Project Report

Ye Qiao, Mason Ma

Part 6:

1. Compile and test gapbs

```
Generate Time:       0.00024
Build Time:          0.00007
Graph has 1024 nodes and 16104 undirected edges for degree: 15
  0    0.121699
  1    0.018927
  2    0.003866
  3    0.000901
  4    0.000213
  5    0.000053
Trial Time:          0.00014
  0    0.121699
  1    0.018927
  2    0.003866
  3    0.000901
  4    0.000213
  5    0.000053
Trial Time:          0.00014
Average Time:        0.00014
```

2. Measure the execution times of all benchmarks for "–u 20 –n 4" input
   BFS: 1.182s
   SSSP: 2.841s
   PR: 1.936s
   CC: 1.275s
   BC: 3.572s
   TC: 12.838s

3. Answer: The time of generating a Uniform random graph is significantly lower than generating a Kronecker graph.

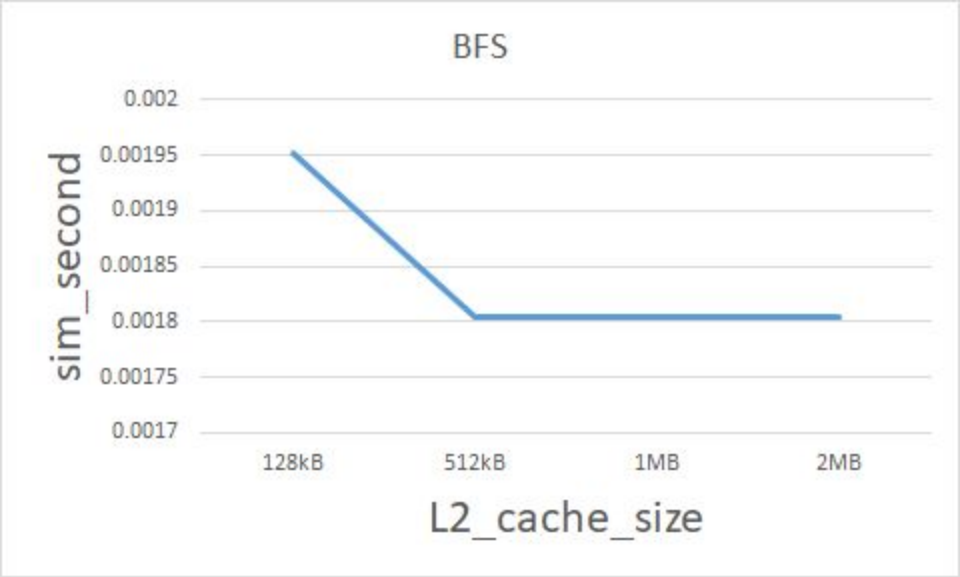
Deliverable:

1. Compile Gem 5

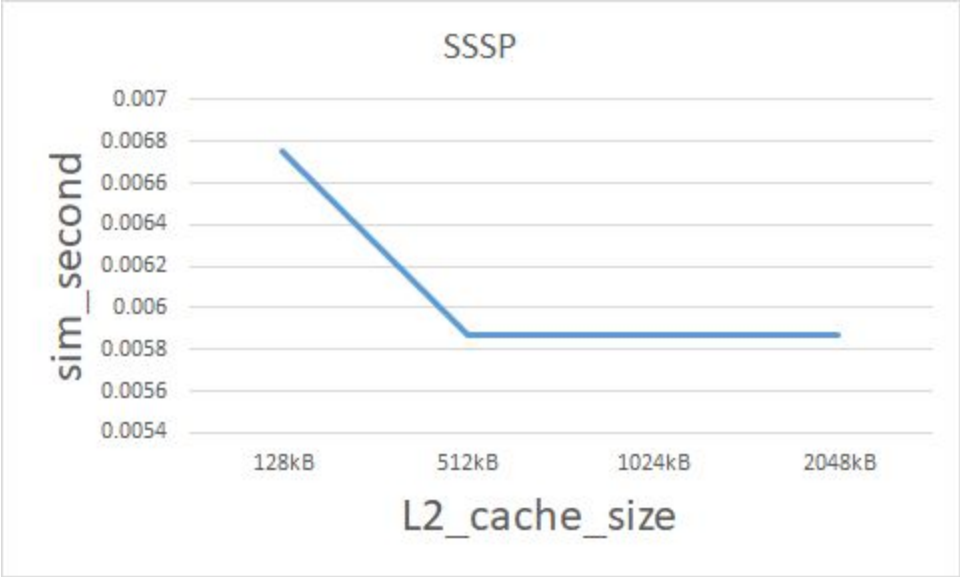
```
[    RANLIB]  -> nomali/libnomali.a
[      LINK]  -> X86/dev/pci/lib.o.partial
[       CXX] X86/base/date.cc -> .o
[      LINK]  -> X86/gem5.opt
scons: done building targets.
*** Summary of Warnings ***
Warning: Your compiler doesn't support incremental linkin
         force lto on anyway, use the --force-lto option.
Warning: Header file <png.h> not found.
         This host has no libpng library.
         Disabling support for PNG framebuffers.
```

