Assignment 4

Haiyu Wang

1. Configuration Files

hw4opts.py

```
from m5 import fatal
import m5.objects
from textwrap import TextWrapper
from common import BPConfig

# add options
def addHW4Opts(parser):
  parser.add_option("--branch-pred", type="str",
default="TAGE",dest="branch_pred")
  parser.add_option("--width", type="int",default=0)

# set parameters taken in from options on command line
def set_config(cpu_list, options):
  for cpu in cpu_list:
    bpClass = BPConfig.get(options.branch_pred)
    cpu.branchPred = bpClass()
    cpu.fetchWidth = options.width
    cpu.decodeWidth = options.width
    cpu.renameWidth = options.width
    cpu.dispatchWidth = options.width
    cpu.issueWidth = options.width
    cpu.wbWidth = options.width
    cpu.commitWidth = options.width
    cpu.squashWidth = options.width
    pass
```

Script for Cache sizes

```
for i in 8 16 32 64
do
   for j in 8 16 32 64
    do
       $GEM5/build/ARM/gem5.opt --outdir=${i}_${j} ./hw4config.py --
cpu-type="DerivO3CPU"          --caches          --l2cache          --
cmd="/project/linuxlab/gem5/test_progs/dibs/gotrackcsv_to_csv"     --
l1d_size=${i}'kB' --l1i_size=${j}'kB'
   done
done
```

Script for Pipeline Width

```
for i in 2 4 8
do
   $GEM5/build/ARM/gem5.opt --outdir=out_width${i} ./hw4config.py --
cpu-type="DerivO3CPU"          --caches          --l2cache          --
cmd="/project/linuxlab/gem5/test_progs/dibs/gotrackcsv_to_csv"     --
width=$i
Done
```

To vary CPU Cores, we can just change `--cpu-type=''`. In my test, I chose AtomicSimpleCPU, DerivO3CPU, MinorCPU, TimingSimpleCPU.

To vary Branch Predictor, we can add an options –branch-pred. The script for Branch Predictor as following: (we can vary i to vary the Branch Predictor)

```
i='SimpleIndirectPredictor'
$GEM5/build/ARM/gem5.opt    --outdir=${i}    ./hw4config.py    --cpu-
type="DerivO3CPU"    -branch-pred=${i}    --caches    --l2cache    --
```

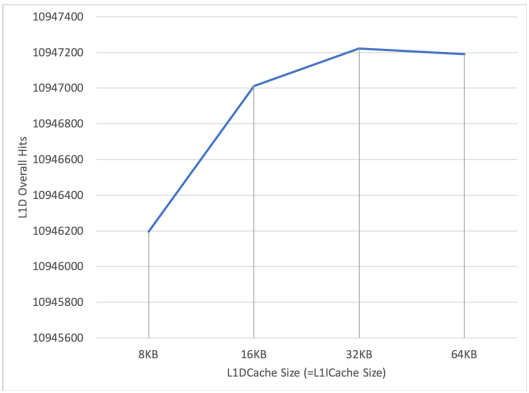
```
cmd="/project/linuxlab/gem5/test_progs/dibs/gotrackcsv_to_csv"
```

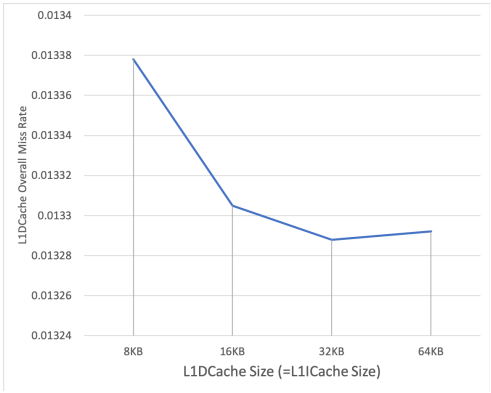
2. Result

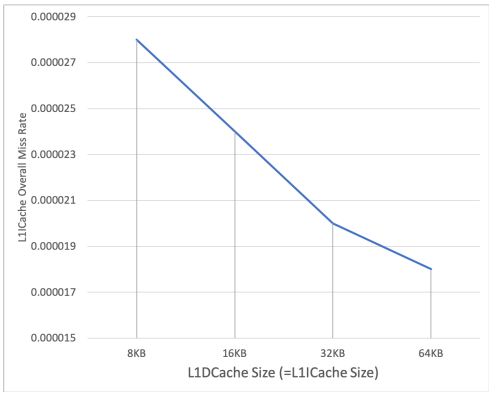
My student ID number is 475533 so I use `gotrackcsv_to_csv` as the test files.

