1. **Show the variation of IPC value if the window size is 10 million cycles (collecting the IPC value for every 10 million cycles). Note: you should modify the simulator to get all the statistics in one run instead of running "sim-outorder" multiple times.**

- **Changed code, which doesn't include the non-changed part in order to save space:**

```
  /* main simulator loop, NOTE: the pipe stages are traverse in reverse order to
eliminate this/next state synchronization and relaxation problems */
  for (;sim_cycle <= 50;)
    {
       max_insts == 10000000;
      /* RUU/LSQ sanity checks */
      if (RUU_num < LSQ_num)
    panic("RUU_num < LSQ_num");
      if (((RUU_head + RUU_num) % RUU_size) != RUU_tail)
    panic "RUU_head/RUU_tail wedged"
      if (((LSQ_head + LSQ_num) % LSQ_size) != LSQ_tail)
    panic("LSQ_head/LSQ_tail wedged");

      ptrace_check_active(regs.regs_PC, sim_num_insn, sim_cycle);/* check if
pipetracing is still active */

      ptrace_newcycle(sim_cycle);/* indicate new cycle in pipetrace */

      ruu_commit();/* commit entries from RUU/LSQ to architected register file */

      ruu_release_fu();/* service function unit release events */

      /* ==> may have ready queue entries carried over from previous cycles */

      /* service result completions, also readies dependent operations */
      /* ==> inserts operations into ready queue --> register deps resolved */
      ruu_writeback();

      if (!bugcompat_mode)
    {
      /* try to locate memory operations that are ready to execute */
      /* ==> inserts operations into ready queue --> mem deps resolved */
      lsq_refresh();

      /* issue operations ready to execute from a previous cycle */
      /* <== drains ready queue <-- ready operations commence execution */
      ruu_issue();
    }

      /* decode and dispatch new operations */
      /* ==> insert ops w/ no deps or all regs ready --> reg deps resolved */
      ruu_dispatch();

      if (bugcompat_mode)

      /* try to locate memory operations that are ready to execute */
      /* ==> inserts operations into ready queue --> mem deps resolved */
      lsq_refresh();
      /* issue operations ready to execute from a previous cycle */
      /* <== drains ready queue <-- ready operations commence execution */
      ruu_issue();
    }

      /* call instruction fetch unit if it is not blocked */
      if (!ruu_fetch_issue_delay)
    ruu_fetch();
      else
    ruu_fetch_issue_delay--;

      /* update buffer occupancy stats */
      IFQ_count += fetch_num;
      IFQ_fcount += ((fetch_num == ruu_ifq_size) ? 1 : 0);
      RUU_count += RUU_num;
      RUU_fcount += ((RUU_num == RUU_size) ? 1 : 0);
      LSQ_count += LSQ_num;
      LSQ_fcount += ((LSQ_num == LSQ_size) ? 1 : 0);

      /* go to next cycle */
      sim_cycle++;
      if (sim_cycle % 10000000 == 0){
    printf("IPC at cycle %d is %f:\n",(float)sim_num_insn/(float)sim_cycle);

      /* finish early? */
      if (max_insts && sim_num_insn >= max_insts)
    return;
    }
}
```

- **Results from Ubuntu terminal:**

```
ceciliazhang@ceciliazhang-VirtualBox:~/simplesim/simplesim-3.0$ ./sim-outorder -max:inst 500000000
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 -max:inst 500000000 equake.ss

sim: simulation started @ Tue Dec  4 17:31:54 2018, 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           500000000 # maximum number of inst's to execute
-fastfwd                    0 # 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>)
-bpred:comb      1024 # combining predictor config (<meta_table_size>)
-bpred:ras                  8 # return address stack size (0 for no return stack)
-bpred:btb       512 4 # BTB config (<num_sets> <associativity>)
# -bpred:spec_update       <null> # speculative predictors update in {ID|WB} (default non-spec)
-decode:width               4 # instruction decode B/W (insts/cycle)
-issue:width                4 # instruction issue B/W (insts/cycle)
-issue:inorder          false # run pipeline with in-order issue
-issue:wrongpath         true # issue instructions down wrong execution paths
-commit:width               4 # instruction commit B/W (insts/cycle)
-ruu:size                  16 # register update unit (RUU) size
-lsq:size                   8 # load/store queue (LSQ) size
-cache:dl1       dl1:128:32:4:l # l1 data cache config, i.e., {<config>|none}
-cache:dl1lat               1 # l1 data cache hit latency (in cycles)
-cache:dl2       ul2:1024:64:4:l # l2 data cache config, i.e., {<config>|none}
-cache:dl2lat               6 # l2 data cache hit latency (in cycles)
-cache:il1       il1:512:32:1:l # l1 inst cache config, i.e., {<config>|dl1|dl2|none}
-cache:il1lat               1 # l1 instruction cache hit latency (in cycles)
-cache:il2                dl2 # l2 instruction cache config, i.e., {<config>|dl2|none}
-cache:il2lat               6 # l2 instruction cache hit latency (in cycles)
-cache:flush            false # flush caches on system calls
-cache:icompress        false # convert 64-bit inst addresses to 32-bit inst equivalents
-mem:lat         18 2 # memory access latency (<first_chunk> <inter_chunk>)
-mem:width                  8 # memory access bus width (in bytes)
-tlb:itlb        itlb:16:4096:4:l # instruction TLB config, i.e., {<config>|none}
-tlb:dtlb        dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}
-tlb:lat                   30 # inst/data TLB miss latency (in cycles)
-res:ialu                   4 # total number of integer ALU's available
-res:imult                  1 # total number of integer multiplier/dividers available
-res:memport                2 # total number of memory system ports available (to CPU)
-res:fpalu                  4 # total number of floating point ALU's available
-res:fpmult                 1 # total number of floating point multiplier/dividers available
# -pcstat              <null> # profile stat(s) against text addr's (mult uses ok)
-bugcompat              false # operate in backward-compatible bugs mode (for testing only)

  Pipetrace range arguments are formatted as follows:
```

```
  {{@|#}<start>}:{{@|#|+}<end>}
```