3. a)

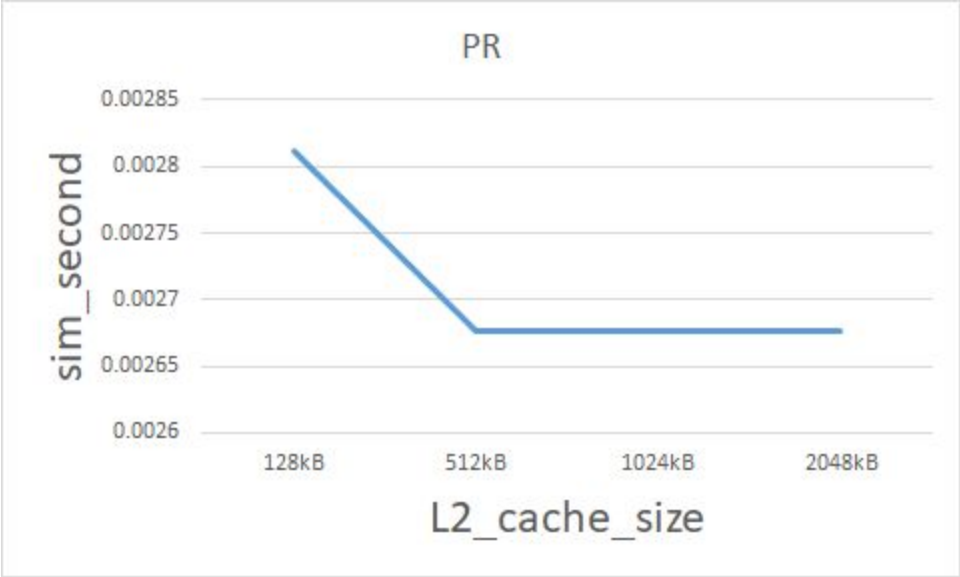
BFS/L2 size	128kB	512kB	1MB	2MB
sim_second	0.001953	0.001804	0.001804	0.001804



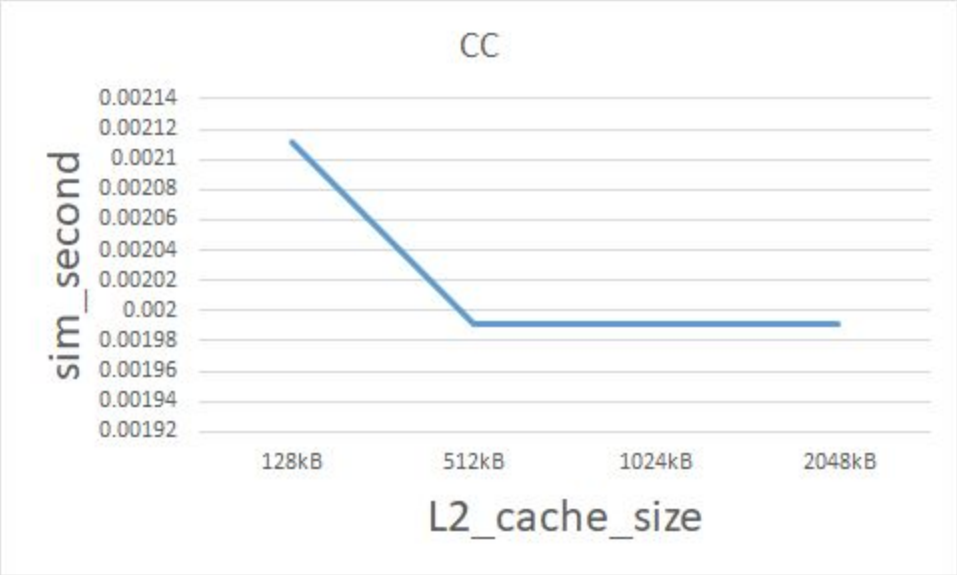
SSSP/L2 size	128kB	512kB	1024kB	2048kB
sim_second	0.006757	0.005873	0.005869	0.005869