(1) Cache Sizes

I chose L1DCache Sizes and L1ICache Sizes both ranging from 8KB to 64KB. When I analyzed the raw data, I found that when L1DCache Sizes == L1ICache Size, the changes are significant.





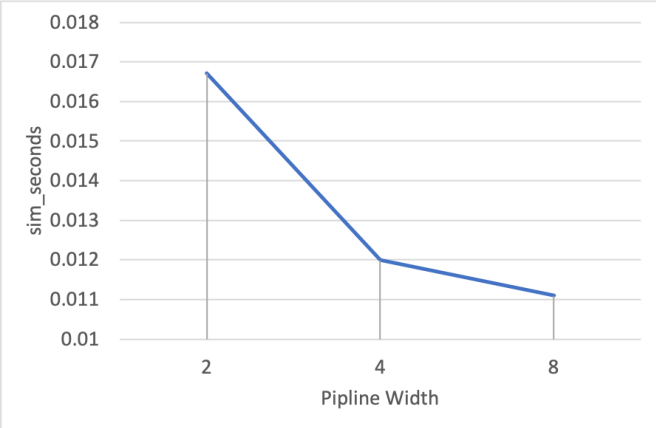


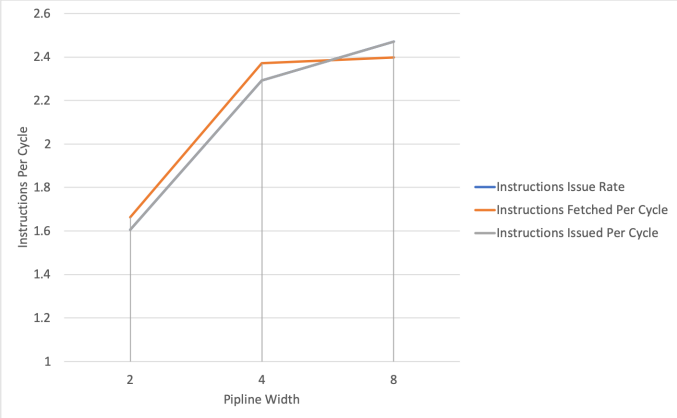
As the cache sizes increase, the L1DCache Overall Hits increase, the L1DCache Overall Miss Rate decrease and the L1ICache Overall Miss Rate also decrease. Because the cache sizes become larger, the caches can store more data so that there could be more hits and the Miss Rate will definitely decrease.

However, we can also see from the 3 diagrams that with the increment of the cache sizes, the rates of increase/decrease become slow. This is Because CPU needs to spend more time looking up the caches to find the data as the caches become larger, which can limit the increase of the rates.

(2) Pipeline Width

From the GEM5' source code, the maximum pipeline width of different stages is 8 and it doesn't support pipeline width 2, so I tested width of 2, 4 and 8.





As we can see from the diagrams, with the increment of the Pipeline Width, the total simulated time decrease, because we can fetch and issue more instructions which is helpful to reduce the simulated time.

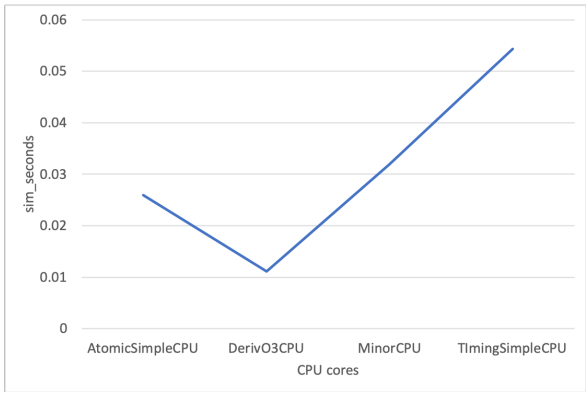
However, the increment of pipeline width can't lead to infinite reduction of the simulated time or increase of the Instructions Issue Rate. Due to delays and dependence between instructions, when the instructions fetched or issued per cycle can reach the pipeline width. When the pipeline width increase from 4 to 8, the issue rate and fetch rate only increase a little.

