Lab 5

Author: Everett Periman

Part A(1):

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

# Added Argparse for Lab4 to support cache changes from the terminal
import argparse

"""
SETUP NOTES
Our configuration script is going to model a very simple system.
We'll have just one simple CPU core.
This CPU core will be connected to a system-wide memory bus.
And we'll have a single DDR3 memory channel, also connected to the memory bus.
"""

"""
Useful Terminal Commands for this lab

1. This is used to recompile the modified hello.c file
    gcc -O0 -std=c99 -static -o a.out hello.c
2. This is how to run the simple.py program
    ../../../build/X86/gem5.opt simple.py
"""

# Create the parent for all of the children objects of the system
# This is creating the system as a whole and defines several characteristics
system = System()

# Setup the clock domain using the default value
# Setup the clock freq as 1GHz
# Setup the voltage domain as the default value
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = '1GHz'
system.clk_domain.voltage_domain = VoltageDomain()

# Setup the system using the timing memory mode, this is the most commonly used memory mode except
# for when fast-forwarding or restoring a simulation
# Define the memory range to be 512MB
system.mem_mode = 'timing'
system.mem_ranges = [AddrRange('512MB')]

# TimingSimpleCPU is the simplest cpu in gem5, this model executes each instruction in a single clock cycle
# Changed to the O3CPU (Lab5)
system.cpu = O3CPU()

"""
Inserting the new cache argparser code (Lab4)
"""

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 Association Parameter. Default: 2.")

# Add IQ, LQ, and ROB entrie parameters (Lab5)
parser.add_argument("--iq_entries",  help="Number of instruction queue entries. Default: 64.") # Param.Unsigned
parser.add_argument("--lq_entries",  help="Number of load queue entries. Default: 32.") # Param.Unsigned
parser.add_argument("--rob_entries",  help="Number of reorder buffer entries. Default: 192.") # Param.Unsigned

options = parser.parse_args()

"""
Insert new logic to modify the CPU properties
"""
if options.iq_entries:
    system.cpu.numIQEntries = options.iq_entries
if options.lq_entries:
    system.cpu.LQEntries = options.lq_entries
if options.rob_entries:
    system.cpu.numROBEntries = options.rob_entries

"""
Inserting the new cache code here
"""
# Create the new L1 caches
system.cpu.icache = L1ICache(options)
system.cpu.dcache = L1DCache(options)

# Attach the new caches to the cpu
system.cpu.icache.connectCPU(system.cpu)
system.cpu.dcache.connectCPU(system.cpu)

# Create a system wide memory bus
#system.membus = SystemXBar()

# These lines were removed when the new caches were added
# If there are no caches then the I-cache and the D-cache are connected directly to the membus
# This example has no caches
#system.cpu.icache_port = system.membus.cpu_side_ports
#system.cpu.dcache_port = system.membus.cpu_side_ports

"""
Inserting the new cache code here
"""
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)

# Create an IO controller on the CPU and connect it to the memory bus
# Need to connect a special port to the membus, this port allows the sys to read/write to memory
# Connecting PIO and interrupt ports to the membus is an X86 requirement
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

# Need to create a memory controller and connect it to the membus
# We are using a simple DDR3 controller
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

# Moved the binary selection code to the command line
if options.binary:
    binary = options.binary
else:
    binary = "hello"

# for gem5 V21 and beyond
system.workload = SEWorkload.init_compatible(binary)

process = Process()
process.cmd = [binary]
system.cpu.workload = process
system.cpu.createThreads()

# Instatiate the simulation and begin execution
root = Root(full_system = False, system = system)
m5.instantiate()

# Start the simulation
print("Beginning simulation!")
exit_event = m5.simulate()

# Inspect the state of the system after simulation
print('Exiting @ tick {} because {}'
      .format(m5.curTick(), exit_event.getCause()))

```

Part A(2):

numIQEntries	Number of ticks
1	1740727000
32	215399000
48	191046000
64	187681000
96	187681000
128	187681000

LQEntries	Number of ticks
1	999611000
16	221875000
24	194339000
32	187681000
48	187549000
64	187549000

numROBEntries	Number of ticks
1	3150472000
48	247485000
96	197370000
192	187681000
288	187549000
384	187549000

Part B(1):

A B C	Number of ticks
0 0 0	196777000
0 0 1	199791000
0 1 0	196872000
1 0 0	207582000
1 0 1	211481000
0 1 1	200191000
1 1 0	207844000
1 1 1	211617000

The negative effects of repeatedly having to load in larger arrays can be shown by these results. In the regular mode of operation (000) NRA is used to iterate over NCA for matrix A. This would result in loading 12 items from memory for each iteration of NRA. When it is inverted this requires the loading of 60 items for each iteration. This is significantly more expensive, and it is shown in the significant slow downs when A or C are reversed. When B is reversed the impact on performance is significantly less than A or C due to the similar size of the array lengths (12 vs 10) that are being swapped.

Part B(2):

Optimization Level	Number of ticks
O0	196777000
O1	121770000
O2	106493000
O3	103301000

The optimizations seem to have a significant positive affect in reducing the Number of ticks.