Assignment 2

Haiyu Wang

1. Configuration File:

   Assignment2.py

```
# Assignment 2, Haiyu Wang
# Implement Cache and test the performance

import m5
from m5.objects import *

import os
gem5_path = os.environ["GEM5"]

import optparse
parser = optparse.OptionParser()
parser.add_option("--prog", type="str", default=None);
parser.add_option("--clock_freq", type="str", default=None);
parser.add_option("--l1i_size",type="str",default=None);
parser.add_option("--l1i_assoc",type="int",default=None);
parser.add_option("--l1d_size",type="str",default=None);
parser.add_option("--l1d_assoc",type="int",default=None);
(options, args) = parser.parse_args()
program = options.prog

system = System()

system.clk_domain = SrcClockDomain()

system.clk_domain.voltage_domain = VoltageDomain()

isa = m5.defines.buildEnv['TARGET_ISA']
```

```
if options.clock_freq:
    system.clk_domain.clock = options.clock_freq
else:
    system.clk_domain.clock = '1.2GHz'

from Caches import *

system.mem_mode = 'timing'
system.mem_ranges = [AddrRange('512MB')]

system.cpu = TimingSimpleCPU()

system.membus = SystemXBar()

system.cpu.icache = L1ICache(options)
system.cpu.dcache = L1DCache(options)

system.cpu.icache.connectCPU(system.cpu)
system.cpu.dcache.connectCPU(system.cpu)

system.l2bus = L2XBar()
system.cpu.icache.connectBus(system.l2bus)
system.cpu.dcache.connectBus(system.l2bus)

system.l2cache = L2Cache(options)
system.l2cache.connectCPUSideBus(system.l2bus)
system.l2cache.connectMemSideBus(system.membus)

system.cpu.createInterruptController()
if isa == 'x86':
    system.cpu.interrupts[0].pio = system.membus.master
    system.cpu.interrupts[0].int_master = system.membus.slave
    system.cpu.interrupts[0].int_slave = system.membus.master
```

```
system.system_port = system.membus.slave

system.mem_ctrl = DDR3_1600_8x8()
system.mem_ctrl.range = system.mem_ranges[0]
system.mem_ctrl.port = system.membus.master

process = Process()
apps_path = "/project/linuxlab/gem5/test_progs"
if program == "daxpy" and isa == "x86":
    process.cmd = [apps_path + '/daxpy/daxpy_x86']
elif program == "daxpy" and isa == "arm":
    process.cmd = [apps_path + '/daxpy/daxpy_arm']
elif program == "queens" and isa == "x86":
    process.cmd = [apps_path + '/queens/queens_x86']
    process.cmd += ["10 -c"]
elif program == "queens" and isa == "arm":
    process.cmd = [apps_path + '/queens/queens_arm']
    process.cmd += ["10 -c"]

system.cpu.workload = process
system.cpu.createThreads()

root = Root(full_system = False, system = system)
m5.instantiate()
print ("Beginning simulation!")

exit_event = m5.simulate()
print('Exiting @ tick %i because %s' % (m5.curTick(),
exit_event.getCause()))
```