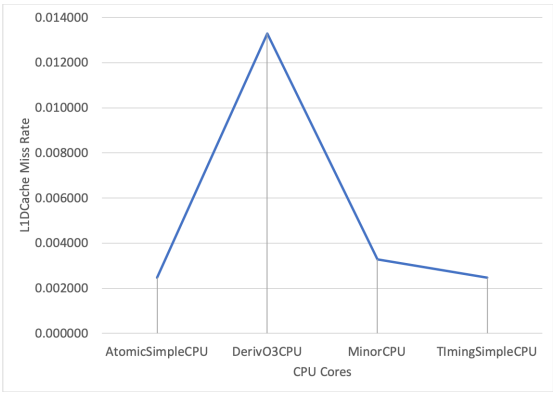
(3) CPU Cores

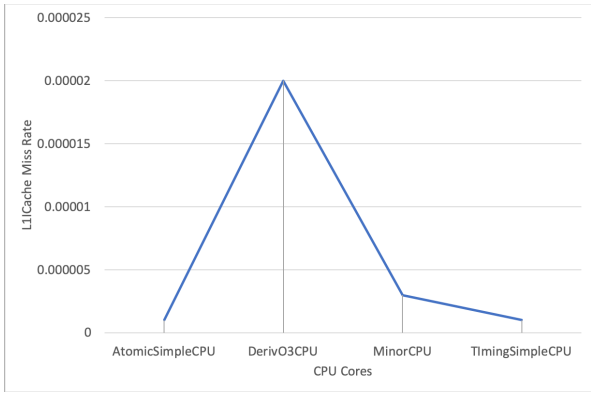
In my analysis, I tested 4 CPU Cores, i.e. AtomicSimpleCPU, DerivO3CPU, MinorCPU and TimingSimpleCPU. Their features are as follows:

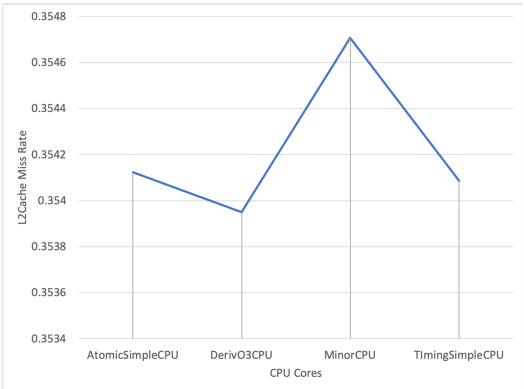
AtomicSimpleCPU	No timing, fast-forwarding and cache warming.
TimingSimpleCPU	Single-cycle (IPC=1) except for memory operations
DerivO3CPU	Out-of-order model. Highly configurable
MinorCPU	In-order model

Results:









As we can see from the 4 graphs, the DerivO3CPU has the least simulated seconds but it also has the highest L1DCache and L1ICache Miss Rate. Because DerivO3CPU is an Out-of-order model, though it has the highest cache miss rates, with lower delay and no need to wait for other instructions, it can issue instructions more quickly which lead to less simulated time.