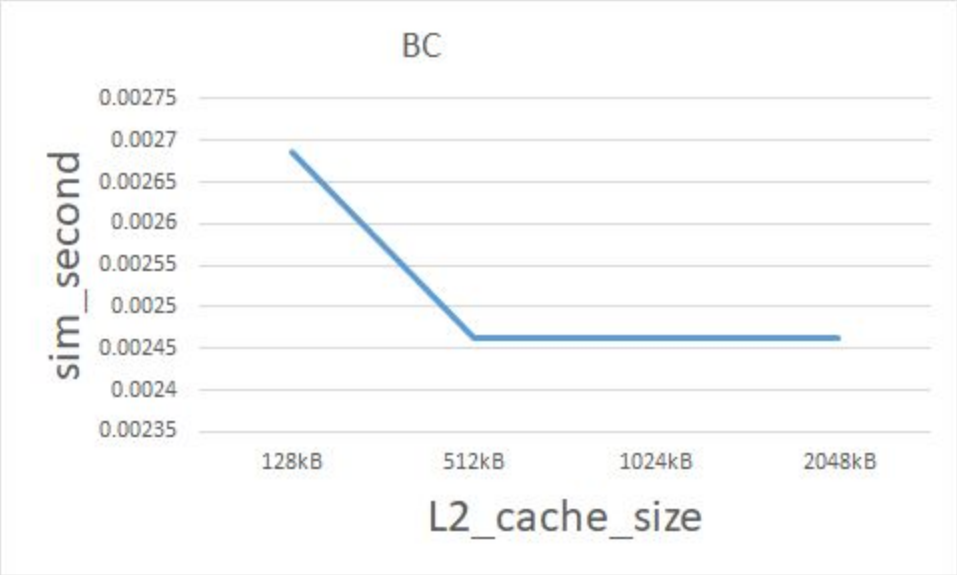
PR/L2 size	128kB	512kB	1024kB	2048kB
sim_second	0.002812	0.002677	0.002677	0.002677



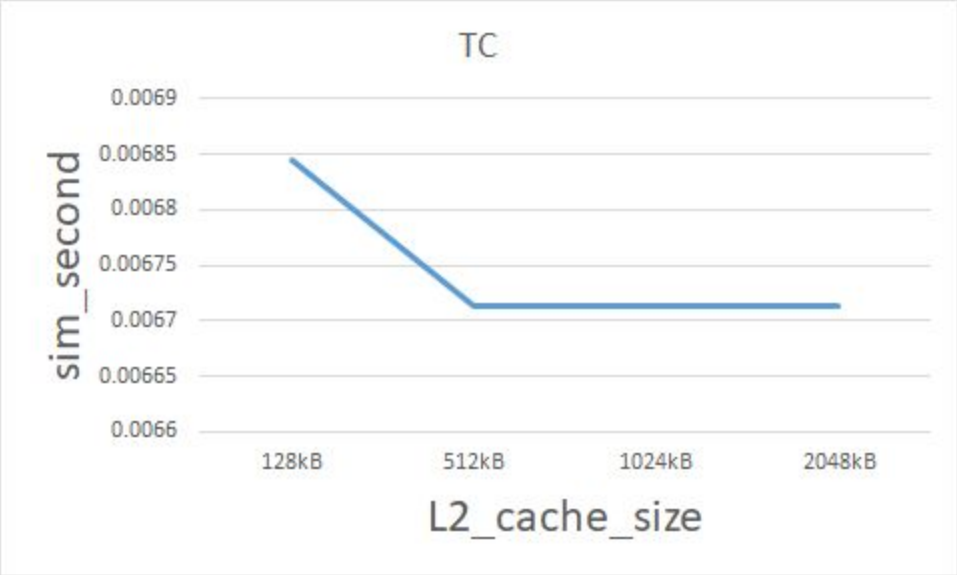
CC/L2 size	128kB	512kB	1024kB	2048kB
sim_second	0.0021117	0.001991	0.001991	0.001991



BC/L2 size	128kB	512kB	1024kB	2048kB
sim_second	0.002685	0.002463	0.002463	0.002463

