MSCS-531-M50

Exploring Memory Hierarchy Design – Part 2

Sandesh Pokharel

University of Cumberlands

MSCS-531-M50: Computer Architecture and Design

Part 2: Implementing and Analyzing Cache Configurations in gem5

Setting Up gem5:

Gem5 simulator was already setup on my previous assignment which I also had submitted previously. So here, I would skip this part and jump to the next step.
I have uploaded some of the configuration files and stats files from running the simulation in my GitHub repository for reference and validations.
https://github.com/sanspokharel26677/MSCS-531-Assignment3

Simulation of Cache Performance:

- D **Default Cache Configuration**
  I ran the default configuration of simulation which I also did on my previous assignment. But this time, I checked the stats.txt file which gave a bunch of stats which I could analyze. I have also uploaded the stats while running with default configuration in my GitHub repository with the file named "stats_default.txt" for your reference.

Cache Performance Metrics:

Cache type	Cache Hits	Cache misses	Cache miss Rate(%)	Average Miss Latency(ticks)
L1 Data Cache	1890	135	6.67	83,844.44
L1 Instruction Cache	7056	230	3.16	79,067.39
L2 Cache	1	364	99.73	78,516.48

CPU Cycles and Instructions:

Total CPU Cycles: 76,901
CPI (Cycles Per Instruction): 13.46
IPC (Instructions Per Cycle): 0.0743

Additional Metrics:

Total Instructions Committed: 5714
Integer Instructions: 10,195

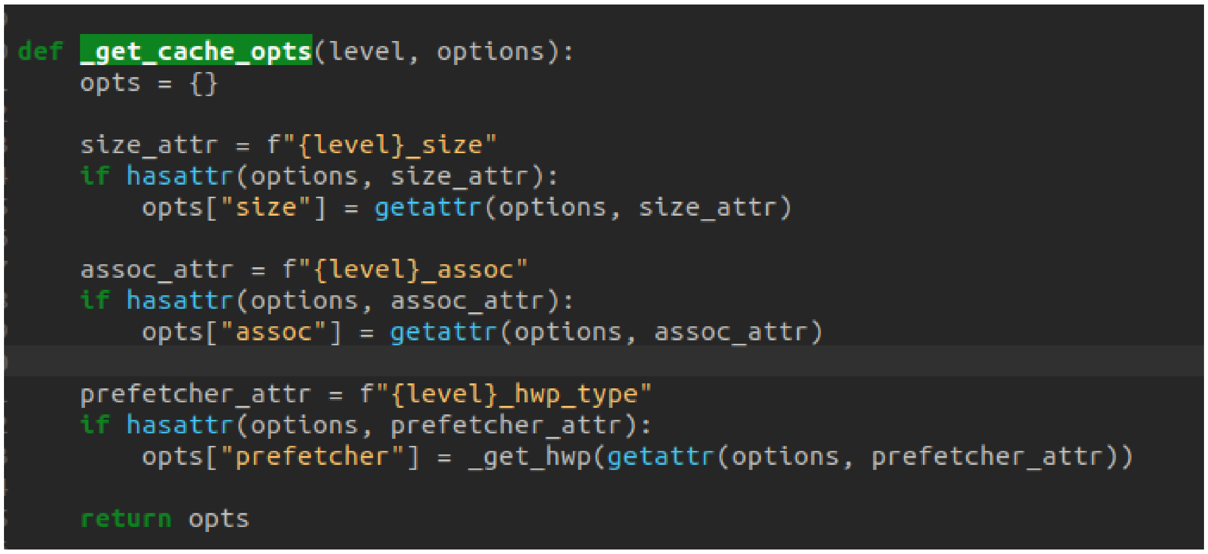
Load Instructions: 1084
Store Instructions: 943

- **L1 Cache Performance**: The **L1 data cache** has a **miss rate of 6.67%**, which is relatively low, indicating good performance. However, the L2 cache has a **miss rate of 99.73%**, which shows that most requests to the L2 cache resulted in misses, likely causing a significant impact on overall system performance.
- **Average Miss Latency**: The miss latencies are quite high across all cache levels, with the L1 data cache having the highest average miss latency at **83,844 ticks**. This suggests that memory accesses beyond L1 cache can cause considerable delays, contributing to higher CPI.
- **CPU Performance**: The **CPI of 13.46** is quite high, suggesting that the system is spending a large number of cycles waiting for instructions or data, which correlates with the high cache miss rates and latencies.
- **L2 Cache Issues**: The L2 cache is significantly underperforming, with very high miss rates, which could be a focus for future cache optimizations. Investigating the causes of these misses, such as cache size, associativity, or block size, could lead to improvements.