Both ends of the range are optional, if neither are specified, the entire execution is traced. Ranges that start with a `@' designate an address range to be traced, those that start with an `#' designate a cycle count range. All other range values represent an instruction count range. The second argument, if specified with a `+', indicates a value relative to the first argument, e.g., 1000:+100 == 1000:1100. Program symbols may be used in all contexts.

```
  Examples:   -ptrace FOO.trc #0:#1000
              -ptrace BAR.trc @2000:
              -ptrace BLAH.trc :1500
              -ptrace UXXE.trc :
              -ptrace FOOBAR.trc @main:+278
```

Branch predictor configuration examples for 2-level predictor:

```
  Configurations:   N, M, W, X
    N   # entries in first level (# of shift register(s))
    W   width of shift register(s)
    M   # entries in 2nd level (# of counters, or other FSM)
    X   (yes-1/no-0) xor history and address for 2nd level index
  Sample predictors:
    GAg     : 1, W, 2^W, 0
    GAp     : 1, W, M (M > 2^W), 0
    PAg     : N, W, 2^W, 0
    PAp     : N, W, M (M == 2^(N+W)), 0
    gshare  : 1, W, 2^W, 1
```

Predictor `comb' combines a bimodal and a 2-level predictor.

The cache config parameter <config> has the following format:

```
<name>:<nsets>:<bsize>:<assoc>:<repl>

  <name>   - name of the cache being defined
  <nsets>  - number of sets in the cache
  <bsize>  - block size of the cache
  <assoc>  - associativity of the cache
  <repl>   - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random

  Examples:   -cache:dl1 dl1:4096:32:1:l
              -dtlb dtlb:128:4096:32:r
```

Cache levels can be unified by pointing a level of the instruction cache hierarchy at the data cache hiearchy using the "dl1" and "dl2" cache configuration arguments. Most sensible combinations are supported, e.g.,

```
  A unified l2 cache (il2 is pointed at dl2):
    -cache:il1 il1:128:64:1:l -cache:il2 dl2
    -cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l

  Or, a fully unified cache hierarchy (il1 pointed at dl1):
    -cache:il1 dl1
    -cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l

sim: ** starting performance simulation **
equake00: Reading nodes.
IPC at cycle 10000000 is 1.281824:
IPC at cycle 20000000 is 1.247626:
IPC at cycle 30000000 is 1.236372:
IPC at cycle 40000000 is 1.232031:
IPC at cycle 50000000 is 1.228845:
IPC at cycle 60000000 is 1.227065:
IPC at cycle 70000000 is 1.226170:
IPC at cycle 80000000 is 1.224852:
IPC at cycle 90000000 is 1.222030:
IPC at cycle 100000000 is 1.219822:
IPC at cycle 110000000 is 1.218249:
IPC at cycle 120000000 is 1.216939:
IPC at cycle 130000000 is 1.215444:
equake00: Reading elements.
IPC at cycle 140000000 is 1.213988:
IPC at cycle 150000000 is 1.198769:
IPC at cycle 160000000 is 1.202367:
IPC at cycle 170000000 is 1.228594:
IPC at cycle 180000000 is 1.252687:
IPC at cycle 190000000 is 1.274501:
IPC at cycle 200000000 is 1.294153:
IPC at cycle 210000000 is 1.311932:
IPC at cycle 220000000 is 1.328042:
IPC at cycle 230000000 is 1.342731:
IPC at cycle 240000000 is 1.356089:
IPC at cycle 250000000 is 1.368563:
IPC at cycle 260000000 is 1.380430:
IPC at cycle 270000000 is 1.391663:
IPC at cycle 280000000 is 1.402124:
IPC at cycle 290000000 is 1.411865:
IPC at cycle 300000000 is 1.420961:
IPC at cycle 310000000 is 1.429478:
IPC at cycle 320000000 is 1.437446:
IPC at cycle 330000000 is 1.444931:
IPC at cycle 340000000 is 1.451900:

sim: ** simulation statistics **
sim_num_insn              500000001 # total number of instructions committed
sim_num_refs              172399515 # total number of loads and stores
committed
sim_num_loads             117159187 # total number of loads committed
sim_num_stores         55240328.0000 # total number of stores committed
sim_num_branches          118352681 # total number of branches committed
sim_elapsed_time                266 # total simulation time in seconds
sim_inst_rate          1879699.2519 # simulation speed (in insts/sec)
sim_total_insn            529493409 # total number of instructions executed
sim_total_refs            181162997 # total number of loads and stores executed
sim_total_loads           123667445 # total number of loads executed
sim_total_stores       57495552.0000 # total number of stores executed
sim_total_branches        125772957 # total number of branches executed
```