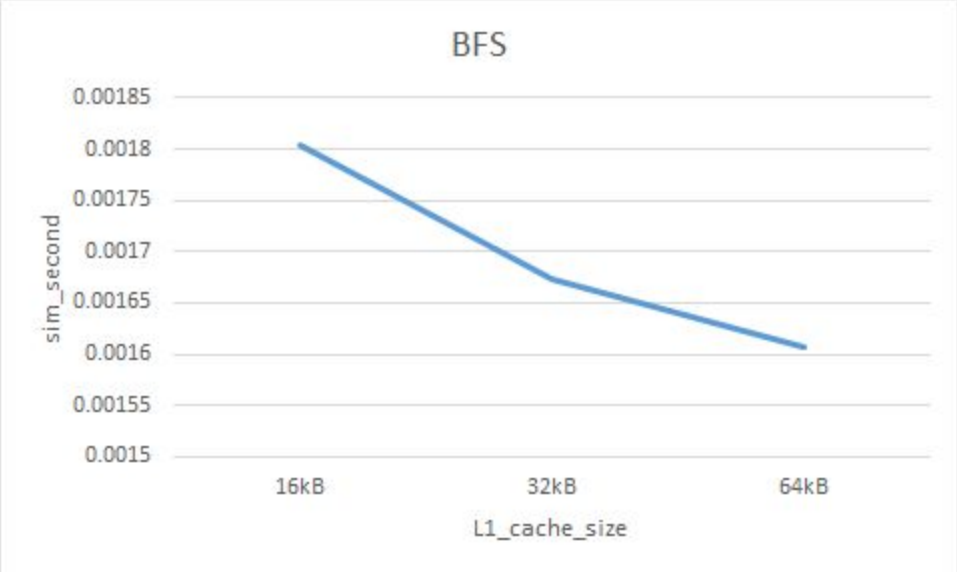


TC/L2 size	128kB	512kB	1024kB	2048kB
sim_second	0.006845	0.006714	0.006714	0.006714



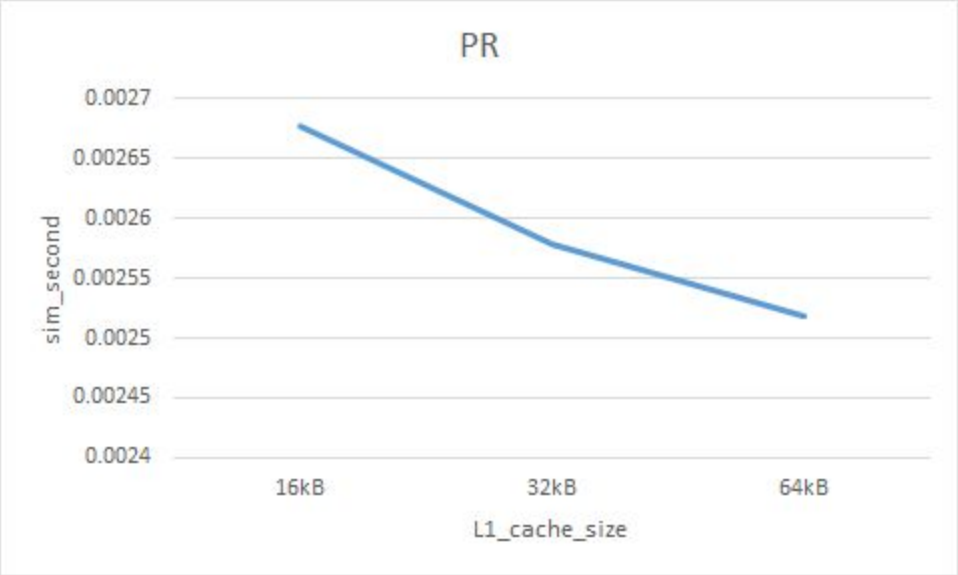
3. b)

