

Assignment 3: Part 2 Implementing and Analyzing Cache Configurations in gem5

Nischal Joshi

University of Cumberlands

MSCS - 531: Computer Architecture and Design

Dr. Charles Lively

October 6, 2024

Running x86 simulation

To simulate an x86 architecture, run the following command in the terminal:

```
scons build/X86/gem5.opt -j4
```

Once the simulation is completed successfully, the following message can be seen in the terminal:

```
Info: Compatible header file <linux/if_tun.h> not found.
Checking for backtrace_symbols_fd((void *)1, 0, 0) in C library None... (cached) yes
Checking for C header file fenv.h... (cached) yes
Checking for C header file png.h... (cached) no
Warning: Header file <png.h> not found.
This host has no libpng library.
Disabling support for PNG framebuffers.
Checking for clock_nanosleep(0,0,NULL,NULL) in C library None... (cached) no
Checking for clock_nanosleep(0,0,NULL,NULL) in C library rt... (cached) no
Warning: Can't find library for POSIX clocks.
Checking for C header file valgrind/valgrind.h... (cached) no
Checking for H5Fcreate("", 0, 0, 0) in C library hdf5... (cached) no
Warning: Couldn't find HDF5 C++ libraries. Disabling HDF5 support.
Checking for shm_open("/test", 0, 0) in C library None... (cached) yes
Warning: Cannot enable KVM, host seems to lack KVM support
Checking whether __i386__ is declared... (cached) no
Checking whether __x86_64__ is declared... (cached) no
Warning: Unrecognized architecture for systemc.
scons: done reading SConscript files.
scons: Building targets ...
[VER TAGS] -> X86/sim/tags.cc
scons: 'build/X86/gem5.opt' is up to date.
scons: done building targets.
*** Summary of Warnings ***
```

Default cache configuration

The default cache configuration for the simulation is given below. By default, the size and associativity of L1 Caches are set to 32KB and 2, whereas for the L2 Cache, it is 256KB and 8.

```
from m5.objects import Cache

class L1Cache(Cache):
    def __init__(self, size="32kB", assoc=2):
        super().__init__()
        self.size = size # Configurable from outside
        self.assoc = assoc # Configurable from outside
        self.tag_latency = 2
        self.data_latency = 2
        self.response_latency = 2
        self.mshrs = 4
        self.tgts_per_mshr = 20

    def connectBus(self, bus):
        self.mem_side = bus.cpu_side_ports

    def connectCPU(self, cpu):
        raise NotImplementedError

class L2ICache(L1Cache):
    def __init__(self, size="32kB", assoc=2):
        super().__init__(size=size, assoc=assoc)

    def connectCPU(self, cpu):
        self.cpu_side = cpu.icache_port
```

```

class L1DCache(L1Cache):
    def __init__(self, size="32kB", assoc=2):
        super().__init__(size=size, assoc=assoc)

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

class L2Cache(Cache):
    def __init__(self, size="256kB", assoc=8):
        super().__init__()
        self.size = size
        self.assoc = assoc # Configurable from outside
        self.tag_latency = 20 # Fixed internal parameters
        self.data_latency = 20
        self.response_latency = 20
        self.mshrs = 20
        self.tgts_per_mshr = 12

    def connectCPUSideBus(self, bus):
        self.cpu_side = bus.mem_side_ports

    def connectMemSideBus(self, bus):
        self.mem_side = bus.cpu_side_ports

```

The Python script in the gem5 file “two_level.py” was replicated to run a two-level cache hierarchy.

```

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

#path for the binary
binary = "tests/test-progs/hello/bin/x86/linux/hello"

system = System()
# Set the clock frequency of the system (and all of its children)
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = "1GHz"
system.clk_domain.voltage_domain = VoltageDomain()
# Set up the system
system.mem_mode = "timing" # Use timing accesses
system.mem_ranges = [AddrRange("512MB")] # Create an address range
# Create a simple CPU

```

```

# Create a simple CPU
system.cpu = X86TimingSimpleCPU()
# Create an L1 instruction and data cache
system.cpu.icache = L1ICache()
system.cpu.dcache = L1DCache()
# Connect the instruction and data caches to the CPU
system.cpu.icache.connectCPU(system.cpu)
system.cpu.dcache.connectCPU(system.cpu)
# Create a memory bus, a coherent crossbar, in this case
system.l2bus = L2XBar()
# Hook the CPU ports up to the l2bus
system.cpu.icache.connectBus(system.l2bus)
system.cpu.dcache.connectBus(system.l2bus)
# Create an L2 cache and connect it to the l2bus
system.l2cache = L2Cache()
system.l2cache.connectCPUSideBus(system.l2bus)
# Create a memory bus
system.membus = SystemXBar()
# Connect the L2 cache to the membus
system.l2cache.connectMemSideBus(system.membus)
# create the interrupt controller for the CPU
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
# Connect the system up to the membus
system.system_port = system.membus.cpu_side_ports
# Create a DDR3 memory 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

```

To calculate essential performance metrics like hit rate, miss rate, and average memory access latency, we need to use the statistical data updated in the stats.txt file.

```

system.cpu.dcache.demandMisses::total 1890
system.cpu.dcache.overallHits::cpu.data 1890
system.cpu.dcache.overallHits::total 1890
system.cpu.dcache.demandMisses::cpu.data 135