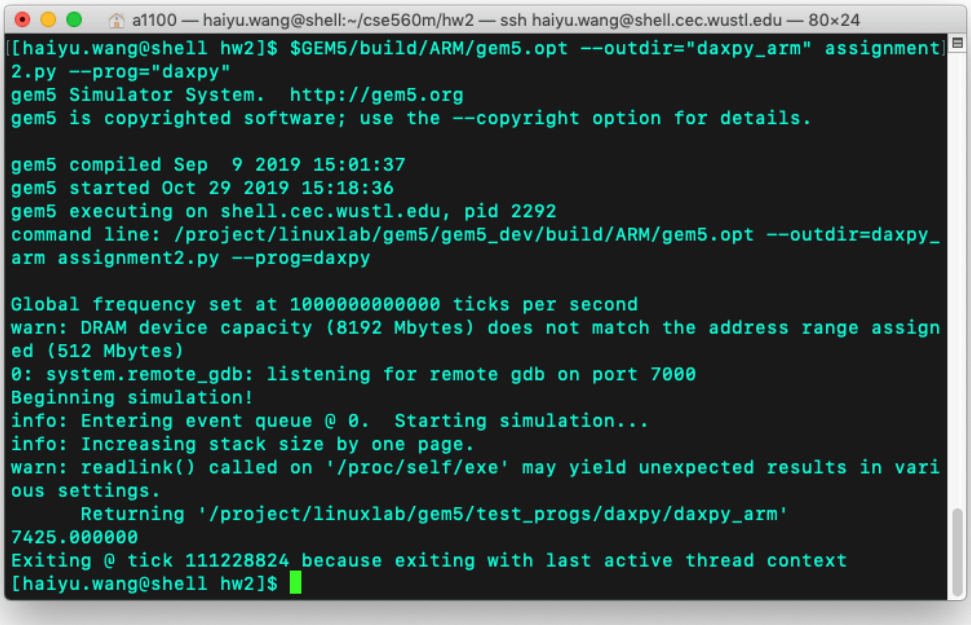
Caches.py

```
from m5.defines import buildEnv
from m5.objects import *

class L1Cache(Cache):
    assoc = 2
    tag_latency = 2
    data_latency = 2
    response_latency = 2
    mshrs = 4
    tgts_per_mshr = 20

    def __init__(self, options=None):
        super(L1Cache,self).__init__()
        pass
    def connectBus(self,bus):
        self.mem_side = bus.slave
    def connectCPU(self,cpu):
        raise NotImplementedError

class L1ICache(L1Cache):
    is_read_only = True
    writeback_clean = True
    size = '16kB'

    def __init__(self,opts=None):
        super(L1ICache,self).__init__(opts)
        if opts.l1i_size:
            self.size = opts.l1i_size
        if opts.l1i_assoc:
            self.assoc = opts.l1i_assoc

    def connectCPU(self, cpu):
```

```
        self.cpu_side = cpu.icache_port

class L1DCache(L1Cache):
    is_read_only = False
    writeback_clean = False
    size = '64kB'

    def __init__(self,opts=None):
        super(L1DCache,self).__init__(opts)
        if opts.l1d_size:
            self.size = opts.l1d_size
        if opts.l1d_assoc:
            self.assoc = opts.l1d_assoc

    def connectCPU(self, cpu):
        self.cpu_side = cpu.dcache_port

class L2Cache(Cache):
    is_read_only = False
    writeback_clean = False
    size = '256kB'
    assoc = 8
    tag_latency = 20
    data_latency = 20
    response_latency = 80
    mshrs = 20
    tgts_per_mshr = 12

    def __init__(self,options=None):
        super(L2Cache,self).__init__()
        pass

    def connectCPUSideBus(self,bus):
```

```
        self.cpu_side = bus.master

    def connectMemSideBus(self,bus):

        self.mem_side = bus.slave
```

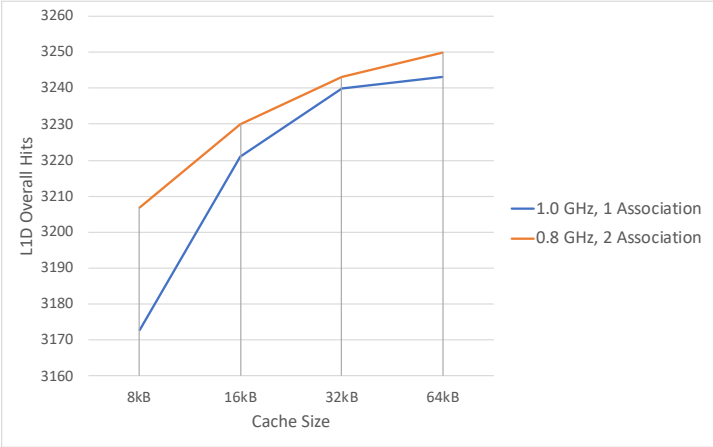
2. Console Output:



3. Graphs and Tables
   a) L1D Overall Hits
      i. Table

L1D Overall Hits		
	1.0 GHz, 1 Association	0.8 GHz, 2 Association
8kB	3173	3207
16kB	3221	3230
32kB	3240	3243
64kB	3243	3250

