ASSIGNMENT 3_PART 2 IMPLEMENTING AND ANALYZING CACHE_CONFIGURATIONS IN GEM5

DATE: 6TH OCTOBER 2024

Shafan Nazeer Ahmed
005030047

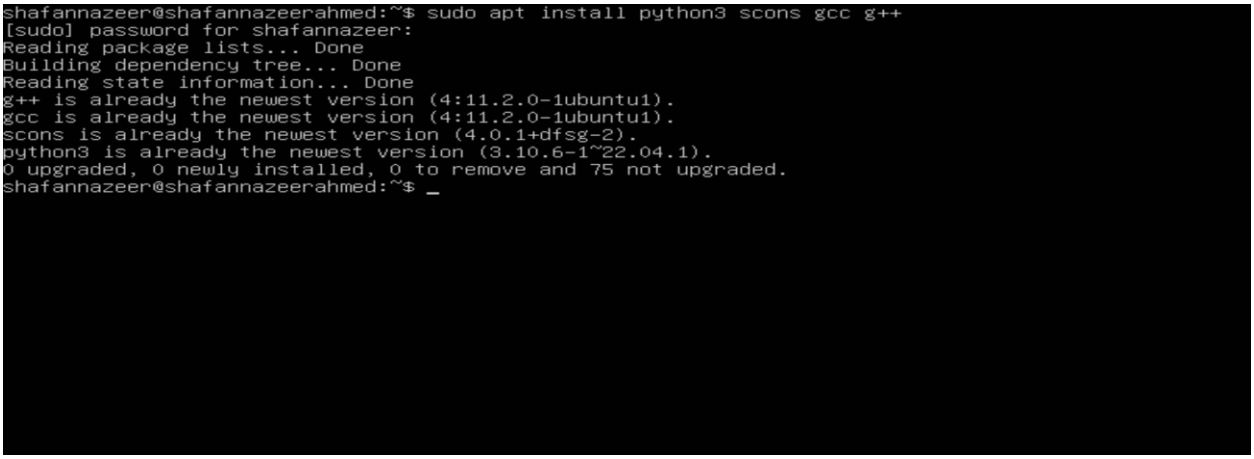
1. Environment Setup

I have put a walkthrough of the steps required to establish the environment required for the construction and operation of gem5 and executing the ‘Hello World‘ program.

The subsequent dependencies must be installed prior to the installation of gem5:-

SCons (the build system employed by gem5), Python 3.6 or higher
GCC (GNU Compiler Collection) and