```

```
system.cpu.dcache.overallMisses::cpu.data      135
system.cpu.dcache.overallMisses::total        135
system.cpu.dcache.demandMissLatency::cpu.data  14450000
```

```
system.cpu.dcache.overallMissLatency::cpu.data 14450000
system.cpu.dcache.overallMissLatency::total    14450000
system.cpu.dcache.demandAccesses::cpu.data     2025
```

```
nischaljsh@Nischals-MBP gem5 % build/X86/gem5.opt configs/assignment3/cache_experiment.py
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 24.0.0.1
gem5 compiled Sep 13 2024 16:30:05
gem5 started Oct  1 2024 16:36:04
gem5 executing on Nischals-MBP, pid 16141
command line: build/X86/gem5.opt configs/assignment3/cache_experiment.py

Global frequency set at 100000000000 ticks per second
src/mem/dram_interface.cc:692: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any
system.remote_gdb: Listening for connections on port 7000
Beginning simulation!
src/sim/simulate.cc:199: info: Entering event queue @ 0.  Starting simulation...
Hello world!
Exiting @ tick 56435000 because exiting with last active thread context
nischaljsh@Nischals-MBP gem5 %
```

The data cache (dcache) details are being used for this instance.

Total overall hits = 1890

Total overall miss = 135

Total cache access = 2025 i.e. (1890 hits+ 135 misses)

Corresponding hit rate = $1890/2025 = 0.93\%$

Corresponding miss rate = 0.0667, i.e. (1-0.93)

Average memory access latency = 29111.95

The execution was completed in 56435000 ticks.

Optimizing Cache Parameters

- **Increasing the cache size:** Increasing the cache size can reduce misses. So, the cache size of L1Cache was updated to 64 KB, respectively. However, similar performance

metrics were captured for all caches in the stats.txt. Given below are the recorded metrics for this configuration.

Hit rate = 0.933%,

Miss rate = 0.0667%

Average memory access latency = 29111.95.

The execution was completed in 56435000 ticks.

```
# L1 Instruction Cache Class
class L1ICache(L1Cache):
    def __init__(self, size="32kB", assoc=2):
        super().__init__(size=size, assoc=assoc)

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

# L1 Data Cache Class
class L1DCache(L1Cache):
    def __init__(self, size="64kB", assoc=2):
        super().__init__(size=size, assoc=assoc)

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

```
system.cpu.dcache.demandHits::cpu.data 1890 # number of demand (
system.cpu.dcache.demandHits::total 1890 # number of demand (
system.cpu.dcache.overallHits::cpu.data 1890 # number of overall
system.cpu.dcache.overallHits::total 1890 # number of overall
system.cpu.dcache.demandMisses::cpu.data 135 # number of demand (
system.cpu.dcache.demandMisses::total 135 # number of demand (
system.cpu.dcache.overallMisses::cpu.data 135 # number of overall
system.cpu.dcache.overallMisses::total 135 # number of overall
system.cpu.dcache.demandMissLatency::cpu.data 14546000 # number of dema
system.cpu.dcache.demandMissLatency::total 14546000 # number of demand
system.cpu.dcache.overallMissLatency::cpu.data 14546000 # number of ov
system.cpu.dcache.overallMissLatency::total 14546000 # number of overa
system.cpu.dcache.demandAccesses::cpu.data 2025 # number of demand
system.cpu.dcache.demandAccesses::total 2025 # number of demand (
system.cpu.dcache.overallAccesses::cpu.data 2025 # number of overa
system.cpu.dcache.overallAccesses::total 2025 # number of overall
system.cpu.dcache.demandMissRate::cpu.data 0.066667 # miss rate for dem
system.cpu.dcache.demandMissRate::total 0.066667 # miss rate for dema
system.cpu.dcache.overallMissRate::cpu.data 0.066667 # miss rate for ov
system.cpu.dcache.overallMissRate::total 0.066667 # miss rate for over
system.cpu.dcache.demandAvgMissLatency::cpu.data 107748.148148 # average o
system.cpu.dcache.demandAvgMissLatency::total 107748.148148 # average over
system.cpu.dcache.overallAvgMissLatency::cpu.data 107748.148148 # average
system.cpu.dcache.overallAvgMissLatency::total 107748.148148 # average ove
```

- **Increasing associativity:** In this case, the associativity of L1ICache and L1DCache was increased by two times, making it 4, and L2 caches updated to 16. For this case, the

associativity was passed an argument in cache instances. Similarly to the cache size change, no change in the performance metrics was recorded in the stats.txt. The measured metrics for the data cache are as follows:

Hit rate = 0.933%

Miss rate = 0.0667%

Average memory access latency = 29111.95.

Likewise, the execution was completed in 56435000 ticks.

