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

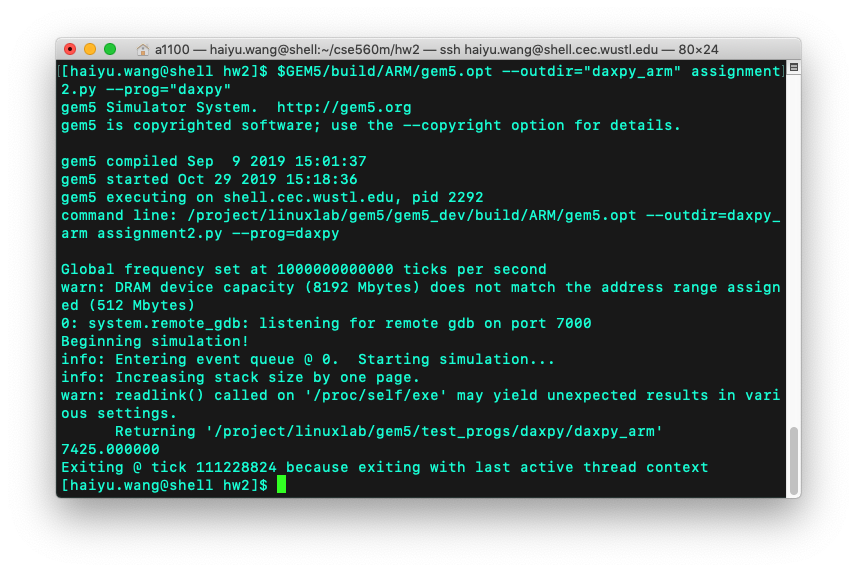
def connectCPUSideBus(self,bus):

self.cpu\_side = bus.master

def connectMemSideBus(self,bus):

self.mem\_side = bus.slave

1. Console Output:



1. Graphs and Tables
   1. L1D Overall Hits
      1. Table



* + 1. Graph



* 1. L1D Number of Replacements
     1. Table



* + 1. Graph



* 1. L1D Overall Miss Rate
     1. Table



* + 1. Graph



* 1. L2 Overall Miss Rate
     1. Table



* + 1. Graph



1. 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