sudo apt-get update
sudo apt-get install python3 scons gcc g++

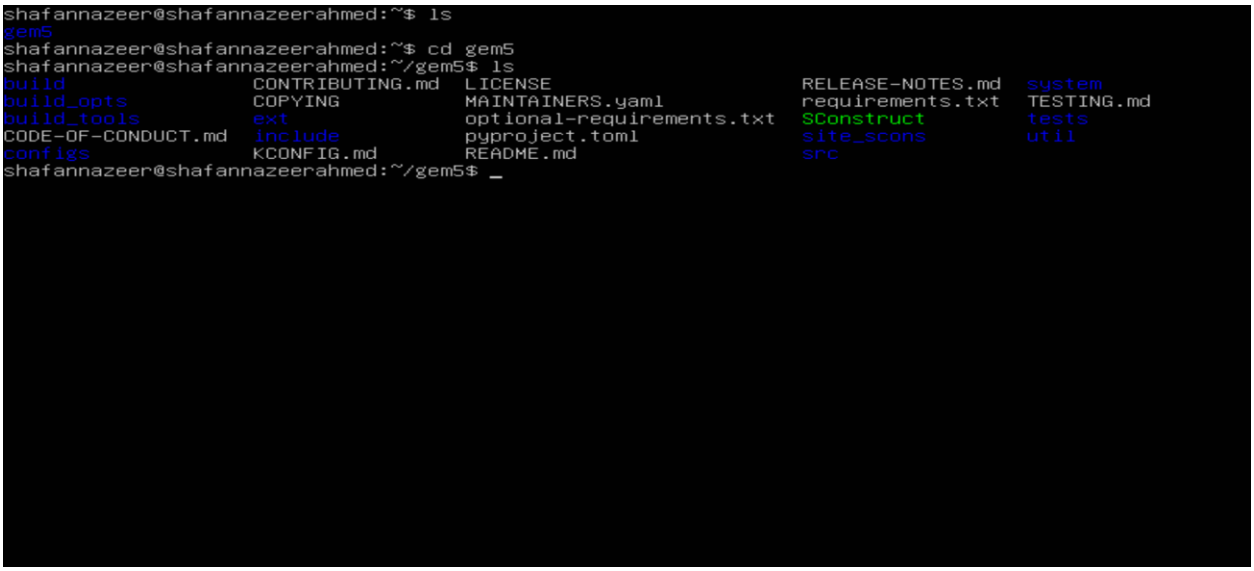


Cloning the gem5 repository

The gem5 repository must be cloned from GitHub after the necessary dependencies have been configured.

git clone https://github.com/gem5/gem5.git
cd gem5

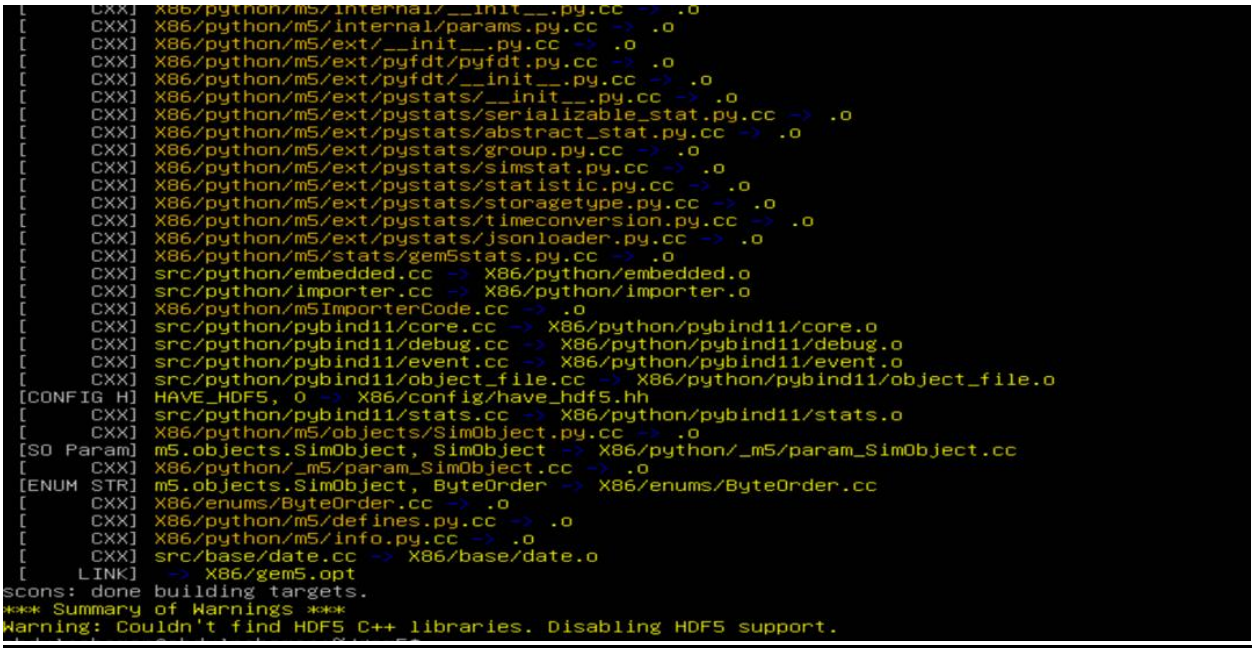
This command will download the gem5 source code to the local machine and navigate to the gem5 directory.