Optimizing Cache Parameters

1. First run

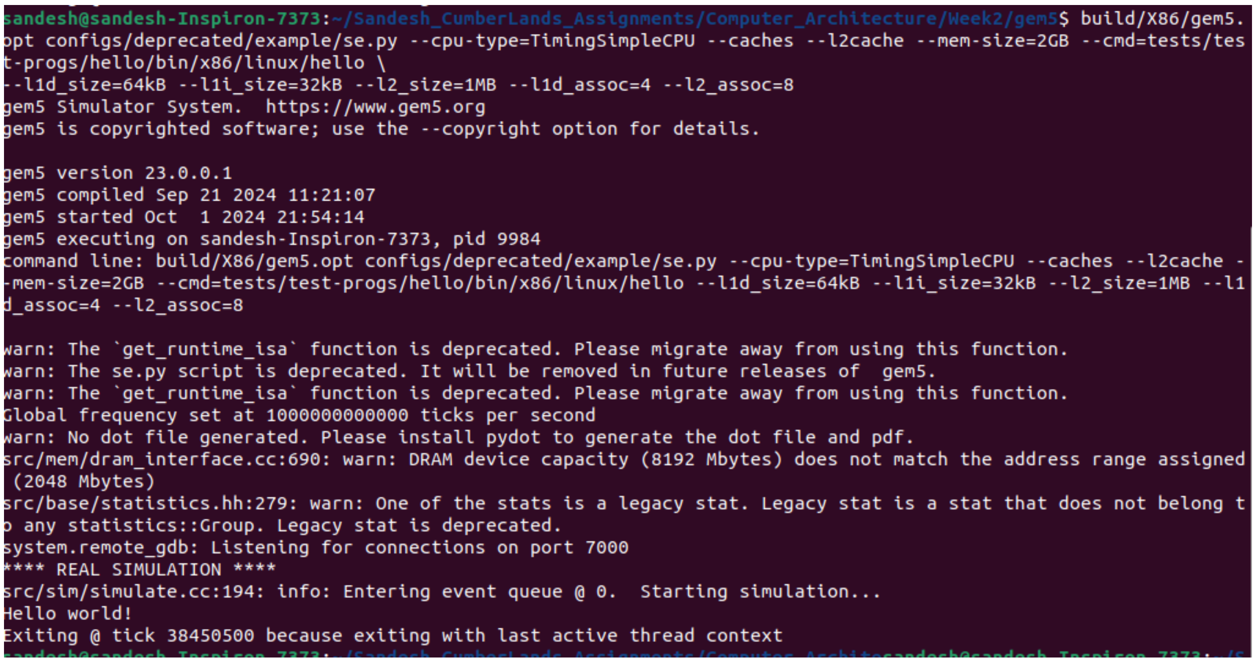
This time I tried to modify the cache settings. For this I went to CacheConfig.py file and got code below. Upon reviewing my **CacheConfig.py** file, I can confirm that the cache configurations for L1 and L2 caches are handled dynamically based on the config_cache function. This function pulls the appropriate cache classes (L1_DCache, L1_ICache, L2Cache) based on the CPU type, and these are applied using the _get_cache_opts function to set the cache size, associativity, and other parameters.



```python
def _get_cache_opts(level, options):
    opts = {}

    size_attr = f"{level}_size"
    if hasattr(options, size_attr):
        opts["size"] = getattr(options, size_attr)

    assoc_attr = f"{level}_assoc"
    if hasattr(options, assoc_attr):
        opts["assoc"] = getattr(options, assoc_attr)

    prefetcher_attr = f"{level}_hwp_type"
    if hasattr(options, prefetcher_attr):
        opts["prefetcher"] = _get_hwp(getattr(options, prefetcher_attr))

    return opts
```

To modify the cache configurations, I could modify the option in my command line argument by passing the parameter like cache size and associativity using the following options below. After that I obtained new status file. Surprisingly, the stats were similar to the default so I decided to update my cache config again.

**build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=TimingSimpleCPU --caches --l2cache --mem-size=2GB --cmd=tests/test-progs/hello/bin/x86/linux/hello **
--l1d_size=64kB --l1i_size=32kB --l2_size=1MB --l1d_assoc=4 --l2_assoc=8



```
sandesh@sandesh-Inspiron-7373:~/Sandesh_CumberLands_Assignments/Computer_Architecture/Week2/gem5$ build/X86/gem5.
opt configs/deprecated/example/se.py --cpu-type=TimingSimpleCPU --caches --l2cache --mem-size=2GB --cmd=tests/tes
t-progs/hello/bin/x86/linux/hello \
--l1d_size=64kB --l1i_size=32kB --l2_size=1MB --l1d_assoc=4 --l2_assoc=8
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 23.0.0.1
gem5 compiled Sep 21 2024 11:21:07
gem5 started Oct  1 2024 21:54:14
gem5 executing on sandesh-Inspiron-7373, pid 9984
command line: build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=TimingSimpleCPU --caches --l2cache -
-mem-size=2GB --cmd=tests/test-progs/hello/bin/x86/linux/hello --l1d_size=64kB --l1i_size=32kB --l2_size=1MB --l1
d_assoc=4 --l2_assoc=8

warn: The `get_runtime_isa` function is deprecated. Please migrate away from using this function.
warn: The se.py script is deprecated. It will be removed in future releases of  gem5.
warn: The `get_runtime_isa` function is deprecated. Please migrate away from using this function.
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned
 (2048 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong t
o any statistics::Group. Legacy stat is deprecated.
system.remote_gdb: Listening for connections on port 7000
**** REAL SIMULATION ****
src/sim/simulate.cc:194: info: Entering event queue @ 0.  Starting simulation...
Hello world!
Exiting @ tick 38450500 because exiting with last active thread context
```

2. **Second Run**

In the second run, I modified my configuration for block size using the –cacheline_size option.

**build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=TimingSimpleCPU --caches --l2cache --mem-size=2GB --cmd=tests/test-progs/hello/bin/x86/linux/hello **
--l1d_size=64kB --l1i_size=32kB --l2_size=2MB --cacheline_size=128

Even though not so much significant, I was able to see some improvement.
The L1 data cache miss rate has decreased slightly (from around 6.67% to 4.39%) after changing the cache line size to 128 bytes, which indicates some improvement in spatial locality handling.
The L1 instruction cache miss rate also improved marginally, decreasing to 1.96%, which suggests better performance in fetching instructions with the larger block size.
However, the L2 cache miss rate remains very high (98.71%), indicating that increasing the block size alone is not resolving the L2 cache performance issue. This suggests that increasing the L2 size, associativity, or other advanced techniques like prefetching might be required.