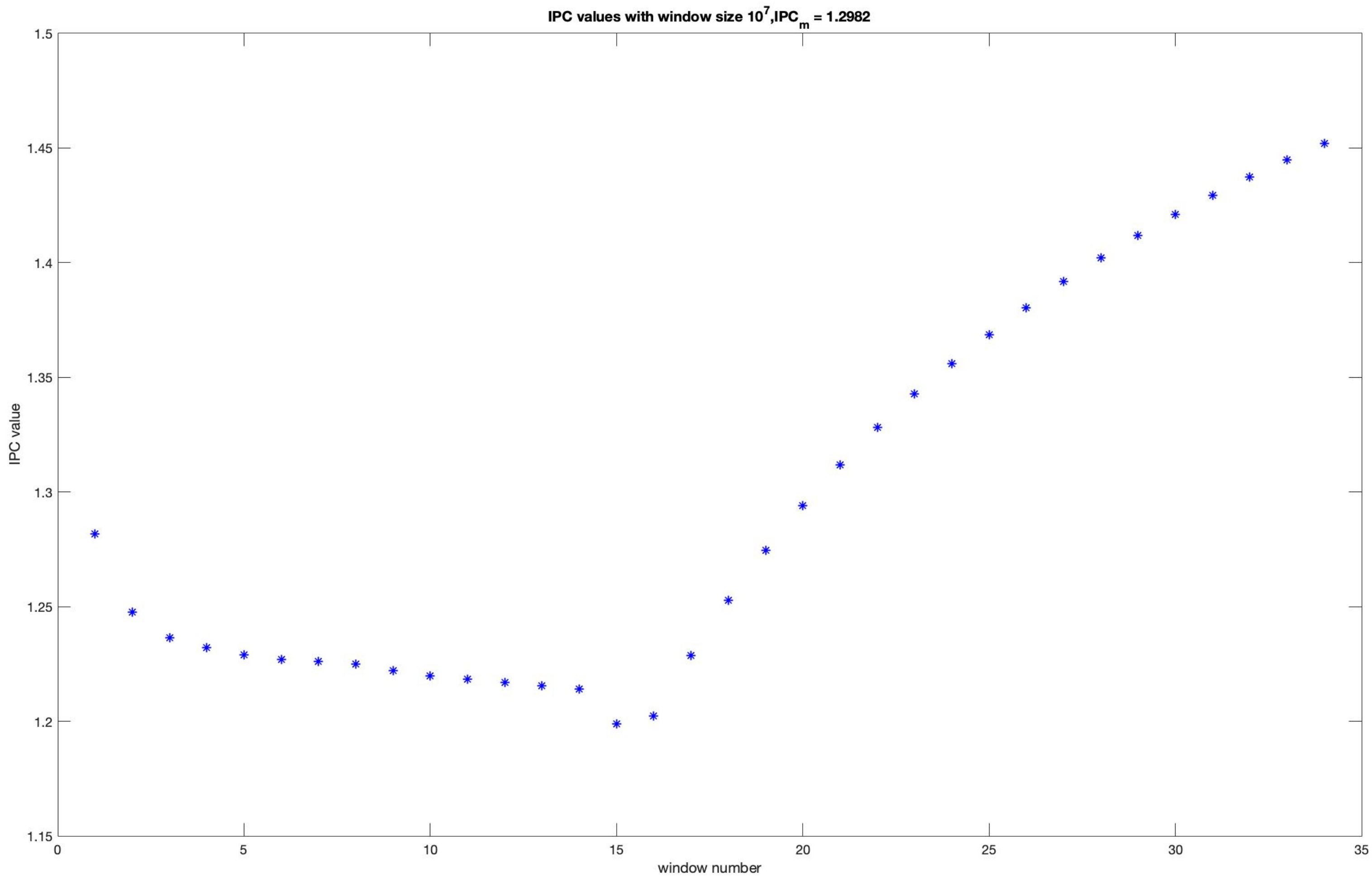
```
sim_cycle                 343775345 # total simulation time in cycles
sim_IPC                      1.4544 # instructions per cycle
sim_CPI                      0.6876 # cycles per instruction
sim_exec_BW                  1.5402 # total instructions (mis-spec + committed) per cycle
sim_IPB                      4.2247 # instruction per branch
IFQ_count                 819880125 # cumulative IFQ occupancy
IFQ_fcount                174459133 # cumulative IFQ full count
ifq_occupancy                2.3849 # avg IFQ occupancy (insn's)
ifq_rate                     1.5402 # avg IFQ dispatch rate (insn/cycle)
ifq_latency                  1.5484 # avg IFQ occupant latency (cycle's)
ifq_full                     0.5075 # fraction of time (cycle's) IFQ was full
RUU_count                3246940107 # cumulative RUU occupancy
RUU_fcount                 89833539 # cumulative RUU full count
ruu_occupancy                9.4449 # avg RUU occupancy (insn's)
ruu_rate                     1.5402 # avg RUU dispatch rate (insn/cycle)
ruu_latency                  6.1322 # avg RUU occupant latency (cycle's)
ruu_full                     0.2613 # fraction of time (cycle's) RUU was full
LSQ_count                1128235790 # cumulative LSQ occupancy
LSQ_fcount                 40860690 # cumulative LSQ full count
lsq_occupancy                3.2819 # avg LSQ occupancy (insn's)
lsq_rate                     1.5402 # avg LSQ dispatch rate (insn/cycle)
lsq_latency                  2.1308 # avg LSQ occupant latency (cycle's)
lsq_full                     0.1189 # fraction of time (cycle's) LSQ was full
sim_slip                 4900566993 # total number of slip cycles
avg_sim_slip                 9.8011 # the average slip between issue and
retirement
bpred_bimod.lookups       128108586 # total number of bpred lookups
bpred_bimod.updates       118352679 # total number of updates
bpred_bimod.addr_hits     114968869 # total number of address-predicted hits
bpred_bimod.dir_hits      115029495 # total number of
direction-predicted hits (includes addr-hits)
bpred_bimod.misses          3323184 # total number of misses
bpred_bimod.jr_hits         7270615 # total number of address-predicted hits for JR's
bpred_bimod.jr_seen         7330968 # total number of JR's seen
bpred_bimod.jr_non_ras_hits.PP       624310 # total number of
address-predicted hits for non-RAS JR's
bpred_bimod.jr_non_ras_seen.PP       684654 # total number of non-RAS JR's seen
bpred_bimod.bpred_addr_rate    0.9714 # branch address-prediction rate (i.e., addr-hits/updates)
bpred_bimod.bpred_dir_rate    0.9719 # branch direction-prediction rate (i.e., all-hits/updates)
bpred_bimod.bpred_jr_rate    0.9918 # JR address-prediction rate (i.e., JR addr-hits/JRs seen)
bpred_bimod.bpred_jr_non_ras_rate.PP    0.9119 # non-RAS JR addr-pred rate (ie, non-RAS JR hits/JRs
seen)
bpred_bimod.retstack_pushes      6767421 # total number of address pushed onto ret-addr stack
bpred_bimod.retstack_pops      6662092 # total number of address popped off of ret-addr stack
bpred_bimod.used_ras.PP      6646314 # total number of RAS predictions used
bpred_bimod.ras_hits.PP      6646305 # total number of RAS hits
bpred_bimod.ras_rate.PP    1.0000 # RAS prediction rate (i.e., RAS hits/used RAS)
il1.accesses              557877598 # total number of accesses
il1.hits                  540110146 # total number of hits
