EEL 5764 Computer Architecture Fall 2021

Department of Electrical and Computer Engineering

University of Florida

Lab 5

Student Name : Mohit Palliyil Sathyaseelan

Part A(1)

```
import m5
from m5.objects import *
from caches import *

import argparse

parser = argparse.ArgumentParser(description='A simple system with 2-level cache.')
parser.add_argument("binary", default="", nargs="?", type=str,
                    help="Path to the binary to execute.")
parser.add_argument("--l1i_size",
                    help=f"L1 instruction cache size. Default: 16kB.")
parser.add_argument("--l1d_size",
                    help="L1 data cache size. Default: Default: 64kB.")
parser.add_argument("--l2_size",
                    help="L2 cache size. Default: 256kB.")
parser.add_argument("--l1i_assoc",
                    help="l1i_assoc Default: 2.")
parser.add_argument("--numIQEntries",help="number of instruction queue entries.
Default: 64.")
parser.add_argument("--numROBEntries", help="number of load queue entries. Default:
192.")
parser.add_argument("--LQEntries", help="number of reorder buffer entries. Default:
32.")
options = parser.parse_args()


system = System()
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = '1GHz'
system.clk_domain.voltage_domain = VoltageDomain()
system.mem_mode = 'timing'
system.mem_ranges = [AddrRange('512MB')]
system.cpu = O3CPU()
if options.numIQEntries:
    system.cpu.numIQEntries = options.numIQEntries
if options.numROBEntries:
    system.cpu.numROBEntries = options.numROBEntries
if options.LQEntries:
    system.cpu.LQEntries = options.LQEntries
```

```
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.membus = SystemXBar()
system.l2cache.connectMemSideBus(system.membus)

system.cpu.createInterruptController()
system.cpu.interrupts[0].pio = system.membus.mem_side_ports
system.cpu.interrupts[0].int_requestor = system.membus.cpu_side_ports
system.cpu.interrupts[0].int_responder = system.membus.mem_side_ports
system.system_port = system.membus.cpu_side_ports
system.mem_ctrl = MemCtrl()
system.mem_ctrl.dram = DDR3_1600_8x8()
system.mem_ctrl.dram.range = system.mem_ranges[0]
system.mem_ctrl.port = system.membus.mem_side_ports
binary = 'configs/tutorial/matmul' # 'tests/test-progs/hello/bin/x86/linux/hello' #
# for gem5 V21 and beyond
system.workload = SEWorkload.init_compatible(binary)

process = Process()
process.cmd = [binary]
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 {} because {}'
        .format(m5.curTick(), exit_event.getCause()))
```

Part A(2):

numIQEntries	Exit Ticks
1	7822350000
32	789324000
48	789324000
64	781102000
96	781115000
128	781560000

LQEntries	Exit Ticks
1	2974951000
16	833621000
24	795116000
32	781102000
48	780957000
64	780957000

numROBEntries	Exit Ticks
1	14741980000
48	891235000
96	805393000
192	781102000
288	781113000
384	781116000

Part B(1):

A B C	Number of Ticks
0 0 0	1269530000
0 0 1	1273882000
0 1 0	1270186000
0 1 1	1274341000
1 0 0	1280746000
1 0 1	1285103000
1 1 0	1281081000
1 1 1	1285522000

Reversing Matrix A ,B and C (111) and Reversing Matrix A and C and leaving B in the original format (101) resulted in the highest number of ticks. The original format (000) resulted in the lowest number of ticks. Hence reversing the order of matrix initialization in matmul plays an important role in speed of execution.

The order of the cases in terms of the highest to the lowest number of ticks will be (111),(101),(110),(100),(011),(001),(010),(000)

Part B(2):

Optimization Level	Number of Ticks
O0	1269530000
O1	1191143000
O2	1155970000
O3	1142970000

As the optimization increases, the execution becomes faster. Optimization O3 is almost 1.11 times faster that optimization O0