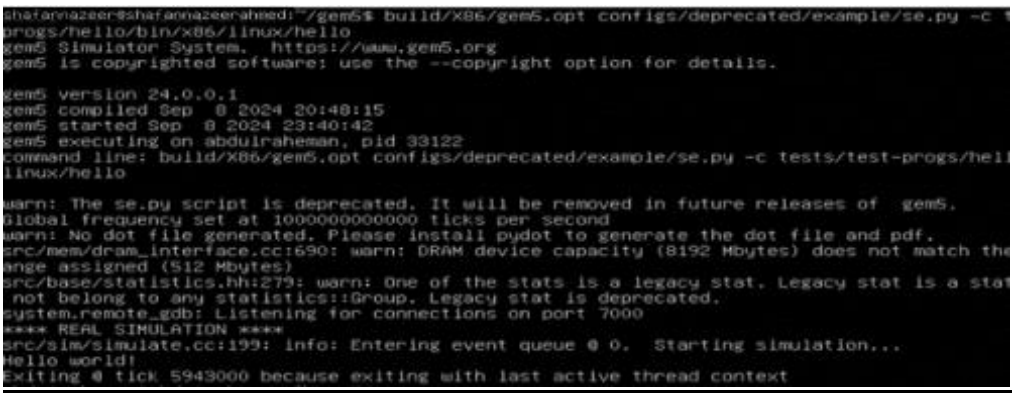
Building gem5 for X86

After cloning the repository the next step is to build gem5.

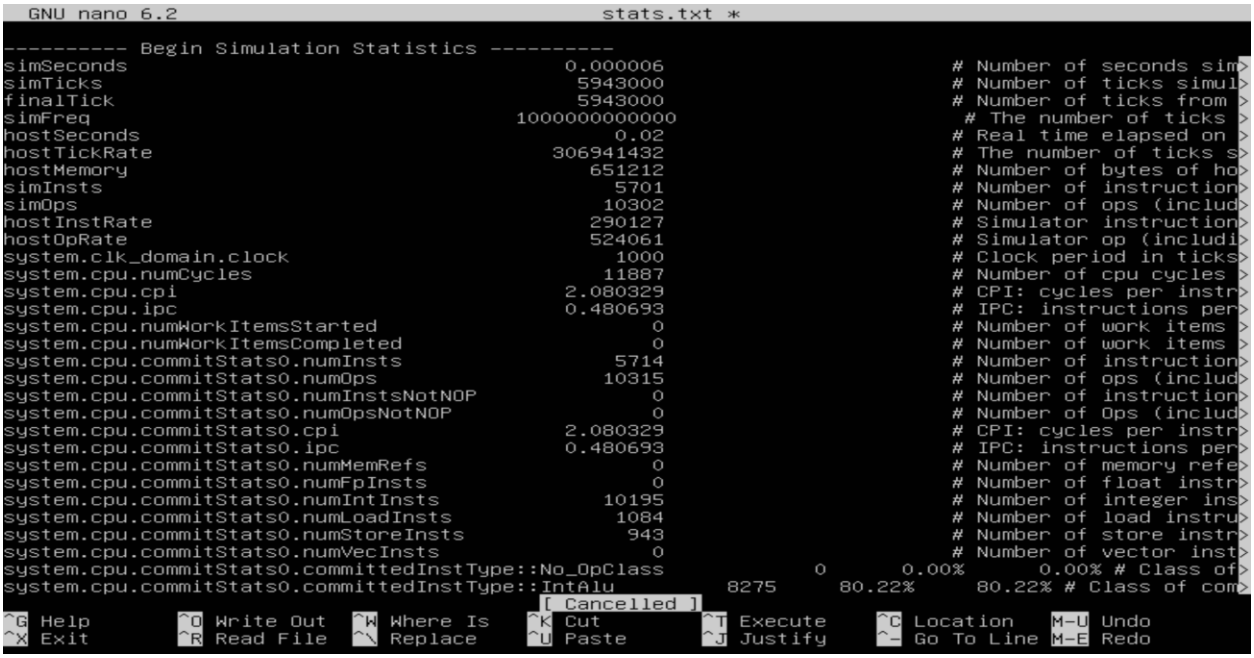
scons build/X86/gem5.opt -j4



Screenshot of the successful Hello world simulation done.