il1.misses                 17767452 # total number of misses
il1.replacements           17767037 # total number of replacements
il1.writebacks                    0 # total number of writebacks
il1.invalidations                 0 # total number of invalidations
il1.miss_rate                0.0318 # miss rate (i.e., misses/ref)
```

```
il1.repl_rate               0.0318 # 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             172886502 # total number of accesses
dl1.hits                 172642207 # total number of hits
dl1.misses                  244295 # total number of misses
dl1.replacements            243783 # total number of replacements
dl1.writebacks              223318 # total number of writebacks
dl1.invalidations                0 # total number of invalidations
dl1.miss_rate               0.0014 # miss rate (i.e., misses/ref)
dl1.repl_rate               0.0014 # replacement rate (i.e., repls/ref)
dl1.wb_rate                 0.0013 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
ul2.accesses              18235065 # total number of accesses
ul2.hits                  18114420 # total number of hits
ul2.misses                  120645 # total number of misses
ul2.replacements            116549 # total number of replacements
ul2.writebacks              108076 # total number of writebacks
ul2.invalidations                0 # total number of invalidations
ul2.miss_rate               0.0066 # miss rate (i.e., misses/ref)
ul2.repl_rate               0.0064 # replacement rate (i.e., repls/ref)
ul2.wb_rate                 0.0059 # writeback rate (i.e., wrbks/ref)
ul2.inv_rate                0.0000 # invalidation rate (i.e., invs/ref)
itlb.accesses            557877598 # total number of accesses
itlb.hits                557877576 # total number of hits
itlb.misses                     22 # total number of misses
itlb.replacements                0 # total number of replacements
itlb.writebacks                  0 # total number of writebacks
itlb.invalidations               0 # total number of invalidations
itlb.miss_rate              0.0000 # miss rate (i.e., misses/ref)
itlb.repl_rate              0.0000 # replacement rate (i.e., repls/ref)
itlb.wb_rate                0.0000 # writeback rate (i.e., wrbks/ref)
itlb.inv_rate               0.0000 # invalidation rate (i.e., invs/ref)
dtlb.accesses            176388536 # total number of accesses
dtlb.hits                176386651 # total number of hits
dtlb.misses                   1885 # total number of misses
dtlb.replacements             1757 # total number of replacements
dtlb.writebacks                  0 # total number of writebacks
dtlb.invalidations               0 # total number of invalidations
dtlb.miss_rate              0.0000 # miss rate (i.e., misses/ref)
dtlb.repl_rate              0.0000 # replacement rate (i.e., repls/ref)
dtlb.wb_rate                0.0000 # writeback rate (i.e., wrbks/ref)
dtlb.inv_rate               0.0000 # invalidation rate (i.e., invs/ref)
sim_invalid_addrs                0 # total non-speculative bogus addresses seen (debug var)
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              1175 # total number of pages allocated
mem.page_mem                4700k # total size of memory pages allocated
mem.ptab_misses             4493 # total first level page table misses
mem.ptab_accesses     3437225764 # total page table accesses
mem.ptab_miss_rate        0.0000 # first level page table miss rate
```