BFS/L1 size	16kB	32kB	64kB
sim_second	0.001804	0.001673	0.001606

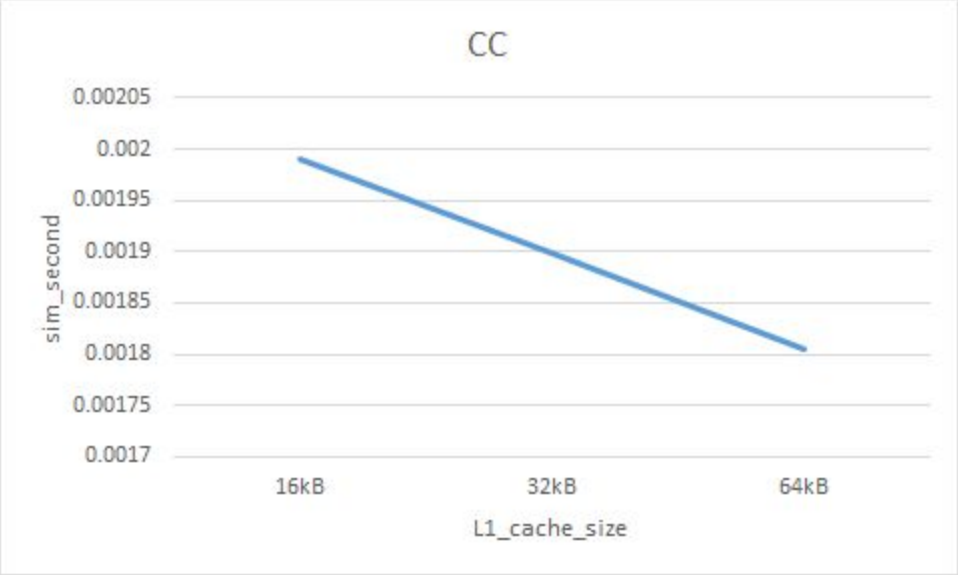


SSSP/L1 size	16kB	32kB	64kB
sim_second	0.005873	0.004536	0.004425

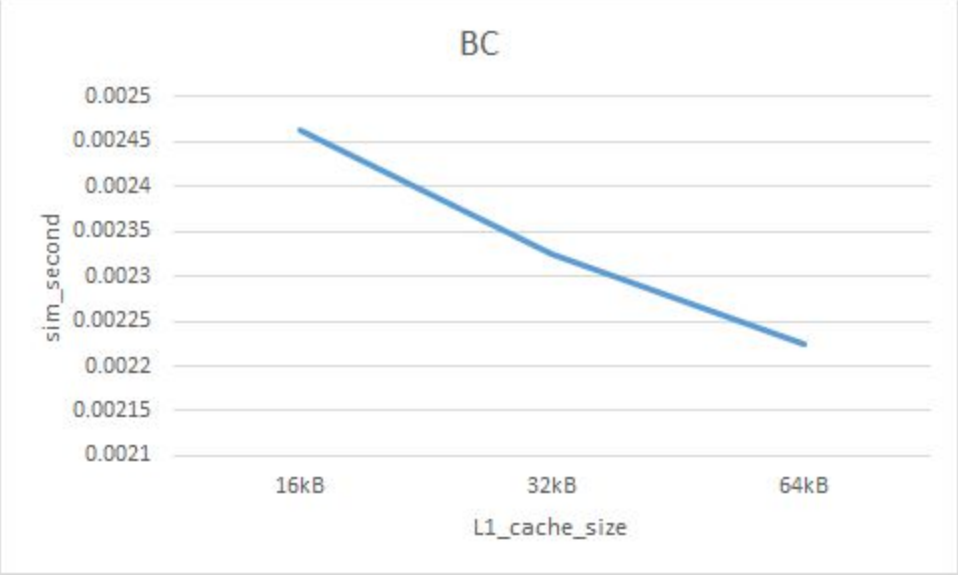
PR/L1 size	16kB	32kB	64kB
sim_second	0.002677	0.002578	0.002518



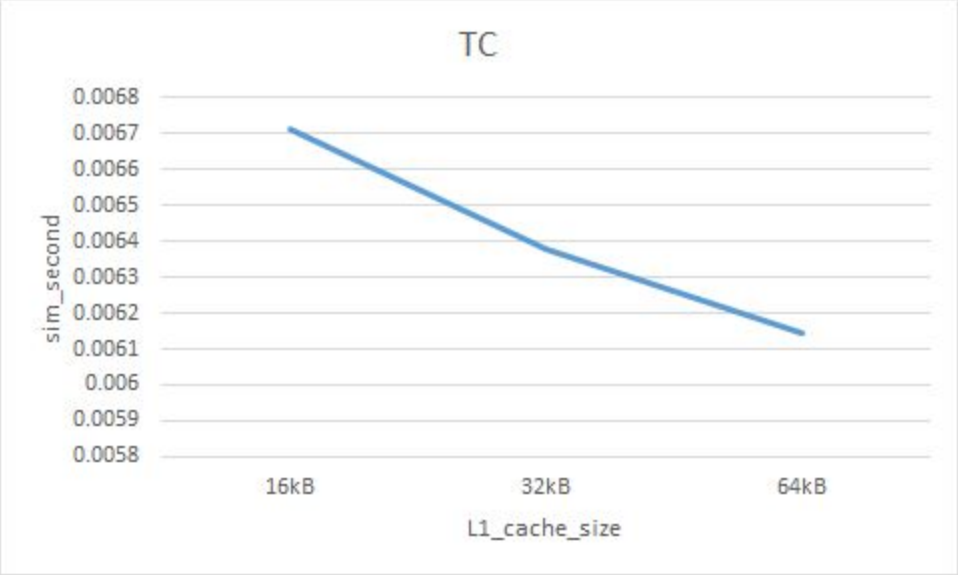
CC/L1 size	16kB	32kB	64kB
sim_second	0.001991	0.001898	0.001805