```
system.cpu.icache = L1ICache(assoc=4)
system.cpu.dcache = L1DCache(assoc=4)
# Connect the instruction and data caches to the CPU
system.cpu.icache.connectCPU(system.cpu)
system.cpu.dcache.connectCPU(system.cpu)
# Create a memory bus, a coherent crossbar, in this case
system.l2bus = L2XBar()
# Hook the CPU ports up to the l2bus
system.cpu.icache.connectBus(system.l2bus)
system.cpu.dcache.connectBus(system.l2bus)
# Create an L2 cache and connect it to the l2bus
system.l2cache = L2Cache(assoc=8)
system.l2cache.connectCPUSideBus(system.l2bus)
# Create a memory bus
system.membus = SystemXBar(0)
# Connect the L2 cache to the membus
system.l2cache.connectMemSideBus(system.membus)
```

```
system.cpu.dcache.demandHits::cpu.data      1890      # number of deman
system.cpu.dcache.demandHits::total          1890      # number of deman
system.cpu.dcache.overallHits::cpu.data      1890      # number of overa
system.cpu.dcache.overallHits::total          1890      # number of overa
system.cpu.dcache.demandMisses::cpu.data     135        # number of deman
system.cpu.dcache.demandMisses::total        135        # number of deman
system.cpu.dcache.overallMisses::cpu.data     135        # number of over
system.cpu.dcache.overallMisses::total        135        # number of overa
system.cpu.dcache.demandMissLatency::cpu.data 14546000      # number of
system.cpu.dcache.demandMissLatency::total    14546000      # number of dem
system.cpu.dcache.overallMissLatency::cpu.data 14546000      # number of
system.cpu.dcache.overallMissLatency::total    14546000      # number of ov
system.cpu.dcache.demandAccesses::cpu.data    2025        # number of dem
system.cpu.dcache.demandAccesses::total       2025        # number of deman
system.cpu.dcache.overallAccesses::cpu.data    2025        # number of ov
system.cpu.dcache.overallAccesses::total       2025        # number of overa
system.cpu.dcache.demandMissRate::cpu.data    0.066667     # miss rate for
system.cpu.dcache.demandMissRate::total       0.066667     # miss rate for d
system.cpu.dcache.overallMissRate::cpu.data    0.066667     # miss rate fo
system.cpu.dcache.overallMissRate::total       0.066667     # miss rate for o
system.cpu.dcache.demandAvgMissLatency::cpu.data 107748.148148  # averag
system.cpu.dcache.demandAvgMissLatency::total 107748.148148  # average o
system.cpu.dcache.overallAvgMissLatency::cpu.data 107748.148148  # avera
system.cpu.dcache.overallAvgMissLatency::total 107748.148148  # average
```

- **Changing the block size:** The cache block size is updated globally using the script

"system.cache_line_size = 128". A visible change in performance metrics was noticed.

```
# Set the clock frequency of the system (and all of its children)
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = "1GHz"
system.clk_domain.voltage_domain = VoltageDomain()
# Set up the system
system.mem_mode = "timing" # Use timing accesses
system.mem_ranges = [AddrRange("512MB")] # Create an address range
# Create (function) cache_line_size: Literal[128]
system.cache_line_size = 128 # Set block size to 128 bytes
# Create an L1 instruction and data cache
system.cpu.icache = L1ICache()
system.cpu.dcache = L1DCache()
# Connect the instruction and data caches to the CPU
system.cpu.icache.connectCPU(system.cpu)
system.cpu.dcache.connectCPU(system.cpu)
# Create a memory bus, a coherent crossbar, in this case
system.l2bus = L2XBar()
# Hook the CPU ports up to the l2bus
system.cpu.icache.connectBus(system.l2bus)
system.cpu.dcache.connectBus(system.l2bus)
```