- **Comments:**

By collecting all the IPC values and doing a basic calculation, here I can obtain a simple graph and their average value：



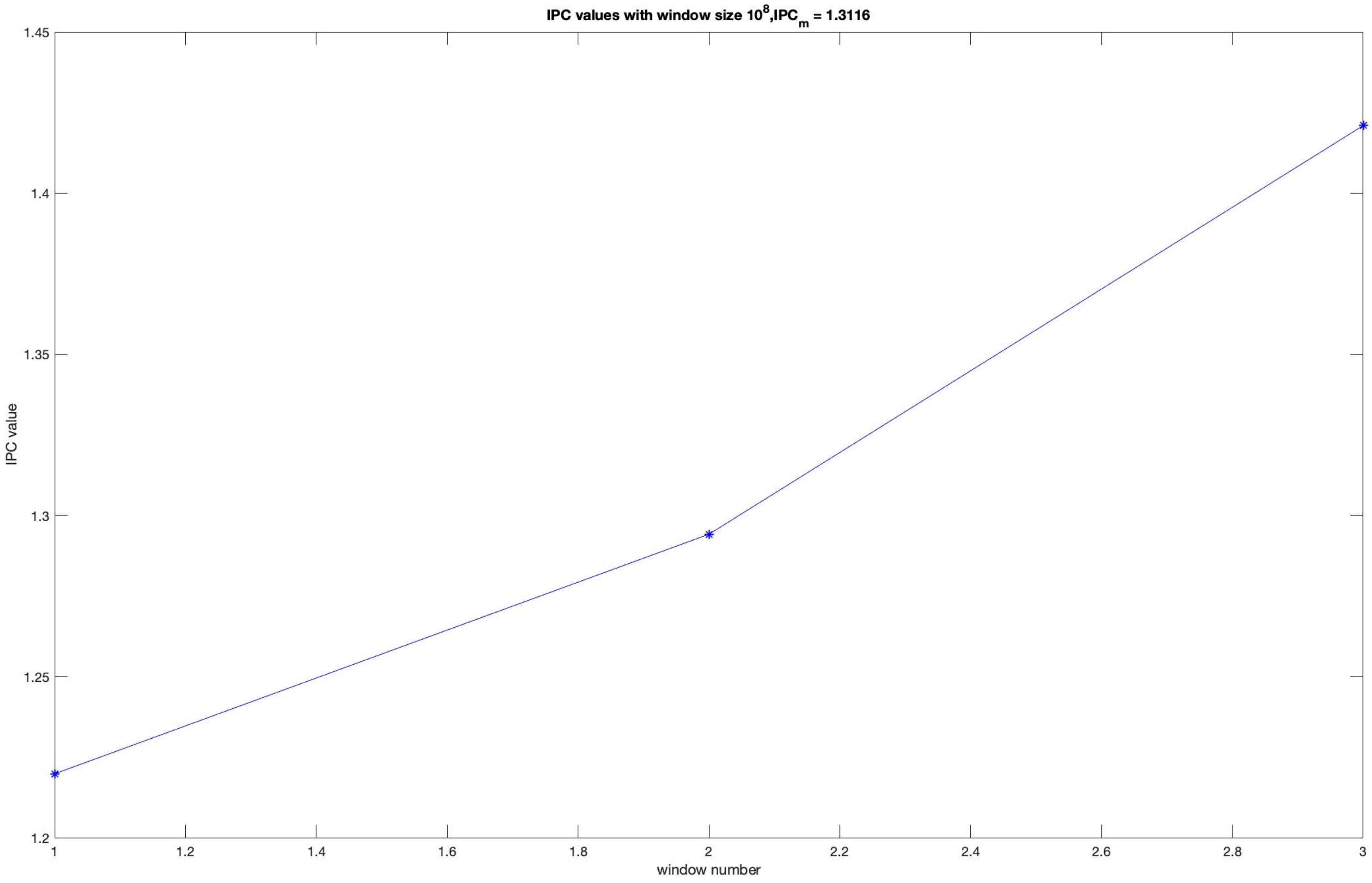
2. **A tricky issue to monitor the change of program behavior is to determine the optimal window size. A too large window won't be able to identify the changes timely, while a too small window will have large overhead and might get unnecessary fluctuations. Use experiments to find the optimal monitor window size for this program. Hint: you can scale the window size up and down by a factor of 10 for fast identification. Show the variation of IPC value if the window size is 10 million cycles (collecting the IPC value for**