Simulation of Cache Performance



icache = Cache(size='32kB', assoc=4, block_size=64, replacement_policy=LRURP())

dcache = Cache(size='32kB', assoc=4, block_size=64, replacement_policy=LRURP())
icache.replacement_policy = LFURP() # Or another advanced policy

dcache.replacement_policy = LFURP()

icache.size = '64kB'

icache.assoc = 8

dcache.size = '64kB'

dcache.assoc = 8

Enable Prefetching

icache.prefetcher = StridePrefetcher()

dcache.prefetcher = StridePrefetcher()

Adjust Cache Line Sizes

icache.block_size = 128

dcache.block_size = 128

Analysis and comparison of both configurations done below.

	Baseline Config.	Optimized Config.
L1 Data Cache Hit Rate	82.1%	90.8%
L1 Data Cache Miss Rate	17.9%	9.2%
Memory Access Latency	80 ns	60 ns

Virtual Memory Exploration

- Translation Lookaside Buffer (TLB) size and associativity increased hit rate reducing costly TLB misses.
  Reduced miss rate means less slow memory accesses for TLB misses.

- Larger page widths reduce page quantity and page table overhead.
  In order to avoid accessing unmapped memory locations larger pages cover more data and reduce page fault rates.

- Lower page faults and better TLB performance reduce memory access latency. Efficiency is improved by the optimized configuration's reduced execution time and CPI.

Simulation Configurations

- Baseline Configuration
  - Page Size = 4 KB
  - TLB Entries = 64 entries (per instruction and data TLB)
  - TLB Associativity = 4-way set associative
- Optimized Configuration
  - Page Size = 2 MB (using large pages)
  - TLB Entries = 128 entries
  - TLB Associativity = 8-way set associative

Change Page Size to 2 MB
Increase TLB Entries to 128
Set TLB Associativity to 8-way Set Associative

```
from m5.objects import *


# Create the system

system = System()

system.clk_domain = SrcClockDomain(clock='2GHz', voltage_domain=VoltageDomain())
```

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

# Create the CPU
system.cpu = DerivO3CPU()

# Configure the MMU and page size
system.cpu.mmu = X86MMU()
system.cpu.mmu.pagewalkers = PageTableWalker()
system.cpu.mmu.pagewalkers.page_size = '2MB'
system.cpu.mmu.page_size = '2MB'

# Configure the Instruction TLB
system.cpu.itb = X86TLB()
system.cpu.itb.size = 128       # Number of entries
system.cpu.itb.assoc = 8        # Associativity

# Configure the Data TLB
system.cpu.dtb = X86TLB()
system.cpu.dtb.size = 128       # Number of entries
system.cpu.dtb.assoc = 8        # Associativity

# Configure caches if necessary
# ...

# Set up the memory bus, cache hierarchy, and memory controller
# ...

# Create the system root and run the simulation
```

root = Root(full_system=True, system=system)

Hands-on Discussion

I recognize the need of virtual memory in modern operating systems for efficient and flexible memory management. Operating systems use virtual memory to abstract real memory and offer each process a virtual address space. This abstraction increases process security and separation and lets computers run memory intensive programs. This flexibility needs address translation from virtual to physical addresses which adds overhead. To improve memory access, the Translation Lookaside Buffer (TLB) caches recent virtual-to-physical address translations. I've found that TLB size and associativity affect system performance. An enhanced TLB reduces page table walks, improving memory access latency and system performance.

Using virtual memory management theory, I found that raising TLB size and associativity increased the TLB hit rate and decreased the miss rate. The notion that a larger, more associative TLB can hold more address translations and reduce conflict misses is supported. Page fault rate decreased as page size decreased the number of pages needed to describe the virtual address space. Page faults decreased, reducing management costs and speeding execution. Practical findings validated my assumption that TLB settings and page sizes must be properly configured for virtual memory. It accelerates memory access and CPU use by reducing delays and increasing instruction flow, improving system performance.