```
system.cpu.commitStats0.committedControl::IsCondControl      986          # Class of control type instructions committed (Count)
system.cpu.commitStats0.committedControl::IsUncondControl    320          # Class of control type instructions committed (Count)
system.cpu.commitStats0.committedControl::IsCall             112          # Class of control type instructions committed (Count)
system.cpu.commitStats0.committedControl::IsReturn           109          # Class of control type instructions committed (Count)
system.cpu.dcache.demandHits::cpu.data                       1936         # number of demand (read+write) hits (Count)
system.cpu.dcache.demandHits::total                           1936         # number of demand (read+write) hits (Count)
system.cpu.dcache.overallHits::cpu.data                       1936         # number of overall hits (Count)
system.cpu.dcache.overallHits::total                          1936         # number of overall hits (Count)
system.cpu.dcache.demandMisses::cpu.data                      89           # number of demand (read+write) misses (Count)
system.cpu.dcache.demandMisses::total                         89           # number of demand (read+write) misses (Count)
system.cpu.dcache.overallMisses::cpu.data                     89           # number of overall misses (Count)
system.cpu.dcache.overallMisses::total                        89           # number of overall misses (Count)
system.cpu.dcache.demandMissLatency::cpu.data                10071000     # number of demand (read+write) miss ticks (Tick)
system.cpu.dcache.demandMissLatency::total                    10071000     # number of demand (read+write) miss ticks (Tick)
system.cpu.dcache.overallMissLatency::cpu.data                10071000     # number of overall miss ticks (Tick)
system.cpu.dcache.overallMissLatency::total                   10071000     # number of overall miss ticks (Tick)
system.cpu.dcache.demandAccesses::cpu.data                   2025         # number of demand (read+write) accesses (Count)
system.cpu.dcache.demandAccesses::total                      2025         # number of demand (read+write) accesses (Count)
system.cpu.dcache.overallAccesses::cpu.data                   2025         # number of overall (read+write) accesses (Count)
system.cpu.dcache.overallAccesses::total                      2025         # number of overall (read+write) accesses (Count)
system.cpu.dcache.demandMissRate::cpu.data                   0.043951     # miss rate for demand accesses (Ratio)
system.cpu.dcache.demandMissRate::total                      0.043951     # miss rate for demand accesses (Ratio)
system.cpu.dcache.overallMissRate::cpu.data                   0.043951     # miss rate for overall accesses (Ratio)
system.cpu.dcache.overallMissRate::total                      0.043951     # miss rate for overall accesses (Ratio)
system.cpu.dcache.demandAvgMissLatency::cpu.data             113157.303371 # average overall miss latency in ticks ((Tick/Count))
system.cpu.dcache.demandAvgMissLatency::total                113157.303371 # average overall miss latency in ticks ((Tick/Count))
system.cpu.dcache.overallAvgMissLatency::cpu.data             113157.303371 # average overall miss latency ((Tick/Count))
system.cpu.dcache.overallAvgMissLatency::total                113157.303371 # average overall miss latency ((Tick/Count))
system.cpu.dcache.blockedCycles::no_mshrs                     0            # number of cycles access was blocked (Cycle)
```

Similar to other instances, we will use the data cache's details (dcache).

Total overall hits = 1936

Total overall miss = 89

Total cache access = 2025 i.e. (1936 hits + 89 misses)

Corresponding hit rate = $1936/2025 = 0.9560\%$

Corresponding miss rate = 0.044, i.e. (1-0.9560)

Average memory access latency = 33867.90

The above observation shows that the hit-and-miss rates have improved after increasing the cache block size. However, the average memory access latency has increased.

Analysis of the Performance Metrics

Given below is a summary table to show the result of performance metrics after updating different cache configurations for the L1D cache:

Cache Parameters	Cache Size (L1D)	Associativity (L1)	Block Size	Hit Rate (dcache)	Miss Rate (dcache)	Average Memory Access latency
Default configuration	32KB	2	64 bytes	0.933%	0.0667%	29111.95
Increased Cache Size	64KB	2	64 bytes	0.933%	0.0667%	29111.95
Increased associativity	32KB	4	64 bytes	0.933%	0.0667%	29111.95
Increase Block size	32KB	4	128 bytes	0.9560%	0.044.	33867.95

Default Configuration: The default configuration serves as the baseline for this analysis. The cache size is moderately tiny, 32KB, with 2-way associativity, and the block size is 64. Regardless of baseline configuration, the miss rate is relatively low 0.0667%. The average memory latency for this configuration is 29911.95

Increased Cache Size: Despite increasing cache size by two times, no significant change in the performance metrics was noted. Both the hit rate and miss rate remained consistent with the

default configuration. It implies that the working set, which in this case was the Hello World program, needed to be more significant to fit in a 32 KB cache, and increasing the size did not provide any significant benefit.

Increased Associativity: Increasing associativity typically helps reduce conflict misses; however, no visible changes in the performance metrics were recorded after doubling the associativity. This implies that there were no conflict misses, to begin with.

Increased Block Size: In this case, the block size of the cache was increased to 128 bytes. An improvement of 0.02% was visible in the hit rate, which went from 0.93% to 0.956%. However, there was a significant increase in the average memory access latency (33867.95). The extensive block results in few cache lines and more extended memory fetches.

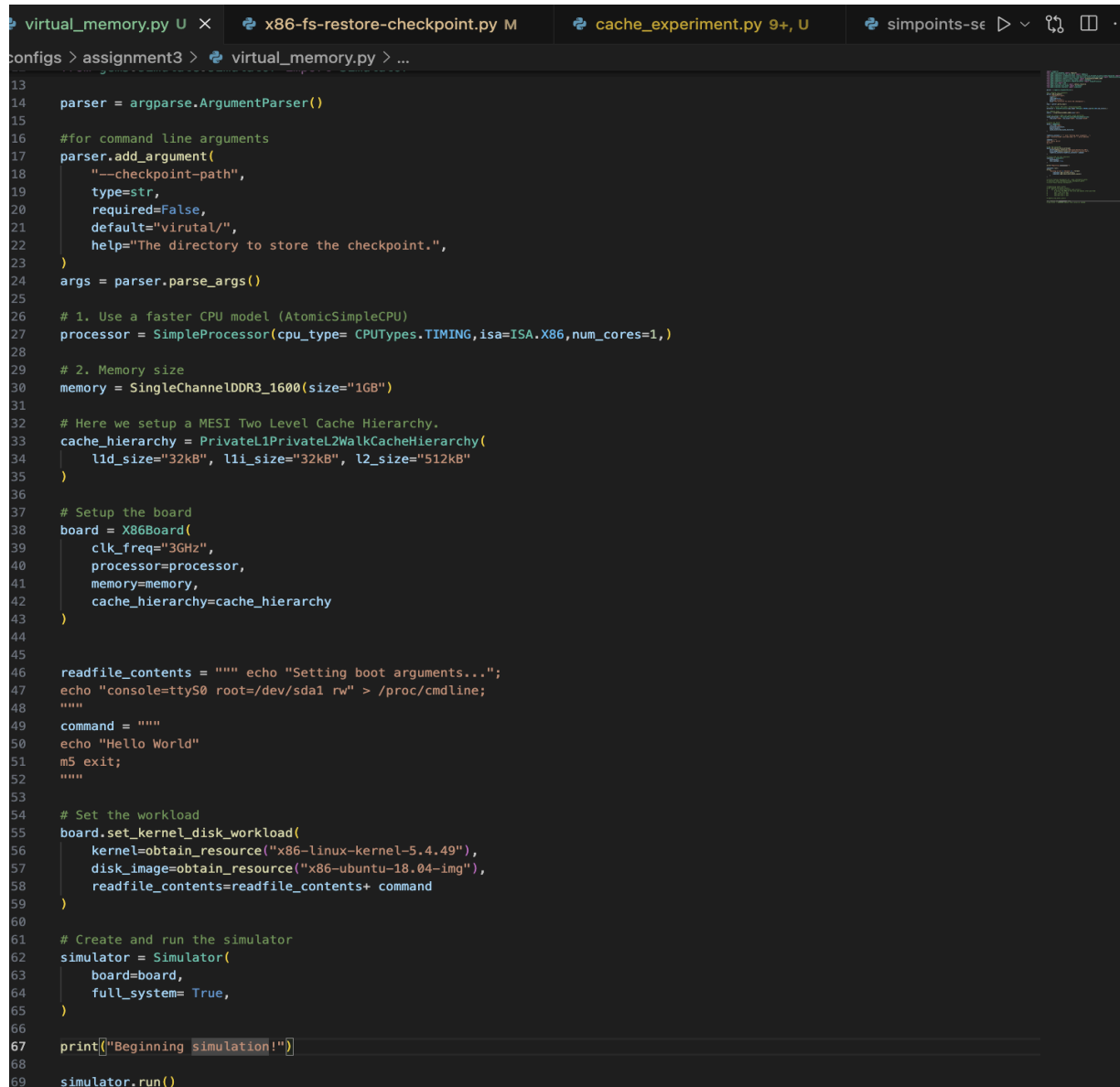
In a general scenario, advanced cache optimization techniques like perfecting and victim caches can be implemented to improve performance metrics. However, in this case, the increase in cache size and associativity did not improve the performance, indicating that perfecting and victim cache will not significantly impact.

In general, advanced cache optimization techniques like perfecting and victim caches can be implemented to improve performance metrics. However, in this case, the increase in cache size and associativity did not improve the performance, indicating that perfecting and victim cache will not significantly impact. A similar pattern can be seen in the performance metrics of the L1T cache and L2 Cache.

Virtual Memory Exploration

Virtual memory is a fundamental concept in modern computer architecture and design that allows processes to use more memory than is available by utilizing the additional space from RAM or secondary storage. Although SE mode creates virtual memory, it does not allow

changing specific configurations like page size and Transitional Lookaside Buffer(TLB) configuration. We need to simulate these attributes in full system mode to manage them. The Python code for virtual_memory.py is below to execute a simple FS mode simulation.