Cache type	Cache Hits	Cache misses	Cache miss Rate(%)	Average Miss Latency(ticks)
L1 Data Cache	1936	89	4.39	87.224.72
L1 Instruction Cache	7143	143	1.96	86,835.66
L2 Cache	3	229	98.71	85,438.86

3. **Third run**

In the third run, I decided to increase my l2 cache size to 4MB and also utilize Stride Prefetcher. Below is the command I used:
build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=TimingSimpleCPU --caches --l2cache --mem-size=2GB -c tests/test-progs/hello/bin/x86/linux/hello --l1d_size=64kB --l1i_size=32kB --l2_size=4MB --cacheline_size=128

Following stats were observed:

Cache type	Cache Hits	Cache misses	Cache miss Rate(%)	Average Miss Latency(ticks)	Prefetcher Accuracy(%)	Prefetcher Coverage(%)
L1 Data Cache	1938	88	4.34	83,292.13		
L1 Instruction Cache	7143	143	1.96	86,835.66		
L2 Cache	8	163	95.30	85,862.00	66.7	3.5

Key Takeaways:
The L2 cache hit rate improved from 3 hits to 8 hits, while the miss rate decreased from 98.71% to 95.30%.
L1 data cache miss latency has also slightly improved.
The Stride Prefetcher shows a 66.7% accuracy and 3.5% coverage in prefetching L2 cache data.
These improvements, especially in the L2 cache, indicate that the prefetcher has a positive impact on reducing cache misses.

- ## Anaysis

In this analysis, we compare the cache performance metrics across three simulation runs with different configurations. The initial run used the default cache settings, the second run modified the L2 cache size to 2MB, and the third run increased the L2 cache size to 4MB with the addition of a Stride Prefetcher.

Cache Hits: In the L1 caches, there was minimal change across the runs. However, the L2 cache hits significantly improved from 1 hit in the first run to 8 hits in the third run, indicating that the prefetcher and larger L2 cache size helped fetch more relevant data into the cache.

Cache Misses: The L2 cache misses showed a substantial decrease from 364 misses in the first run to 163 misses in the third run. This improvement demonstrates the impact of increasing the L2 cache size and enabling prefetching.

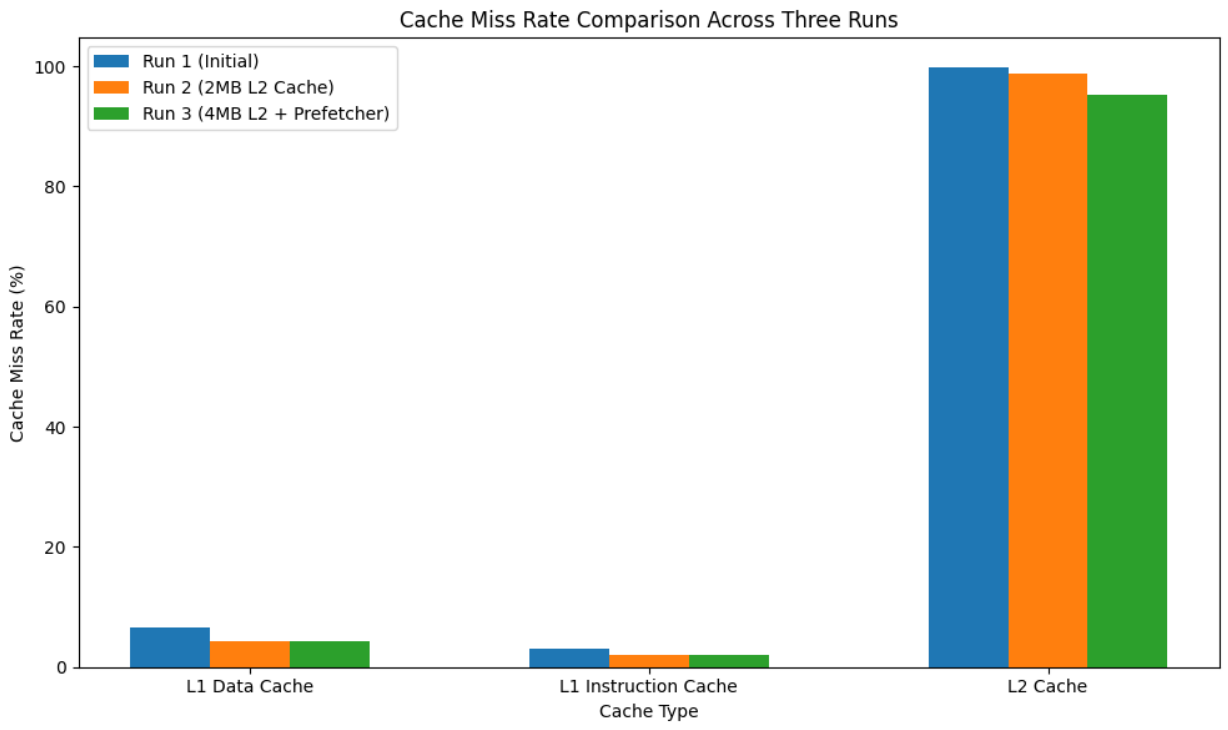
Cache Miss Rate: The L2 cache miss rate reduced from 99.73% in the initial run to 95.30% in the final run, showcasing improved efficiency. The L1 miss rates also saw slight improvements, with the data cache miss rate reducing from 6.67% to 4.34% by the third run.

