the optimal configuration. Start from the default configuration and keep halving or doubling the sizes of L1 instruction cache and data cache, respectively. Show the changes on cache miss rates and performance by varying the cache size. What would be the optimal instruction and data cache sizes for this program based on your experiments?

To change the values of instruction and data cache size, simple change the respective value in the configuration file. This can be changed as following –

```
# l1 data cache config, i.e., {<config>|none}
-cache:dl1          dl1:128:64:4:l (64-bit data cache)
-cache:dl1          dl1:128:32:4:l (32-bit data cache)
-cache:dl1          dl1:128:16:4:l (16-bit data cache)
-cache:dl1          dl1:128:8:4:l    (8-bit data cache)
```

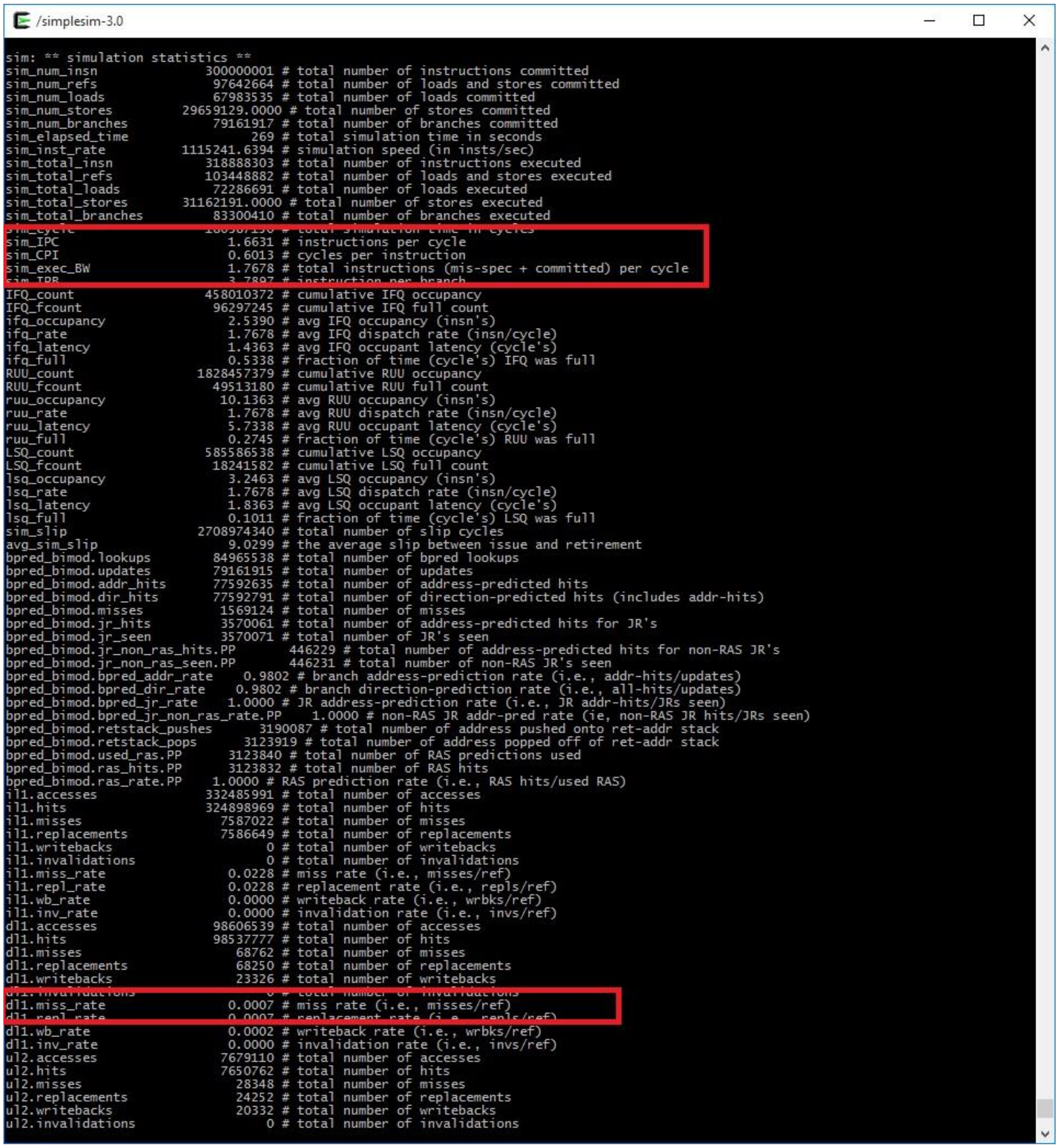
For every execution, I have changed the configuration file and executed the program. For the l1 instruction cache, we do the same thing by changing the following –

```
# l1 inst cache config, i.e., {<config>|dl1|dl2|none}
-cache:il1          il1:512:64:1:l (64-bit Instruction cache)
-cache:il1          il1:512:32:1:l (32-bit Instruction cache)
-cache:il1          il1:512:16:1:l (16-bit Instruction cache)
-cache:il1          il1:512:8:1:l   (8-bit Instruction cache)
```