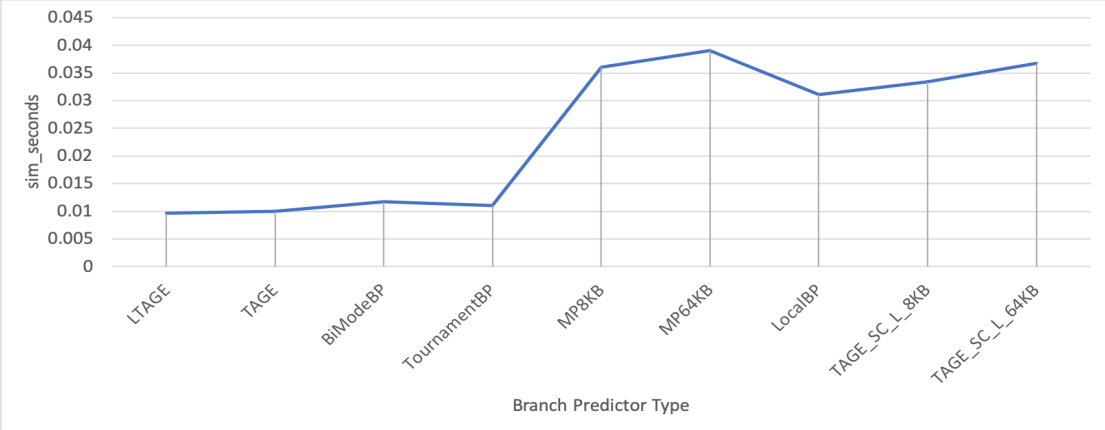
MinorCPU has the highest L2Cache Miss Rate, while the DerivO3CPU has the lowest. Considering that MinorCPU is a In-order model, we can infer that the Out-of-order CPU may can improve the performance of the L2Cache with the cost of the performance of the L1DCache and L1ICache.

(4) Branch Predictor

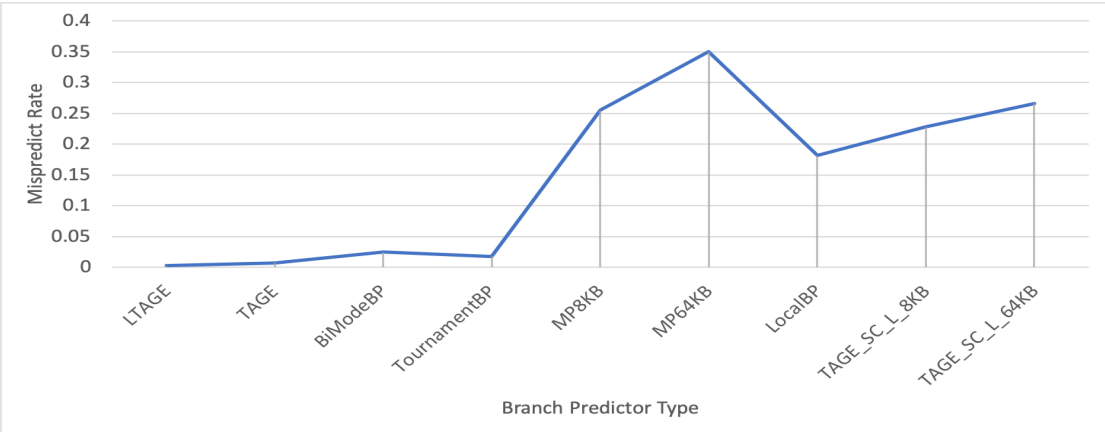
I have tested direct branch predictor of `'TAGE_SC_L_8KB', 'LocalBP', 'BiModeBP',`

'TAGE', 'MultiperspectivePerceptron64KB', 'MultiperspectivePerceptronTAGE8KB', 'TAGE_SC_L_64KB', 'TournamentBP', 'LTAGE'.

The results are as follows: (MP stands for MultiperspectivePerceptron)



This image shows that the different simulation time with the same programs in different branch predictor. As it shows, LTAGE, TAGE, BiModeBP and TournamentBP perform better than others. Their simulation time is around 0.01 seconds and LTAGE has the best performance. MultiperspectivePerceptron64KB has the poorest performance because its simulation time is the longest and it's near 0.04 seconds.



This image shows how the Mispredict Rate varies in different branch predictor types. Mispredict Rate is caculated by the parameters `system.cpu.commit.branchMispredicts` and `system.cpu.commit.branches`. `Mispredict Rate = branchMispredicts / branches.`

The trend of the this image is the same as that of above. LTAGE has the lowest Mispredict Rate while MultiperspectivePerceptron64KB has the highest. Less misprediction definitely leads to less execution time. Above all, LTAGE performs best when dealing with the program `gotrackcsv_to_csv`.