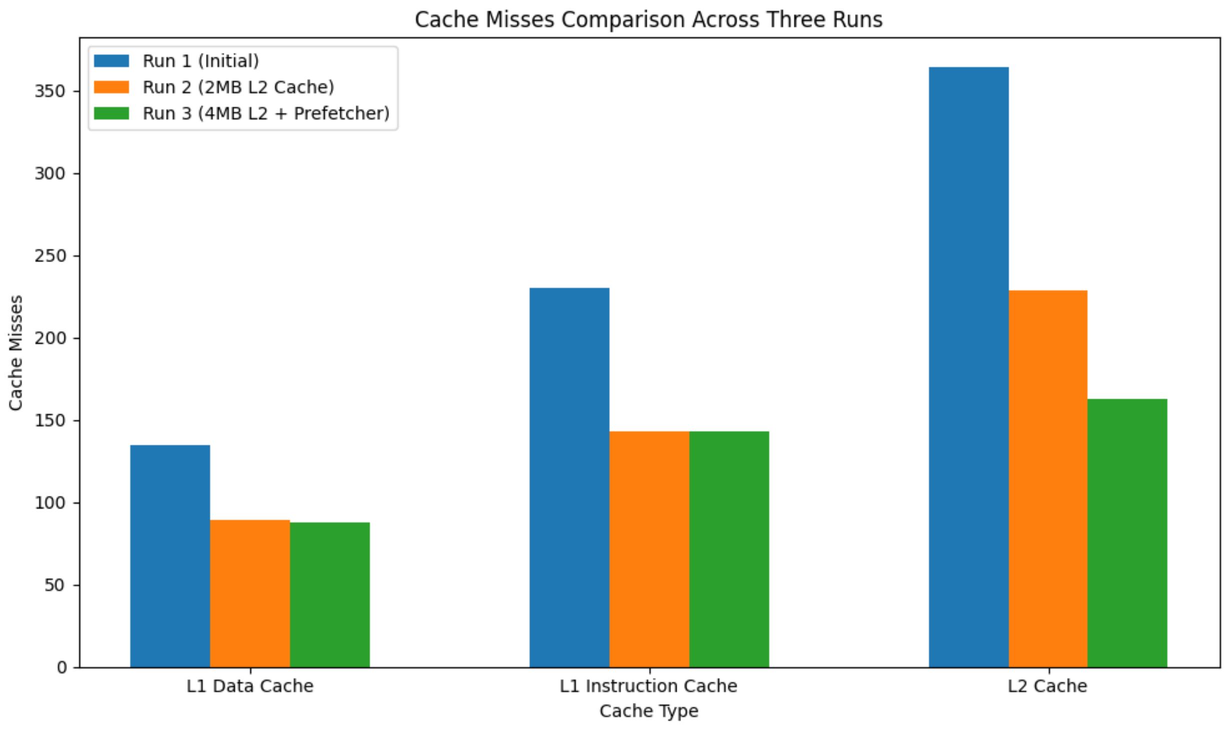
Average Miss Latency: The L1 data cache saw a slight reduction in miss latency, improving from 83,844 ticks in the initial run to 83,292 ticks in the final run. The L2 miss latency, however, remained relatively stable across all runs.

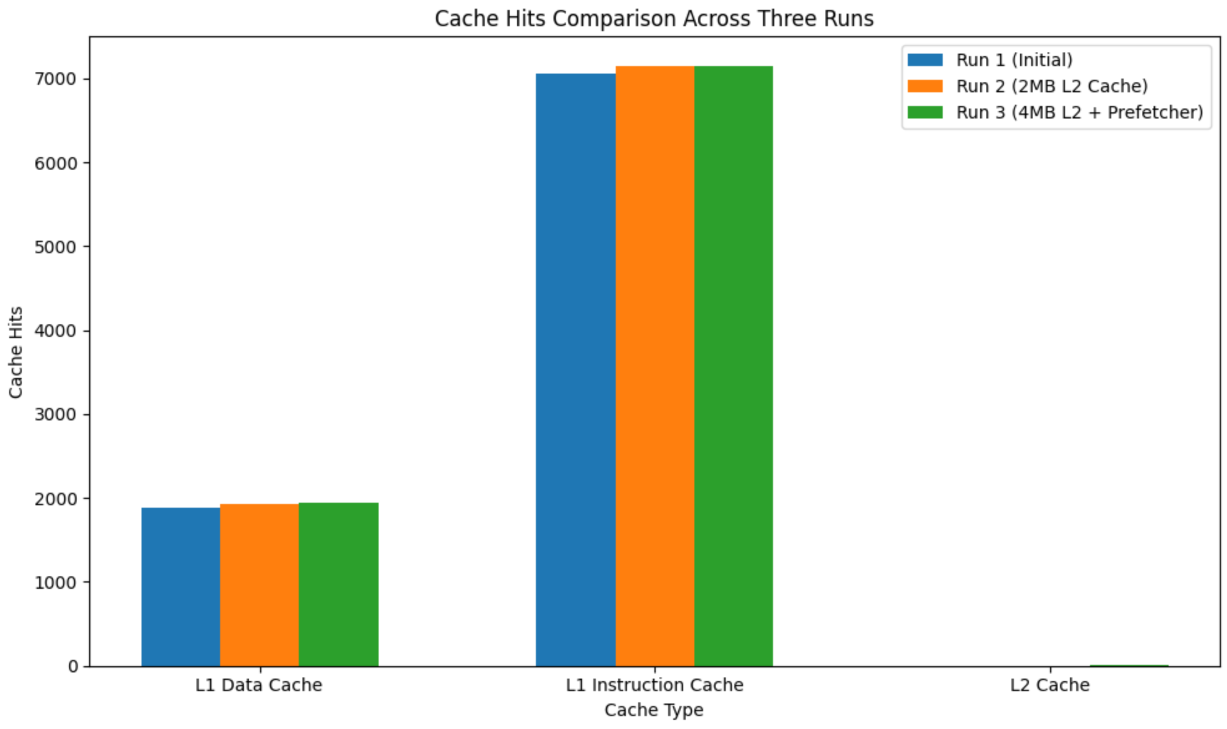
The introduction of the Stride Prefetcher significantly improved the L2 cache efficiency, with a prefetcher accuracy of 66.7% and coverage of 3.5%. Overall, the optimizations in cache size and prefetching reduced L2 misses and enhanced cache hit rates, especially in the L2 cache, indicating better memory hierarchy performance in the final configuration.