every 10 million cycles). Note: you should modify the simulator to get all the statistics in one run instead of running “sim-outorder” multiple times.

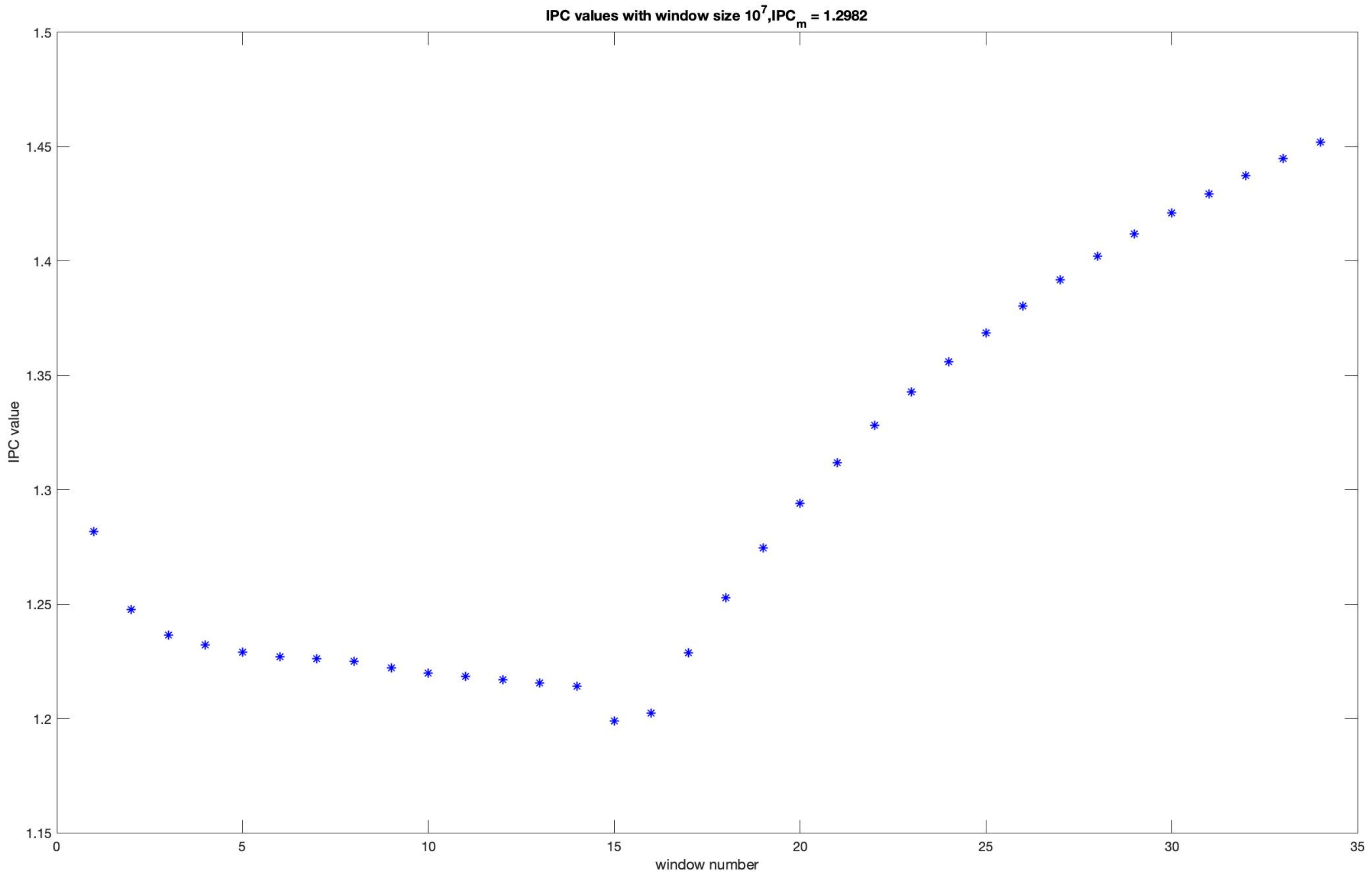
- **By changing the window size multiple times, we can collect values and do a basic analysis:**