      ii. Graph

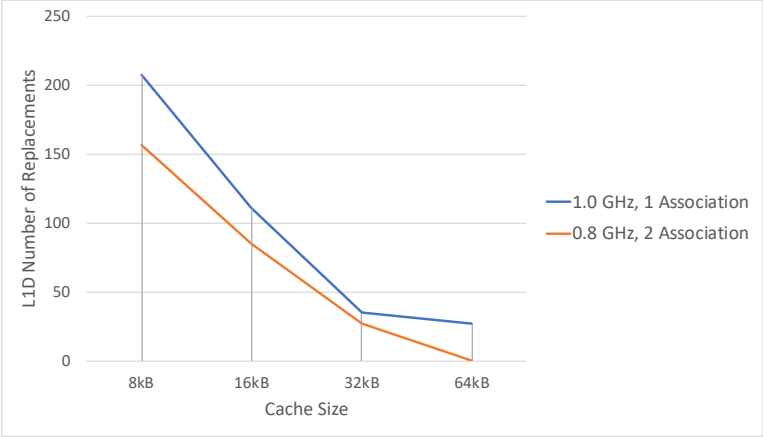


b) L1D Number of Replacements

i. Table

L1D Number of Replacements		
	1.0 GHz, 1 Association	0.8 GHz, 2 Association
8kB	208	157
16kB	111	85
32kB	35	27
64kB	27	0

ii. Graph

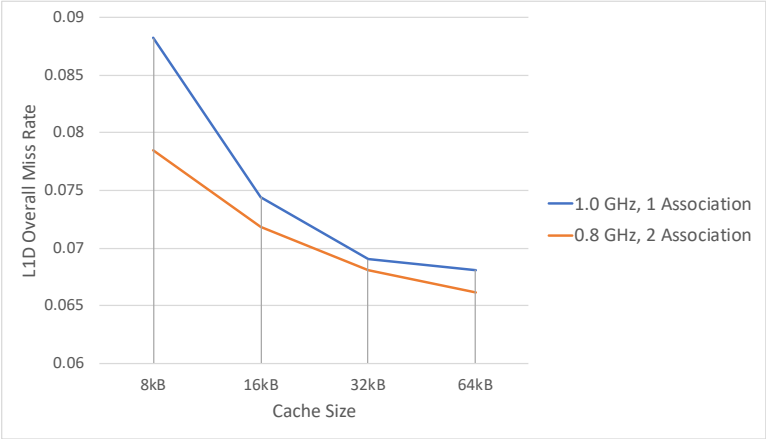


c) L1D Overall Miss Rate

i. Table

L1D Overall Miss Rate		
	1.0 GHz, 1 Association	0.8 GHz, 2 Association
8kB	0.088218	0.078448
16kB	0.074425	0.071839
32kB	0.068996	0.068103
64kB	0.068103	0.066092

ii. Graph

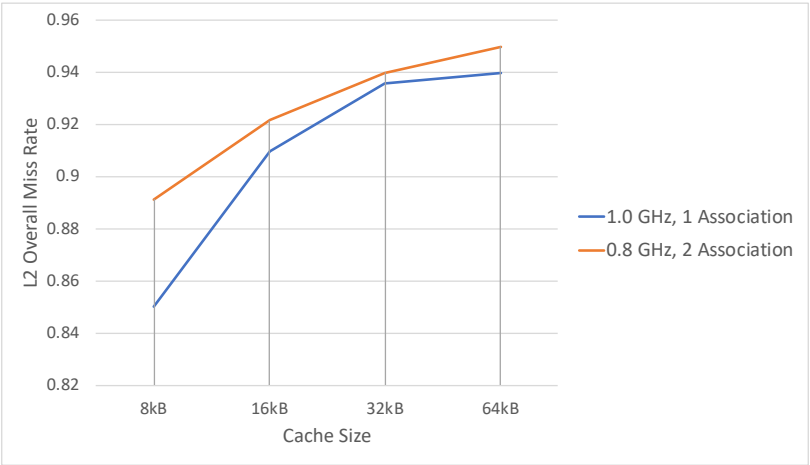


d) L2 Overall Miss Rate

i. Table

L2 Overall Miss Rate		
	1.0 GHz, 1 Association	0.8 GHz, 2 Association
8kB	0.850136	0.891429
16kB	0.909621	0.921713
32kB	0.935532	0.939759
64kB	0.939759	0.949772

ii. Graph



4. Answers

CPI = number of Cycles / number of instructions.

Case 1:

1.0 GHz, 1 Association, 8KB: CPI = 13.84

1.0 GHz, 1 Association, 16KB: CPI = 13.72

In this case, the cache size doubles, and the instructions committed remain, but the number of CPU cycles in the bigger cache drops (because it can cache

more data, so the number of hits increase which leads to decreasing of number of CPU cycles), so CPI drops.

Case 2:

1.0 GHz, 1 Association, 8KB: CPI = 13.84

0.8 GHz, 2 Association, 8KB: CPI = 13.08

In this case, although the case sizes are the same, the latter one has 2 associations which means there are more hits when running the programs. Thus, the total cycles will decrease, which leads to the decreasing of CPI