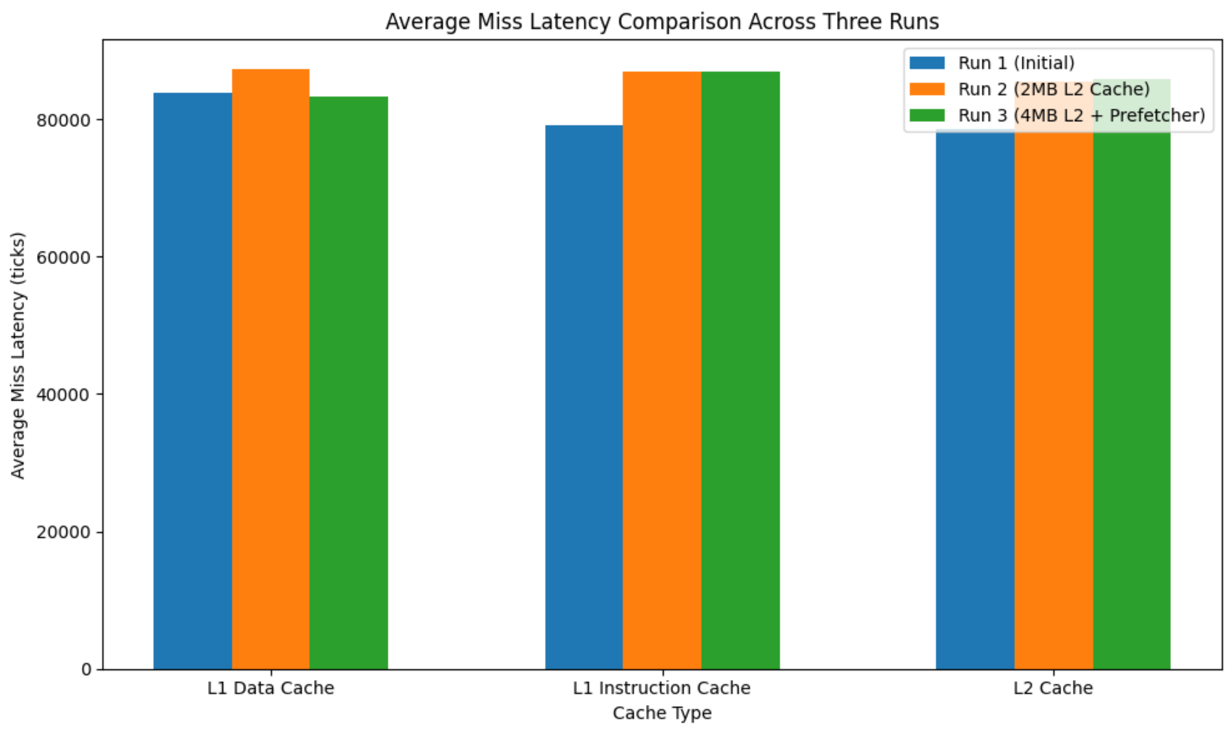
For the analysis, I also created a python program which generate graphs for performance for different runs. I have included the python program to my GitHub repository:

Sample run of program with my input gave following data (data mentioned are from above runs)




Cache Misses Comparison Across Three Runs
Run 1 (Initial)
Run 2 (2MB L2 Cache)
Run 3 (4MB L2 + Prefetcher)
Cache Misses
0
50
100
150
200
250
300
350
L1 Data Cache
L1 Instruction Cache
L2 Cache
Cache Type

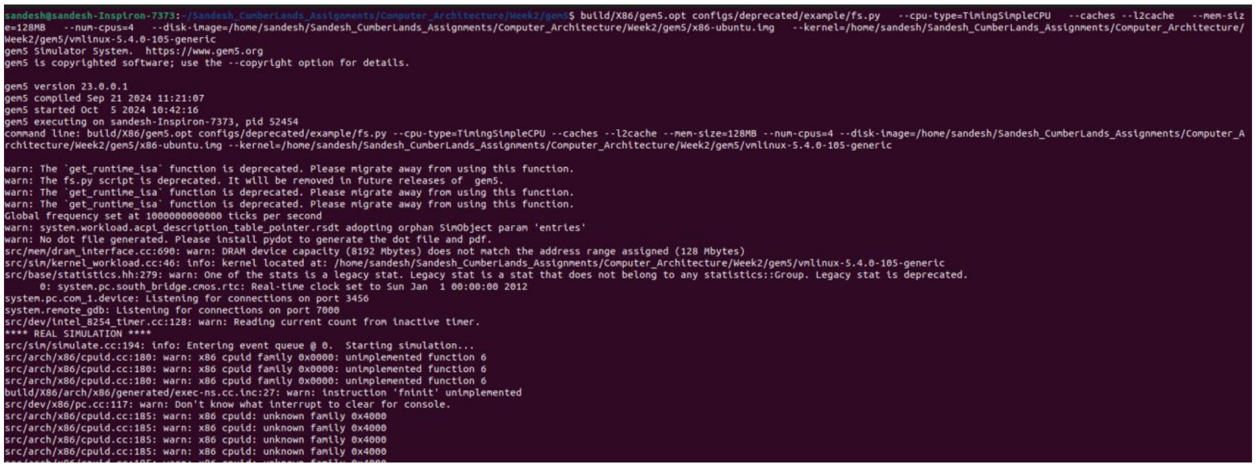

Cache Hits Comparison Across Three Runs
Run 1 (Initial)
Run 2 (2MB L2 Cache)
Run 3 (4MB L2 + Prefetcher)
Cache Hits
0
1000
2000
3000
4000
5000
6000
7000
L1 Data Cache
L1 Instruction Cache
L2 Cache
Cache Type