Graphs describe the changing tendency of IPC values with window number and fixed window size are following(the same process including changing the window size, let terminal compile the codes, run the C file, collecting data and use a software to generate the graph):

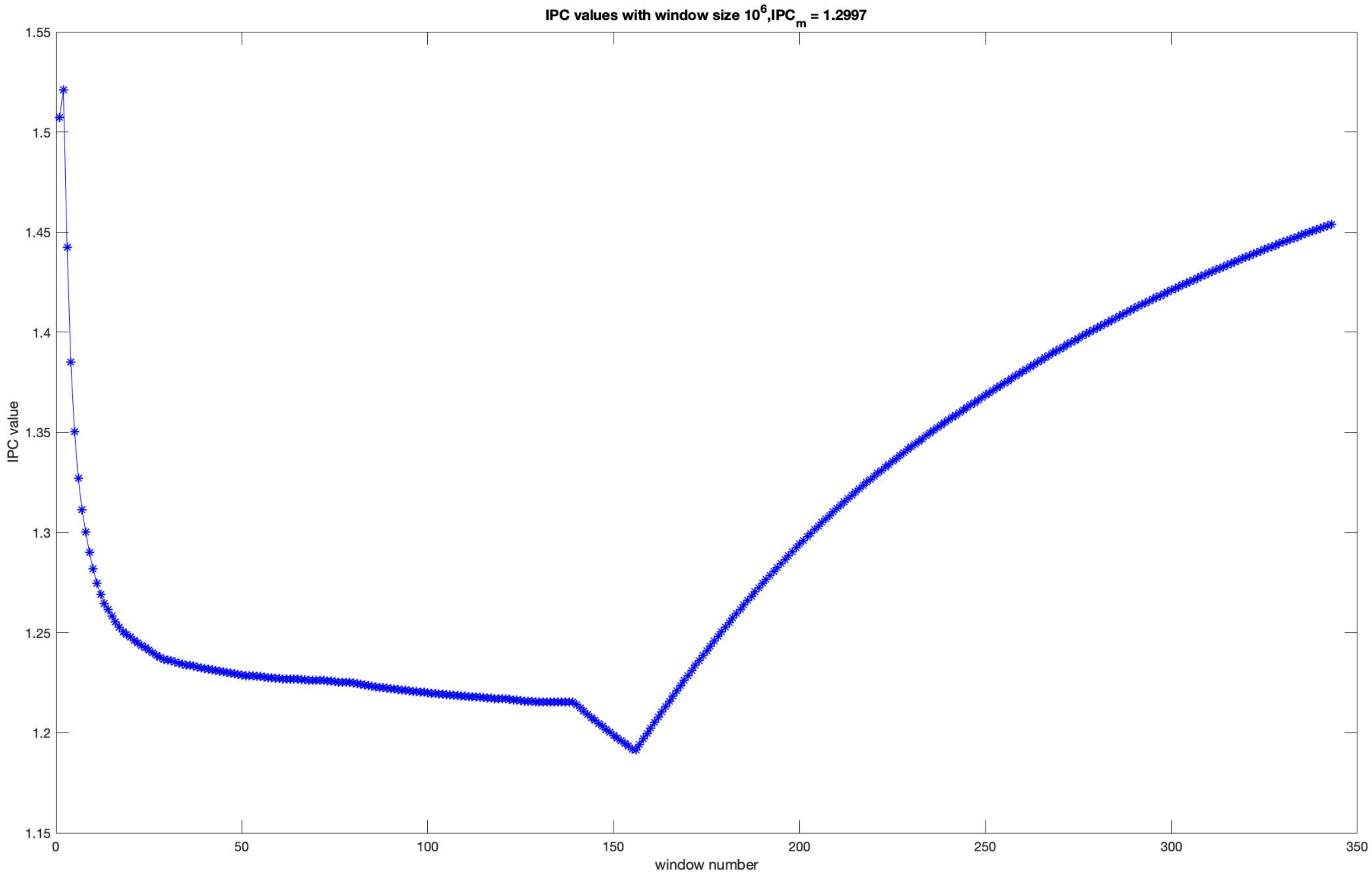
1. Window size: 10^8, average IPC:1.3116



