Caching and Virtual Memory with GEM5

In order to understand the difference between gem5 cache levels and their impact on performance, I started by creating a simple gem5 simulation file. GEM5, by default, allowed me to simulate a sample `hello world` program without caching. I ran the simulation and got the expected output on the terminal. In order to measure the performance of the cache levels, I will be using the stats.txt file that gem5 produces at the end of the simulation.

After a simulation run, gem5 produced a stats.txt file containing numerous counters related to cache performance. For instance, for L1 caches, we could examine counters such as *system.cpu.icache.accesses*, *system.cpu.icache.misses, system.cpu.dcache.accesses,* and *system.cpu.dcache.misses*. For the simulation of the L2 cache, there are counters like *system.l2.overall_accesses* and *system.l2.overall_misses*. When combined with latency statistics, these metrics allowed me to calculate hit/miss rates and provide insights into other performance aspects, such as miss penalties.

The next step I performed was adding default caching to the system and running a simulation. After completing the simulation, I recorded the results from the stats.txt file. The default cache options included the following properties:

```
class L1_ICache(Cache):
    size = '32kB'
    assoc = 2            # Default associativity is typically lower
    tag_latency = 2      # Lower latencies for default L1 caches
    data_latency = 2
    response_latency = 2
    mshrs = 4            # Fewer MSHRs by default
    tgts_per_mshr = 8    # Remains the same as many default configs
```

Essential performance metrics were recorded for both the L1 Data and Instruction caches. For the L1 Data Cache, 3,996 demand accesses were observed, of which 3,718 resulted in hits and 278 in misses, leading to a miss rate of approximately 6.96% and a

corresponding hit rate of about 93.04%. The average miss latency for the L1 Data Cache was roughly 62,766 ticks. In comparison, the L1 Instruction Cache exhibited 13,954 demand accesses with 13,430 hits and 524 misses, yielding a miss rate of around 3.76% (or a hit rate near 96.24%) and an average miss latency of approximately 59,038 ticks. After that, I increased the value provided for associativity and other caching-related properties and reran the simulation. Here are the values that I provided for the second run:

```
class L1_ICache(Cache):
    size = '64kB'
    assoc = 4           # Default associativity is typically lower
    tag_latency = 10      # Lower latencies for default L1 caches
    data_latency =10
    response_latency = 10
    mshrs = 10            # Fewer MSHRs by default
    tgts_per_mshr = 8    # Remains the same as many default configs
```

In this simulation, the L1 instruction cache registered about 13,954 accesses with 518 misses—yielding a miss rate of roughly 3.7% (and therefore a hit rate of about 96.3%)—with an average miss latency of approximately 75,585 ticks. Meanwhile, the L1 data cache saw 3,996 accesses with 277 misses, translating to a miss rate of around 6.9% (or a 93.1% hit rate) and an average miss latency of roughly 78,993 ticks. The DRAM subsystem processed 796 read bursts at the memory level with a total memory access latency of about 21,032,250 ticks, averaging around 26,422 ticks per burst. I did not observe much difference between the first two runs based on the percentage of hits and misses. I used the following cache settings for the final run and executed the simulation again.

```
class L1_ICache(Cache):
    size = '128kB'
    assoc = 8
```

```
    tag_latency = 20
    data_latency = 20
    response_latency = 20
    mshrs = 20
    tgts_per_mshr = 8
```

The statistics showed a similar result again. The simulation results showed that the L1 data cache processed 3,996 demand accesses, achieving 3,719 hits and 277 misses—translating to a hit rate of about 93% and a miss rate of roughly 7%, with an average miss latency of approximately 95,845 ticks. Meanwhile, the L1 instruction cache handled 13,954 accesses, with 13,436 hits and 518 misses, resulting in a hit rate of around 96.3%, a miss rate near 3.7%, and an average miss latency of roughly 97,919 ticks. This observation was similar to the first two runs. Here is the summary of the simulations based on varying parameters for the L1 cache

Run	Cache Size	Associativity	Tag Latency	Response Latency
Default	32KB	2	2 ticks	2 ticks
1st run	64KB	4	10 ticks	10 ticks
2nd run	128KB	8	20 ticks	20 ticks

However, I did not see considerable changes in the hit/miss rate for both the Instruction and Data cache. The following table shows the hit/miss rate.

Cache Type	Run	Demand access	Misses	Hit rate	Miss rate	Avg. Miss Latency (ticks)
L1D Cache	Baseline	3996	278	~93.04%	~6.96%	~62,766
L1D Cache	Second run	3996	277	~93.1%	~6.9%	~78993
L1D Cache	Final run	3996	277	~93%	~7%	~95845

L1I Cache	Baseline	13954	524	~96.24%	~3.76%	~59038
L1I Cache	Second run	13954	518	~96.3%	~3.7%	~75585
L1I Cache	Final Run	13954	518	~96.3%	~3.7%	~97919

Increasing the L1 cache size and associativity did not significantly improve the hit-or-miss rates. The data cache maintained around a 93% hit rate in all three runs, and the instruction cache had about a 96% hit rate. However, as the cache configuration was scaled up (from 32 kB/2-way to 64 kB/4-way and finally 128 kB/8-way), the average miss latency increased noticeably (from ~62,766 to ~78,993, then ~95,845 ticks for the data cache, and similarly for the instruction cache). This indicates that while the larger, more associative caches did not enhance the hit rates (likely because the working set already fit in the baseline cache), the increased access latencies of the larger configurations adversely affected overall performance.
In order to run the virtual memory, I needed to choose a compatible CPU. I switched my CPU type to X8603 CPU as it supports virtual memory. Along with that, I had to add the properties below. I tested the simulation for the sizes 32 and 64.

```
system.cpu.itb = X86TLB(size=64, entry_type='instruction')  # Instruction
TLB
system.cpu.dtb = X86TLB(size=64, entry_type='data')  # Data TLB
```

Once the virtual memory was added, I ran the simulation and recorded the performance twice.I ran the simulation twice with different size values of 32 and 64; however, I got the same result for both runs. The stats summary showed that the miss rates for the data and instruction TLBs remained consistent regardless of the page size. For the data TLB, there were 2,986 read accesses, resulting in 88 misses, which equates to an approximate miss rate of 2.95%. In addition, the data TLB handled 2,466 write accesses with only 19 misses, reflecting a lower miss rate of about 0.77%. On the other hand, the instruction TLB did not record any read

accesses, yielding a 0% miss rate for those, but it did encounter 207 misses out of 2,945 write accesses, resulting in a miss rate of roughly 7.03%. Overall, adding the virtual memory should allow the larger and more complex programs to run efficiently even when the RAM is limited. However, I could not see much improvement in small programs like Hello World.