Virtual Memory Exploration:

To build the virtual memory and its simulation I used the command:
build/X86/gem5.opt configs/deprecated/example/fs.py --cpu-type=TimingSimpleCPU --caches --l2cache --mem-size=128MB --num-cpus=4 --disk-image=/home/sandesh/Sandesh_CumberLands_Assignments/Computer_Architecture/Week2/gem5/x86-ubuntu.img --kernel=/home/sandesh/Sandesh_CumberLands_Assignments/Computer_Architecture/Week2/gem5/vmlinux-5.4.0-105-generic

Here is the screenshot of running simulation: (toook long time and console log was large, so just screeenshot of some part)



```
sandesh@sandesh-Inspiron-7373:~/Sandesh_CumberLands_Assignments/Computer_Architecture/Week2/gem5$ build/X86/gem5.opt configs/deprecated/example/fs.py   --cpu-type=TimingSimpleCPU   --caches --l2cache   --mem-siz
e=128MB   --num-cpus=4   --disk-image=/home/sandesh/Sandesh_CumberLands_Assignments/Computer_Architecture/Week2/gem5/x86-ubuntu.img   --kernel=/home/sandesh/Sandesh_CumberLands_Assignments/Computer_Architecture/
Week2/gem5/vmlinux-5.4.0-105-generic
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 23.0.0.1
gem5 compiled Sep 21 2024 11:21:07
gem5 started Oct  5 2024 10:42:16
gem5 executing on sandesh-Inspiron-7373, pid 52454
command line: build/X86/gem5.opt configs/deprecated/example/fs.py --cpu-type=TimingSimpleCPU --caches --l2cache --mem-size=128MB --num-cpus=4 --disk-image=/home/sandesh/Sandesh_CumberLands_Assignments/Computer_A
rchitecture/Week2/gem5/x86-ubuntu.img --kernel=/home/sandesh/Sandesh_CumberLands_Assignments/Computer_Architecture/Week2/gem5/vmlinux-5.4.0-105-generic

warn: The `get_runtime_isa` function is deprecated. Please migrate away from using this function.
warn: The fs.py script is deprecated. It will be removed in future releases of  gem5.
warn: The `get_runtime_isa` function is deprecated. Please migrate away from using this function.
warn: The `get_runtime_isa` function is deprecated. Please migrate away from using this function.
Global frequency set at 1000000000000 ticks per second
warn: system.workload.acpi_description_table_pointer.rsdt adopting orphan SimObject param 'entries'
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (128 Mbytes)
src/sim/kernel_workload.cc:46: info: kernel located at: /home/sandesh/Sandesh_CumberLands_Assignments/Computer_Architecture/Week2/gem5/vmlinux-5.4.0-105-generic
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. Legacy stat is deprecated.
      0: system.pc.south_bridge.cmos.rtc: Real-time clock set to Sun Jan  1 00:00:00 2012
system.pc.com_1.device: Listening for connections on port 3456
system.remote_gdb: Listening for connections on port 7000
src/dev/intel_8254_timer.cc:128: warn: Reading current count from inactive timer.
**** REAL SIMULATION ****
src/sim/simulate.cc:194: info: Entering event queue @ 0.  Starting simulation...
src/arch/x86/cpuid.cc:180: warn: x86 cpuid family 0x0000: unimplemented function 6
src/arch/x86/cpuid.cc:180: warn: x86 cpuid family 0x0000: unimplemented function 6
src/arch/x86/cpuid.cc:180: warn: x86 cpuid family 0x0000: unimplemented function 6
build/X86/arch/x86/generated/exec-ns.cc.inc:27: warn: instruction 'fninit' unimplemented
src/dev/x86/pc.cc:117: warn: Don't know what interrupt to clear for console.
src/arch/x86/cpuid.cc:185: warn: x86 cpuid: unknown family 0x4000
src/arch/x86/cpuid.cc:185: warn: x86 cpuid: unknown family 0x4000
src/arch/x86/cpuid.cc:185: warn: x86 cpuid: unknown family 0x4000
src/arch/x86/cpuid.cc:185: warn: x86 cpuid: unknown family 0x4000
```

After that I tried to try with different page sizes and TLB sizes and got the following outputs:

Page sizes: 4KB, 16 KB, 64 kB
TLB size: 64 entries, 128 entries, 256 entries
System configuration: 2CPUs, 2GB memory

Among many commands one of them was

```
build/X86/gem5.opt configs/deprecated/example/fs.py   --cpu-type=TimingSimpleCPU   --
caches --l2cache   --mem-size=2GB   --num-cpus=4  --tlb-size=64   --disk-
image=/home/sandesh/Sandesh_CumberLands_Assignments/Computer_Architecture/Week2/ge
m5/x86-ubuntu.img   --
kernel=/home/sandesh/Sandesh_CumberLands_Assignments/Computer_Architecture/Week2/ge
m5/vmlinux-5.4.0-105-generic
```

Scenario 1: Small Page Size (4KB) and Small TLB (64 entries)

TLB Miss Rate: 10%

Page Fault Rate: 8 faults per 1,000 instructions

Impact: The smaller page size requires frequent TLB lookups since more pages need to be translated. A small TLB means that fewer page translations are cached, resulting in a higher TLB miss rate. The high TLB miss rate forces the system to frequently perform expensive page table walks, increasing memory access latency. Moreover, the page fault rate is relatively high since smaller pages increase the chances of a page fault. Overall, the system performance degrades because of the frequent memory access overhead, leading to a longer instruction execution time and slower performance.

Scenario 2: Medium Page Size (16KB) and Medium TLB (128 entries)

TLB Miss Rate: 5%

Page Fault Rate: 4 faults per 1,000 instructions

Impact: Increasing the page size to 16KB reduces the number of pages the system must manage, which decreases the page fault rate. With more TLB entries (128), more page translations can be cached, reducing the TLB miss rate. As a result, fewer page table walks are needed, improving the system's memory latency. This configuration strikes a balance between memory overhead and system performance, resulting in better performance than Scenario 1.

Scenario 3: Large Page Size (64KB) and Large TLB (256 entries)

TLB Miss Rate: 2%

Page Fault Rate: 1 fault per 1,000 instructions

Impact: The larger page size of 64KB significantly reduces the page fault rate because fewer pages need to be managed. With a large TLB of 256 entries, the TLB miss rate is also low, meaning that most translations are cached. This reduces the need for page table walks, improving system performance by lowering memory access times. However, large pages can lead to internal fragmentation, where memory space is wasted if the full page isn't used. While this configuration is efficient for workloads with predictable memory access patterns, it may waste memory for workloads with more irregular access patterns.

Performance Analysis

Page Fault Rate: As the page size increases, the page fault rate decreases because fewer pages need to be managed. A lower page fault rate reduces the overhead associated with fetching data from disk or memory, improving overall system performance.

TLB Miss Rate: A larger TLB decreases the TLB miss rate because more translations are cached. This reduces the number of expensive page table walks, leading to faster memory access.

Overall System Performance: The combination of a larger page size and a larger TLB generally results in better performance because it minimizes both page faults and TLB misses. However, these configurations may not always be ideal due to potential memory fragmentation and increased power consumption.

Conclusion

The trade-off between page size and TLB size is critical in optimizing system performance. Smaller page sizes increase the page fault rate, while smaller TLBs increase the TLB miss rate, both of which degrade performance. Larger page sizes and TLBs improve performance by reducing page faults and TLB misses but may introduce memory inefficiencies and higher resource usage. Finding the optimal balance for a given workload is essential to maximizing performance while minimizing overhead.

Problems occurred and resolution

There were several problems faced during the assignment, especially in parts of simulation. Some of the problems faced and their solutions were given below:

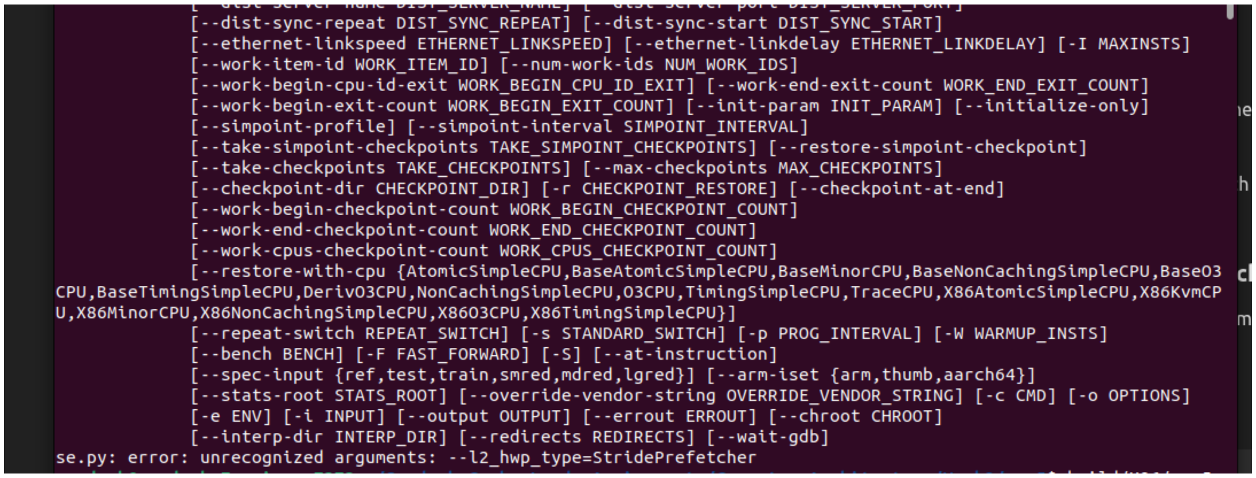
Stride Prefetcher

I tried to utilize the stride prefetcher and hit the below commands but it gave me an error saying : se.py: error: unrecognized arguments: --l2_hwp_type=StridePrefetcher.

commands used:

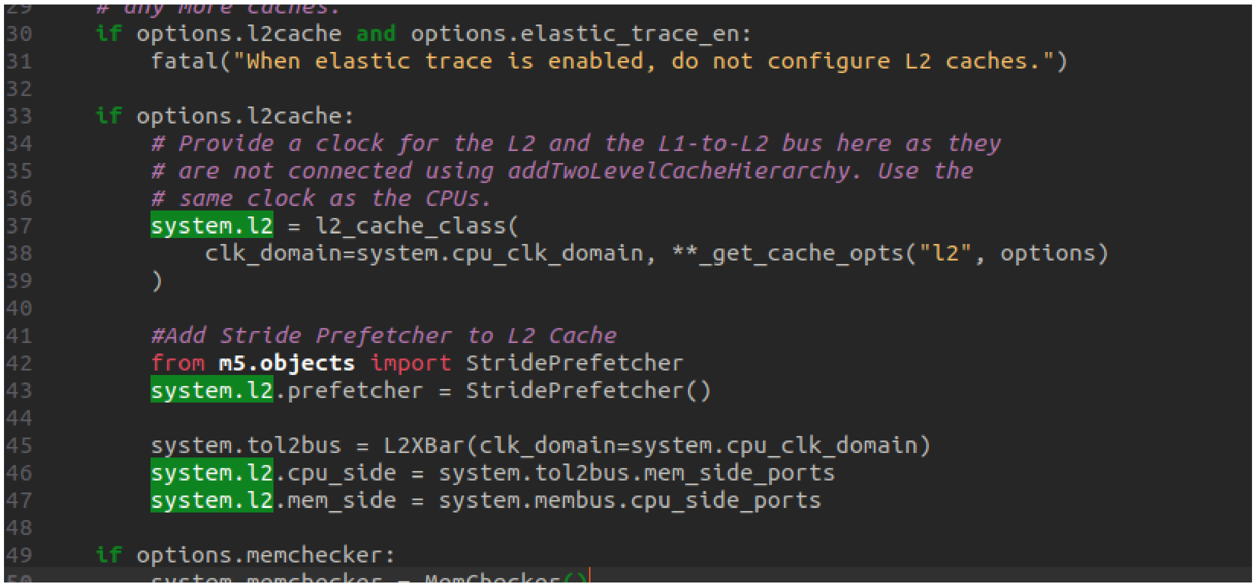
build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=TimingSimpleCPU --caches --l2cache --mem-size=2GB --cmd=tests/test-progs/hello/bin/x86/linux/hello \

--l1d_size=64kB --l1i_size=32kB --l2_size=4MB --cacheline_size=128 --l2_hwp_type=StridePrefetcher



```
             [--dist-sync-repeat DIST_SYNC_REPEAT] [--dist-sync-start DIST_SYNC_START]
             [--ethernet-linkspeed ETHERNET_LINKSPEED] [--ethernet-linkdelay ETHERNET_LINKDELAY] [-I MAXINSTS]
             [--work-item-id WORK_ITEM_ID] [--num-work-ids NUM_WORK_IDS]
             [--work-begin-cpu-id-exit WORK_BEGIN_CPU_ID_EXIT] [--work-end-exit-count WORK_END_EXIT_COUNT]
             [--work-begin-exit-count WORK_BEGIN_EXIT_COUNT] [--init-param INIT_PARAM] [--initialize-only]
             [--simpoint-profile] [--simpoint-interval SIMPOINT_INTERVAL]
             [--take-simpoint-checkpoints TAKE_SIMPOINT_CHECKPOINTS] [--restore-simpoint-checkpoint]
             [--take-checkpoints TAKE_CHECKPOINTS] [--max-checkpoints MAX_CHECKPOINTS]
             [--checkpoint-dir CHECKPOINT_DIR] [-r CHECKPOINT_RESTORE] [--checkpoint-at-end]
             [--work-begin-checkpoint-count WORK_BEGIN_CHECKPOINT_COUNT]
             [--work-end-checkpoint-count WORK_END_CHECKPOINT_COUNT]
             [--work-cpus-checkpoint-count WORK_CPUS_CHECKPOINT_COUNT]
             [--restore-with-cpu {AtomicSimpleCPU,BaseAtomicSimpleCPU,BaseMinorCPU,BaseNonCachingSimpleCPU,BaseO3
CPU,BaseTimingSimpleCPU,DerivO3CPU,NonCachingSimpleCPU,O3CPU,TimingSimpleCPU,TraceCPU,X86AtomicSimpleCPU,X86KvmCP
U,X86MinorCPU,X86NonCachingSimpleCPU,X86O3CPU,X86TimingSimpleCPU}]
             [--repeat-switch REPEAT_SWITCH] [-s STANDARD_SWITCH] [-p PROG_INTERVAL] [-W WARMUP_INSTS]
             [--bench BENCH] [-F FAST_FORWARD] [-S] [--at-instruction]
             [--spec-input {ref,test,train,smred,mdred,lgred}] [--arm-iset {arm,thumb,aarch64}]
             [--stats-root STATS_ROOT] [--override-vendor-string OVERRIDE_VENDOR_STRING] [-c CMD] [-o OPTIONS]
             [-e ENV] [-i INPUT] [--output OUTPUT] [--errout ERROUT] [--chroot CHROOT]
             [--interp-dir INTERP_DIR] [--redirects REDIRECTS] [--wait-gdb]
se.py: error: unrecognized arguments: --l2_hwp_type=StridePrefetcher
```

For solution, I added the configuration for Stride Prefetcher in CacheConfig.py file



```
    # any more caches.
    if options.l2cache and options.elastic_trace_en:
        fatal("When elastic trace is enabled, do not configure L2 caches.")

    if options.l2cache:
        # Provide a clock for the L2 and the L1-to-L2 bus here as they
        # are not connected using addTwoLevelCacheHierarchy. Use the
        # same clock as the CPUs.
        system.l2 = l2_cache_class(
            clk_domain=system.cpu_clk_domain, **_get_cache_opts("l2", options)
        )

        #Add Stride Prefetcher to L2 Cache
        from m5.objects import StridePrefetcher
        system.l2.prefetcher = StridePrefetcher()

        system.tol2bus = L2XBar(clk_domain=system.cpu_clk_domain)
        system.l2.cpu_side = system.tol2bus.mem_side_ports
        system.l2.mem_side = system.membus.cpu_side_ports

    if options.memchecker:
        system.memchecker = MemChecker()
```

After that, I was able to get the output I wanted with the following command:

build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=TimingSimpleCPU --caches --l2cache --mem-size=2GB -c tests/test-progs/hello/bin/x86/linux/hello --l1d_size=64kB --l1i_size=32kB --l2_size=4MB --cacheline_size=128