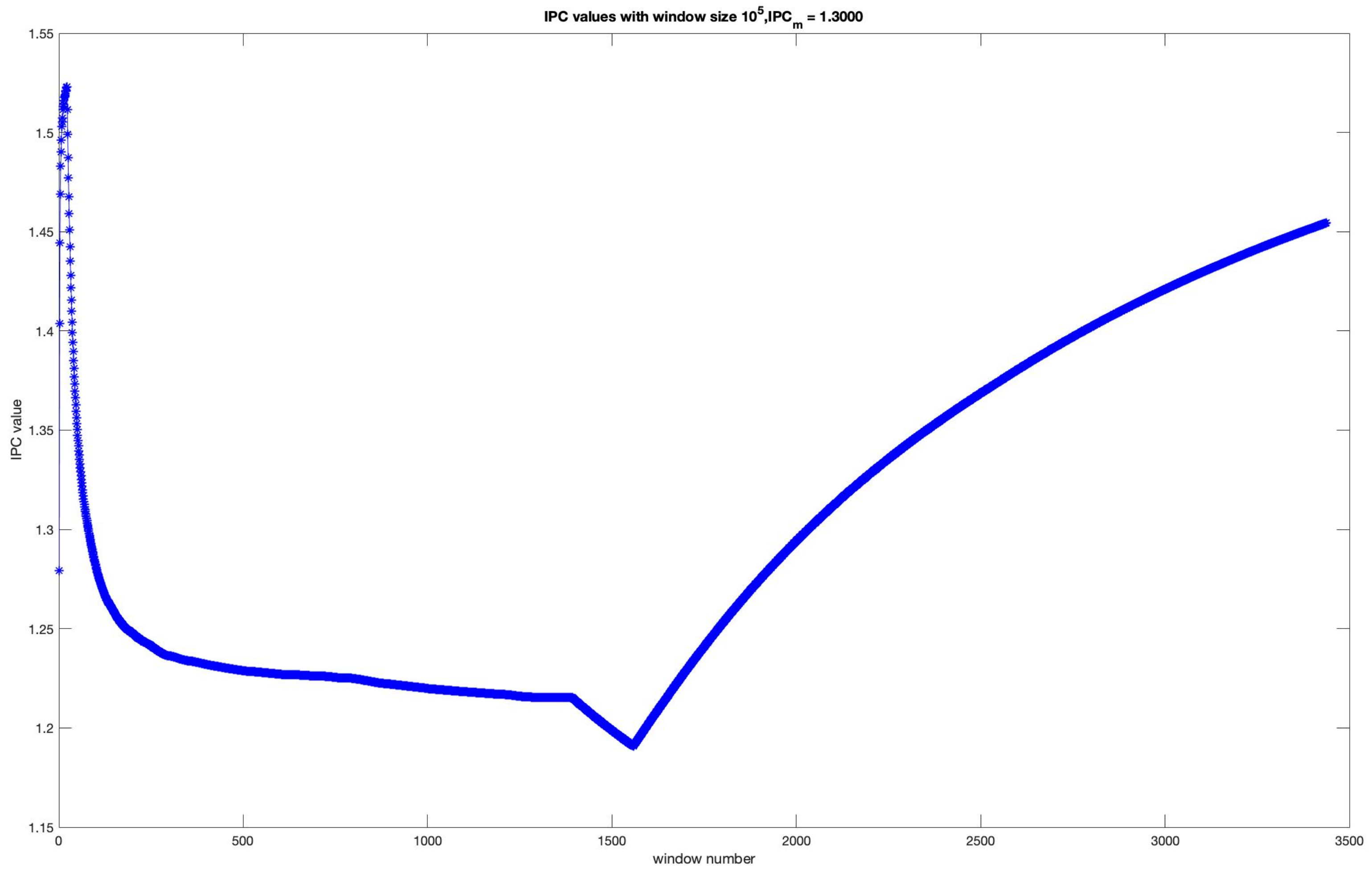
2. Window size=10^7, average IPC=1.2982



3. Window size=10^6, average IPC=1.2997



4. Window size=10^5, average IPC=1.3000



After comparing these graphs, it can be concluded that with a fixed window size the IPC values has a similar tendency and the slop is going to be flat. The worst average IPC is from the case that the window size is 10^7, the best is 10^5.