```

13
14 parser = argparse.ArgumentParser()
15
16 #for command line arguments
17 parser.add_argument(
18     "--checkpoint-path",
19     type=str,
20     required=False,
21     default="virtual/",
22     help="The directory to store the checkpoint.",
23 )
24 args = parser.parse_args()
25
26 # 1. Use a faster CPU model (AtomicSimpleCPU)
27 processor = SimpleProcessor(cpu_type= CPUTypes.TIMING, isa=ISA.X86, num_cores=1,)
28
29 # 2. Memory size
30 memory = SingleChannelDDR3_1600(size="1GB")
31
32 # Here we setup a MESI Two Level Cache Hierarchy.
33 cache_hierarchy = PrivateL1PrivateL2WalkCacheHierarchy(
34     l1d_size="32kB", l1i_size="32kB", l2_size="512kB"
35 )
36
37 # Setup the board
38 board = X86Board(
39     clk_freq="3GHz",
40     processor=processor,
41     memory=memory,
42     cache_hierarchy=cache_hierarchy
43 )
44
45
46 readfile_contents = "" echo "Setting boot arguments...";
47 echo "console=ttyS0 root=/dev/sda1 rw" > /proc/cmdline;
48 ""
49 command = ""
50 echo "Hello World"
51 m5 exit;
52 ""
53
54 # Set the workload
55 board.set_kernel_disk_workload(
56     kernel=obtain_resource("x86-linux-kernel-5.4.49"),
57     disk_image=obtain_resource("x86-ubuntu-18.04-img"),
58     readfile_contents=readfile_contents+ command
59 )
60
61 # Create and run the simulator
62 simulator = Simulator(
63     board=board,
64     full_system= True,
65 )
66
67 print(["Beginning simulation!"])
68
69 simulator.run()

```

The above script simulates a simple x86 system. First, a SimpleProcessor with a single TIMING CPU core and x86 ISA is created. Following that, we are configuring a SingleChannelDDR3_1600 memory of size 512 MB. A prebuilt cache hierarchy is used to

specify the system's L1 and L2 caches. The X86Board combines these all. The Linux kernel and Ubuntu disk images are available in the gem5 resources for the kernel and disk images. The workload defined creates a command to echo Hello World and exit the simulation. To run the simulation, use the following script in the terminal