In both the cases, I have changed the data and instruction cache separately; meaning that when the data cache is changed, the instruction cache is set to default and vice-versa. The summary of the changes made can be seen from the table below and is evident that when the data cache is being halved, it directly affects the IPC, CPI and the miss rate. Lower bit data cache has lesser IPC values and has higher miss rates. When the data cache is halved, the miss rate is doubled. Hence ideally, higher bit data cache would be recommended. Likewise, when the instruction cache is halved, it clearly affects the IPC value and the miss rate. The impact of halving the instruction cache is more severe than halving data cache. As clearly seen, optimum data and instruction cache would 64 bit for both cases.

Observed Results:

Hardware changed	IPC	CPI	Miss Rate
64-bit data cache only	1.6631	0.6013	0.0007
32-bit data cache only	1.6623	0.6016	0.0014
16-bit data cache only	1.6607	0.6021	0.0028
8-bit data cache only	1.6576	0.6033	0.0056
64-bit Instruction cache only	1.7672	0.5659	0.0175
32-bit Instruction cache only	1.6623	0.6016	0.0228
16-bit Instruction cache only	0.9021	1.1085	0.1001
8-bit Instruction cache only	0.3382	2.9566	0.2908


/simplesim-3.0

```
sim: ** simulation statistics **
sim_num_insn              300000001 # total number of instructions committed
sim_num_refs               97642664 # total number of loads and stores committed
sim_num_loads              67983535 # 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                269 # total simulation time in seconds
sim_inst_rate         1115241.6394 # simulation speed (in insts/sec)
sim_total_insn            318888303 # total number of instructions executed
sim_total_refs            103448882 # total number of loads and stores executed
sim_total_loads            72286691 # total number of loads executed
sim_total_stores      31162191.0000 # total number of stores executed
sim_total_branches         83300410 # total number of branches executed
sim_cycle                 180387190 # total simulation time in cycles
sim_IPC                      1.6631 # instructions per cycle
sim_CPI                      0.6013 # cycles per instruction
sim_exec_BW                  1.7678 # total instructions (mis-spec + committed) per cycle
sim_IPB                      3.7897 # instruction per branch
IFQ_count                 458010372 # cumulative IFQ occupancy
IFQ_fcount                 96297245 # cumulative IFQ full count
ifq_occupancy                2.5390 # avg IFQ occupancy (insn's)
ifq_rate                     1.7678 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                  1.4363 # avg IFQ occupant latency (cycle's)
ifq_full                     0.5338 # fraction of time (cycle's) IFQ was full
RUU_count                1828457379 # cumulative RUU occupancy
RUU_fcount                 49513180 # cumulative RUU full count
ruu_occupancy               10.1363 # avg RUU occupancy (insn's)
ruu_rate                     1.7678 # avg RUU dispatch rate (insn/cycle)
ruu_latency                  5.7338 # avg RUU occupant latency (cycle's)
ruu_full                     0.2745 # fraction of time (cycle's) RUU was full
LSQ_count                 585586538 # cumulative LSQ occupancy
LSQ_fcount                 18241582 # cumulative LSQ full count
lsq_occupancy                3.2463 # avg LSQ occupancy (insn's)
lsq_rate                     1.7678 # avg LSQ dispatch rate (insn/cycle)