BC/L1 size	16kB	32kB	64kB
sim_second	0.002463	0.002325	0.002224

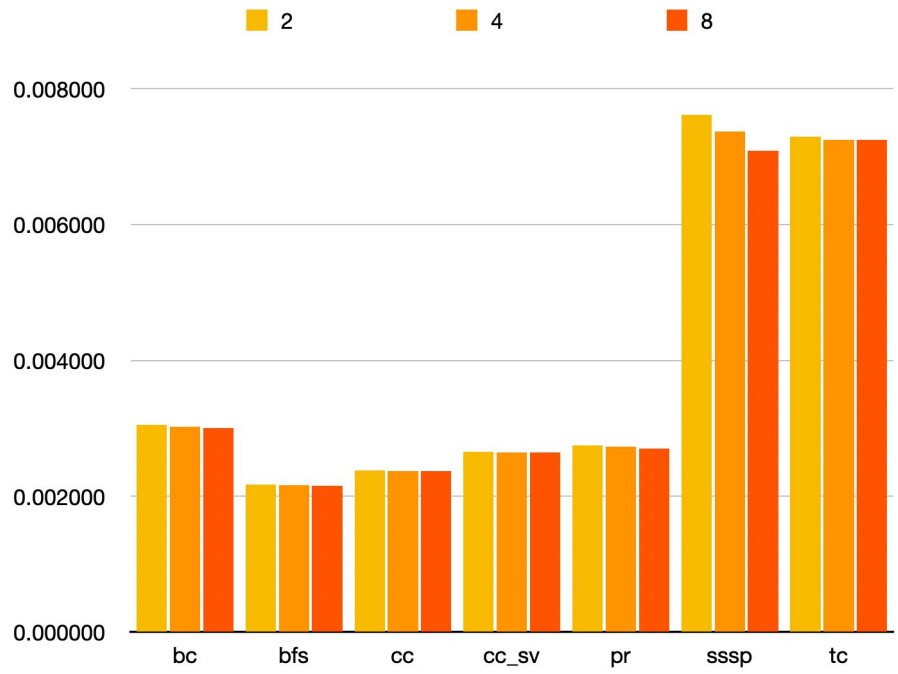


TC/L1 size	16kB	32kB	64kB
sim_second	0.006714	0.006377	0.006142



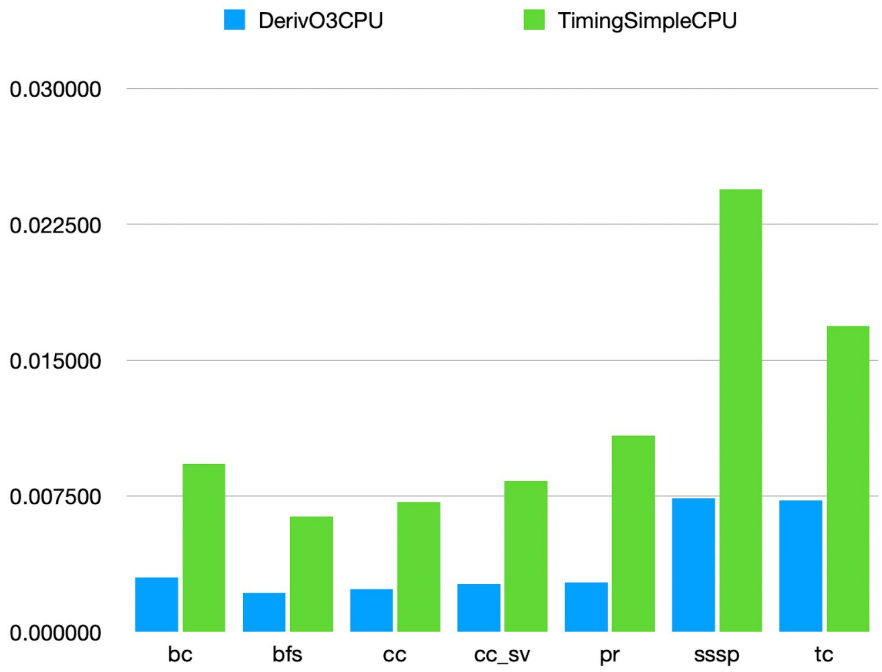
3. c)