```
./build/X86/gem5.opt configs/assignment3/virtual_memory.py
```

```
nischaljsh@Nischals-MBP gem5 % ./build/X86/gem5.opt configs/assignment3/virtual_memory.py
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 24.0.0.1
gem5 compiled Oct  3 2024 16:49:17
gem5 started Oct  5 2024 06:09:23
gem5 executing on Nischals-MBP, pid 87920
command line: ./build/X86/gem5.opt configs/assignment3/virtual_memory.py

info: Using default config
Beginning simulation!
Global frequency set at 100000000000 ticks per second
src/mem/dram_interface.cc:692: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (1024 GiB)
src/sim/kernel_workload.cc:46: info: kernel located at: /Users/nischaljsh/.cache/gem5/x86-linux-kernel-5.4.49
0: board.pc.south_bridge.cmos.rtc: Real-time clock set to Sun Jan  1 00:00:00 2012
board.pc.com_1.device: Listening for connections on port 3457
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group.
src/dev/intel_8254_timer.cc:128: warn: Reading current count from inactive timer.
board.remote_gdb: Listening for connections on port 7001
src/sim/simulate.cc:199: info: Entering event queue @ 0. Starting simulation...
Exiting @ tick 100 because simulate() limit reached.
nischaljsh@Nischals-MBP gem5 %
```

```
nischaljsh@Nischals-MBP gem5 % ./build/X86/gem5.opt configs/assignment3/virtual_memory.py
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 24.0.0.1
gem5 compiled Oct  3 2024 16:49:17
gem5 started Oct  4 2024 20:03:46
gem5 executing on Nischals-MBP, pid 79539
command line: ./build/X86/gem5.opt configs/assignment3/virtual_memory.py

info: Using default config
Global frequency set at 100000000000 ticks per second
"neato" with args ['-Tsvg', '/var/folders/py/r1rt0zjx5rddyldv4fhnyfw0000gn/T/tmp_9064qtx'] returned code: 1
stdout, stderr:
b''
b"Error: /var/folders/py/r1rt0zjx5rddyldv4fhnyfw0000gn/T/tmp_9064qtx: syntax error in line 3 near '.'\n"

warn: failed to generate dot output from m5out/config.board.cache_hierarchy.ruby_system.dot
src/mem/dram_interface.cc:692: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/sim/kernel_workload.cc:46: info: kernel located at: /Users/nischaljsh/.cache/gem5/x86-linux-kernel-5.4.49
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group.
gacy stat is deprecated.
0: board.pc.south_bridge.cmos.rtc: Real-time clock set to Sun Jan  1 00:00:00 2012
board.pc.com_1.device: Listening for connections on port 3456
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group.
gacy stat is deprecated.
src/dev/intel_8254_timer.cc:128: warn: Reading current count from inactive timer.
board.remote_gdb: Listening for connections on port 7000
src/sim/simulate.cc:199: info: Entering event queue @ 0. Starting simulation...
src/mem/ruby/system/Sequencer.cc:682: warn: Replacement policy updates recently became the responsibility of SLICC state machines. Make s
to setMRU() near callbacks in .sm files!
Simulation completed successfully!
Simulated ticks: 1000000
```

Once the simulation starts, we can access the terminal by connecting to the respective port. For this navigate to utils/term in the gem5 directory and run the following script in the terminal:

```
gcc -o m5term term.c
```

This will activate the terminal. Once activated we can run the following script to get access to the simulated Ubuntu terminal:

```
./m5term localhost 7000
```

```
[ OK ] Started Stop ureadahead data collection 45s after completed startup.
Starting Update UTMP about System Runlevel Changes...
[ OK ] Started Update UTMP about System Runlevel Changes.

Ubuntu 18.04.2 LTS gem5-host ttyS0
gem5-host login: root (automatic login)

Segmentation fault
Segmentation fault
Segmentation fault
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 5.4.49 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Starting gem5 init... reading run script file.
Running m5 script from /tmp/script
Setting boot arguments...
/tmp/script: line 3: echo: write error: Input/output error
Hello World
Done running script, exiting.
root@gem5-host:~#
```

Simulation in full system mode takes much time. So, a better approach to testing different configurations is to create a checkpoint. The updated `virtual_memroy.py` script is below to create a checkpoint for the kernel disk image and CPU configuration. Once the checkpoint is established, a respective directory is created with an `m5.cpt` file, and we can restore the simulation to add the respective changes on top of the saved checkpoint. To execute this checkpoint script, use the following command

```
./build/X86/gem5.opt configs/assignment3/virtual_memory.py
```

```
terminal.py 2, M  virtual_memory.py U  virtual_memory_restore_checkpoint.py U  cache_partitioning.py 9+
nfigs > assignment3 > virtual_memory.py > ...