lsq_latency                  1.8363 # avg LSQ occupant latency (cycle's)
lsq_full                     0.1011 # fraction of time (cycle's) LSQ was full
sim_slip                 2708974340 # total number of slip cycles
avg_sim_slip                 9.0299 # the average slip between issue and retirement
bpred_bimod.lookups        84965538 # total number of bpred lookups
bpred_bimod.updates        79161915 # total number of updates
bpred_bimod.addr_hits      77592635 # total number of address-predicted hits
bpred_bimod.dir_hits       77592791 # total number of direction-predicted hits (includes addr-hits)
bpred_bimod.misses          1569124 # total number of misses
bpred_bimod.jr_hits         3570061 # total number of address-predicted hits for JR's
bpred_bimod.jr_seen         3570071 # total number of JR's seen
bpred_bimod.jr_non_ras_hits.PP       446229 # total number of address-predicted hits for non-RAS JR's
bpred_bimod.jr_non_ras_seen.PP       446231 # total number of non-RAS JR's seen
bpred_bimod.bpred_addr_rate    0.9802 # branch address-prediction rate (i.e., addr-hits/updates)
bpred_bimod.bpred_dir_rate    0.9802 # branch direction-prediction rate (i.e., all-hits/updates)
bpred_bimod.bpred_jr_rate    1.0000 # JR address-prediction rate (i.e., JR addr-hits/JRs seen)
bpred_bimod.bpred_jr_non_ras_rate.PP    1.0000 # non-RAS JR addr-pred rate (ie, non-RAS JR hits/JRs seen)
bpred_bimod.retstack_pushes      3190087 # total number of address pushed onto ret-addr stack
bpred_bimod.retstack_pops      3123919 # total number of address popped off of ret-addr stack
bpred_bimod.used_ras.PP     3123840 # total number of RAS predictions used
bpred_bimod.ras_hits.PP     3123832 # total number of RAS hits
bpred_bimod.ras_rate.PP    1.0000 # RAS prediction rate (i.e., RAS hits/used RAS)
il1.accesses              332485991 # total number of accesses
il1.hits                  324898969 # total number of hits
il1.misses                  7587022 # total number of misses
il1.replacements            7586649 # total number of replacements
il1.writebacks                    0 # total number of writebacks
il1.invalidations                 0 # total number of invalidations
il1.miss_rate                0.0228 # miss rate (i.e., misses/ref)
il1.repl_rate                0.0228 # replacement rate (i.e., repls/ref)
il1.wb_rate                  0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
dl1.accesses               98606539 # total number of accesses
dl1.hits                   98537777 # total number of hits
dl1.misses                    68762 # total number of misses
dl1.replacements              68250 # total number of replacements
dl1.writebacks                23326 # total number of writebacks
dl1.invalidations                 0 # total number of invalidations
dl1.miss_rate                0.0007 # miss rate (i.e., misses/ref)
dl1.repl_rate                0.0007 # replacement rate (i.e., repls/ref)
dl1.wb_rate                  0.0002 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
ul2.accesses                7679110 # total number of accesses
ul2.hits                    7650762 # total number of hits
ul2.misses                    28348 # total number of misses
ul2.replacements              24252 # total number of replacements
ul2.writebacks                20332 # total number of writebacks
ul2.invalidations                 0 # total number of invalidations
```

Figure 6: Observed results for 64-bit data cache