L1_Assoc	bc	bfs	cc	cc_sv	pr	sssp	tc
2	0.003050	0.002174	0.002384	0.002652	0.002754	0.007612	0.007291
4	0.003023	0.002161	0.002374	0.002642	0.002731	0.007372	0.007251
8	0.003008	0.002152	0.002375	0.002642	0.002700	0.007086	0.007247



3. d)

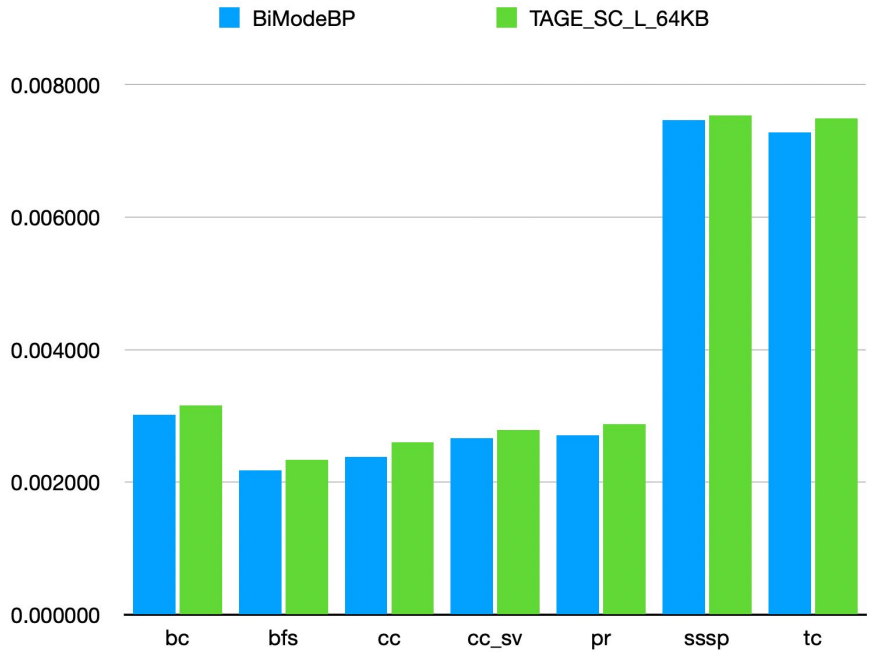
CPU_Type	bc	bfs	cc	cc_sv	pr	sssp	tc
DerivO3CPU	0.003023	0.002161	0.002374	0.002642	0.002731	0.007372	0.007251
TimingSimpleCPU	0.009270	0.006363	0.007174	0.008329	0.010836	0.024431	0.016879



Notes: Clearly the DerivO3 CPU outperforms the TimingSimple CPU in every benchmark. The major difference between these two is that the TimingSimple CPU stalls on cache access and waits for it before continuing to proceed. The DerivO3 CPU, however, is an out-of-order CPU as its name suggests. It does not wait for the previous instructions to execute if this instruction does not depend on them, so some independent instructions can still be executed during the time when the previous instruction waits for the cache miss being satisfied.

3. e)

branch_predictor	bc	bfs	cc	cc_sv	pr	sssp	tc
BiModeBP	0.003015	0.002180	0.002382	0.002662	0.002712	0.007462	0.007278
TAGE_SC_L_64KB	0.003155	0.002335	0.002604	0.002786	0.002872	0.007536	0.007489



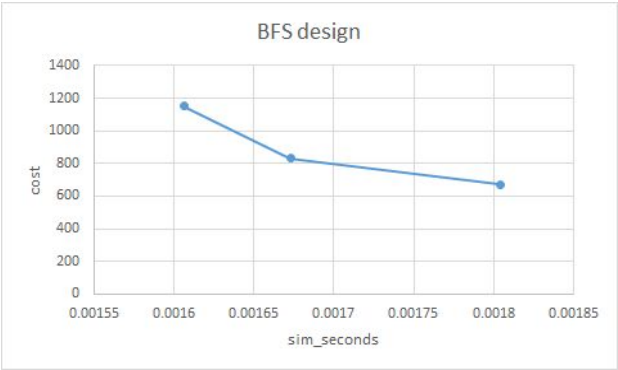
Part 2:

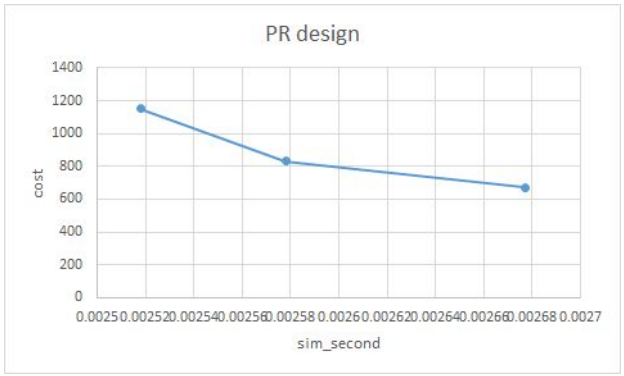
1: Cost = $10\alpha L_1 + \alpha L_2$, we only count cache size into the cost function because the cost-effectiveness of cache associativity, CPU type, and branch prediction modes is unknown.

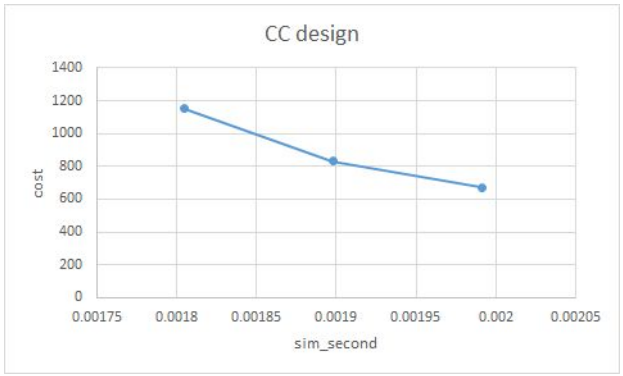
Evaluation = $\frac{1}{\gamma * sim\_time^4 * cost}$ since we are looking for relative evaluation result for each design, the absolute value of weight coefficients are not very important. We give higher polynomial to sim_second to promote its importance in our design.

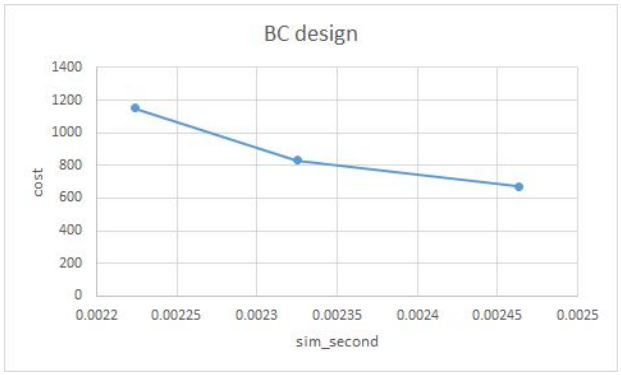
2: Interm of our price assumption (L1 cache is 10 times expensive than L2 cache) and simple evaluation function, the design choice for benchmarks are L1 = 32 kB with 8-way associative, L2 = 512kB with DerivO3CPU, and BiMode branch predictor.

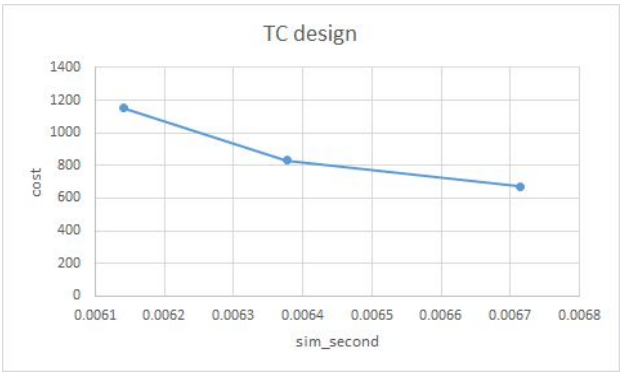
3:




PR design
cost
1400
1200
1000
800
600
400
200
0
0.0025 0.00252 0.00254 0.00256 0.00258 0.0026 0.00262 0.00264 0.00266 0.00268 0.0027
sim_second


CC design
cost
1400
1200
1000
800
600
400
200
0
0.00175 0.0018 0.00185 0.0019 0.00195 0.002 0.00205
sim_second


BC design
cost
1400
1200
1000
800
600
400
200
0
0.0022 0.00225 0.0023 0.00235 0.0024 0.00245 0.0025
sim_second


TC design
cost
1400
1200
1000
800
600
400
200
0
0.0061 0.0062 0.0063 0.0064 0.0065 0.0066 0.0067 0.0068
sim_second