5 parser.add_argument(
6     "--checkpoint-path",
7     type=str,
8     required=False,
9     default="virtual/",
10    help="The directory to store the checkpoint.",
11)
12
13 args = parser.parse_args()
14 # 1. Creating a simple processor
15 processor = SimpleProcessor(cpu_type= CPUTypes.TIMING, isa=ISA.X86, num_cores=1,)
16
17 # 2. Memory size
18 memory = SingleChannelDDR3_1600(size="1GB")
19
20 # Here we setup a MESI Two Level Cache Hierarchy.
21 cache_hierarchy = PrivateL1PrivateL2WalkCacheHierarchy(
22     l1d_size="32kB", l1i_size="32kB", l2_size="512kB"
23 )
24
25 # Setup the board
26 board = X86Board(
27     clk_freq="3GHz",
28     processor=processor,
29     memory=memory,
30     cache_hierarchy=cache_hierarchy
31 )
32
33 readfile_contents = """ echo "Setting boot arguments...";
34 echo "console=ttyS0 root=/dev/sda1 rw" > /proc/cmdline;
35 """
36
37 command = """
38 echo "Hello World"
39 m5 exit;
40 """
41
42 # Set the workload
43 board.set_kernel_disk_workload(
44     kernel=obtain_resource("x86-linux-kernel-5.4.49"),
45     disk_image=obtain_resource("x86-ubuntu-18.04-img"),
46     readfile_contents=readfile_contents+ command
47 )
48
49 # Create and run the simulator
50 simulator = Simulator(
51     board=board,
52     full_system= True,
53 )
54
55 print("Beginning simulation!")
56 simulator.run()
57
58 print(["Taking checkpoint at", args.checkpoint_path])
59 print(
60     "Exiting @ tick {} because {}".format(
61         simulator.get_current_tick(),
62         simulator.get_last_exit_event_cause()
63     )
64 )
```

Here, `simulator.save_checkpoint(agsr.checkpoint_path)` creates a checkpoint. Also, a folder named `virtual` is created in the root directory of the `gem5` after the successful creation of the checkpoint.

The performance metrics for this simulation are shown in the snapshot given below.

board.processor.cores.core.fetchStats0.numFetchSuspends	0	# Number of times Execute suspended instruction
board.processor.cores.core.interrupts.clk_domain.clock	5328	# Clock period in ticks (Tick)
board.processor.cores.core.mmu.dtb.rdAccesses	528811621	# TLB accesses on read requests (Count)
board.processor.cores.core.mmu.dtb.wrAccesses	338303719	# TLB accesses on write requests (Count)
board.processor.cores.core.mmu.dtb.rdMisses	1436270	# TLB misses on read requests (Count)
board.processor.cores.core.mmu.dtb.wrMisses	107921	# TLB misses on write requests (Count)
board.processor.cores.core.mmu.dtb.walker.power_state.pwrStateResidencyTicks::UNDEFINED	3109359688506	# Cumulative ti
board.processor.cores.core.mmu.itb.rdAccesses	0	# TLB accesses on read requests (Count)
board.processor.cores.core.mmu.itb.wrAccesses	2886202477	# TLB accesses on write requests (Count)
board.processor.cores.core.mmu.itb.rdMisses	0	# TLB misses on read requests (Count)
board.processor.cores.core.mmu.itb.wrMisses	354790	# TLB misses on write requests (Count)
board.processor.cores.core.mmu.itb.walker.power_state.pwrStateResidencyTicks::UNDEFINED	3109359688506	# Cumulative ti

Now, we can create a script to run the execution from the previous checkpoint. Below is the script for `virtual_memory_checkpoint_restore.py` that allows the simulation to continue from the last checkpoint.

```

virtual_memory_restore_checkpoint.py U • fs.py U
configs > assignment3 > virtual_memory_restore_checkpoint.py > ...
2 from gem5.components.boards.x86_board import X86Board
3 from gem5.components.cachehierarchies.classic.private_l1_private_l2_walk_cache_hierarchy import (
4     PrivateL1PrivateL2WalkCacheHierarchy,
5 )
6 from gem5.components.memory import SingleChannelDDR3_1600
7 from gem5.components.processors.cpu_types import CPUTypes
8 from gem5.components.processors.simple_processor import SimpleProcessor
9 from gem5.isas import ISA
10 from gem5.resources.resource import (
11     obtain_resource,
12 )
13 from gem5.simulate.simulator import Simulator
14 from gem5.utils.requires import requires
15
16 processor = SimpleProcessor(cpu_type=CPUTypes.O3, isa=ISA.X86, num_cores=1)
17
18 memory = SingleChannelDDR3_1600(size="1GB")
19
20 cache_hierarchy = PrivateL1PrivateL2WalkCacheHierarchy(
21     l1d_size="32kB", l1i_size="32kB", l2_size="512kB"
22 )
23
24 # Setup the board.
25 board = X86Board(
26     clk_freq="3GHz",
27     processor=processor,
28     memory=memory,
29     cache_hierarchy=cache_hierarchy,
30 )
31
32 # Set the Full System workload.
33 board.set_kernel_disk_workload([
34     kernel=obtain_resource("x86-linux-kernel-5.4.49"),
35     disk_image=obtain_resource("x86-ubuntu-18.04-img"),
36     checkpoint= Path("/Users/nischaljsh/Documents/gem5/virtual"),
37 ])
38
39 sim = Simulator(board=board, full_system=True)
40 print("Restoring from checkpoint and beginning simulation!")
41 sim.run()
42 print(
43     "Exiting @ tick {} because {}.".format(
44         sim.get_current_tick(),
45         sim.get_last_exit_event_cause()
46     )
47 )

```

The checkpoint path is provided via the `checkpoint` parameter in the `set_kernel_disk_workload` method.

Now, the script can be updated to change the data and instruction table buffer size. Here, the `modify_tlb_sizes` function is used to modify the data and instruction transitional buffer sizes. To change the TLB size, we need access to the system's MMU and change the size accordingly. I updated the size to 512.

```
virtual_memory_restore_checkpoint.py U ●
onfigs > assignment3 > virtual_memory_restore_checkpoint.py > ...
3 from gems.simulate.simulator import Simulator
4 from gems.utils.requires import requires
5
6 processor = SimpleProcessor(cpu_type=CPUTypes.03, isa=ISA.X86, num_cores=1)
7
8 memory = SingleChannelDDR3_1600(size="1GB")
9
10 cache_hierarchy = PrivateL1PrivateL2WalkCacheHierarchy(
11     l1d_size="32kB", l1i_size="32kB", l2_size="512kB"
12 )
13
14 # Setup the board.
15 board = X86Board(
16     clk_freq="3GHz",
17     processor=processor,
18     memory=memory,
19     cache_hierarchy=cache_hierarchy,
20 )
21
22 # Set the Full System workload.
23 board.set_kernel_disk_workload([
24     kernel=obtain_resource("x86-linux-kernel-5.4.49"),
25     disk_image=obtain_resource("x86-ubuntu-18.04-img"),
26     checkpoint= Path("/Users/nischaljsh/Documents/gem5/virtual"),
27 ])
28
29 # Modify TLB sizes after board initialization
30 def modify_tlb_sizes(board):
31     for core in board.get_processor().get_cores():
32         # Modify ITLB (Instruction TLB) size
33         core.get_mmu().itb.size = 512 # Set to desired size
34         core.get_mmu().dtb.size = 512 # Set to desired size
35
36         print(f" ITLB size: {core.get_mmu().itb.size}")
37         print(f" DTLB size: {core.get_mmu().dtb.size}")
38
39 # Call the function to modify TLB sizes
40 modify_tlb_sizes(board)
41 sim = Simulator(board=board, full_system=True)
42 print("Restoring from checkpoint and beginning simulation!")
43 sim.run()
44 print(
45     "Exiting @ tick {} because {}".format(
46         sim.get_current_tick(),
47         sim.get_last_exit_event_cause()
48     )
49 )
50
51
```


Issue Encountered

Although checkpoint is known to speed up the simulation process, it took much time for the bootup to complete in my system. An alternative approach to speed up the bootup time is to use KVM. However, macOS is impossible.

```
nischaljsh@Nischals-MBP gem5 % ./build/X86/gem5.opt configs/assignment3/virtual_memory_restore_checkpoint.py
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 24.0.0.1
gem5 compiled Oct  3 2024 16:49:17
gem5 started Oct  6 2024 06:39:00
gem5 executing on Nischals-MBP, pid 15704
command line: ./build/X86/gem5.opt configs/assignment3/virtual_memory_restore_checkpoint.py

info: Using default config
      ITLB size: 512
      DTLB size: 512
Restoring from checkpoint and beginning simulation!
Global frequency set at 1000000000000 ticks per second
src/mem/dram_interface.cc:692: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (1024
src/sim/kernel_workload.cc:46: info: kernel located at: /Users/nischaljsh/.cache/gem5/x86-linux-kernel-5.4.49
      0: board.pc.south_bridge.cmos rtc: Real-time clock set to Sun Jan  1 00:00:00 2012
board.pc.com_1.device: Listening for connections on port 3456
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any s
p. Legacy stat is deprecated.
src/dev/intel_8254_timer.cc:128: warn: Reading current count from inactive timer.
board.remote_gdb: Listening for connections on port 7000
src/sim/simulate.cc:199: info: Entering event queue @ 3111482391436. Starting simulation...
build/x86/arch/x86/generated/exec-ns.cc.inc:27: warn: instruction 'fwait' unimplemented
src/arch/generic/debugfaults.hh:145: warn: MOVNTDQ: Ignoring non-temporal hint, modeling as cacheable!
src/dev/x86/pc.cc:117: warn: Don't know what interrupt to clear for console.
288619833140451: board.pc.com_1.device: attach terminal 0
425812213299249: board.pc.com_1.device: detach terminal 0
461085751071436: board.pc.com_1.device: attach terminal 0
█
```

Analysis of performance metrics

The page fault rate and TLB miss rate are the key performance metrics used to check the impact of virtual memory. Page fault is when a process tries to access the data from the RAM but is not present, so it needs to access it from the secondary memory. Page fault rate determines how often the page fault occurs. A higher page fault rate can lead to degraded system performance. Likewise, TLB (Transition et al.) is the cache that holds the translation of virtual memory addresses to physical addresses. It helps the lookup process for address translation by providing the recent address mapping. TLB miss rate is a performance metric that determines how often the translation is not found in the buffer. A higher TLB miss rate leads to more page lookups and

slower memory access. A larger TLB size can reduce the miss rate but would add complexity to the architecture and increase power consumption.

In contrast, a smaller buffer size may increase the miss rate, slowing down the system's performance. Likewise, page size is responsible for managing how the memory is divided. A larger page size helps minimize the number of pages needed for the memory, which eventually results in fewer TLB misses. However, if the program to be executed is small, the unused memory on the significant page gets wasted. This can be prevented by using smaller pages. However, it will result in a potential increase in the TLB miss rate. All these metrics can be found in the stat.txt file. However, the system took time to boot in my case, so I did not add a shot of the metrics after modifying the buffer sizes.

All in all, virtual memory is a crucial part of modern operating systems as it offers an abstraction layer between RAM and the application on the system. It allows the OS to give contiguous address space to each process. Also, paging helps in the efficient utilization of physical memory. If a process does not require all the memory, it allows the part of the process to be stored on the disk and loaded into the RAM.

GitHub Repository: https://github.com/NischalJoshi777/computer_arch_assignment3

: