CSE520: Comp Arch II – Assignment 1
(1213366971 – Sushant Trivedi)

TASK 1: Memory Hierarchy Performance Measurement

Method 1 Used: Perf Tool

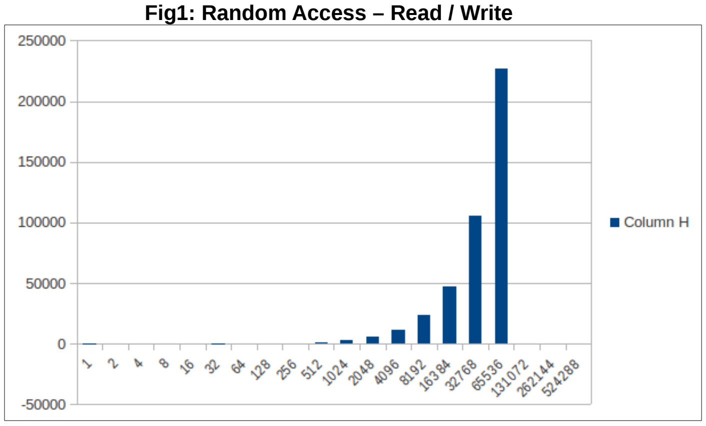
I started out with Perf Tool which measures the cycles and cpu-freq for the whole program execution. In order to measure the cache access time I had to take two programs to execute measure the cache access time.
The way to have it is one complete program that takes the array size from script and the other program to measure the overheads cause due to the memory allocation, array initialization process. By taking the difference of overheads from the complete program execution time we should be able to get a measure of average cache access time.

Problem Faced: 1. Varying CPU Frequency   2. Temporal Error

1. The first problem resulted in our clock cycle period varying resulting in an approximation in the measure. We resolved this by cycles measured / CPU Freq.
2. Temporal Error: As we measure the 2 files at 2 different instances in time, it doesn't give reliable performances. The averaging of the measure by -r flag may be resulting in a value that is not accurate for each instances of measure and thus, multiple negative cache access times.

Our findings were that Random Read access times showed the expected just at the boundary of L1 – L2 cache at 32kB boundary (on the right →). But graphs for the rest of the scenarious (Linear Access Read, Write) measurements
didn't show the expected outcomes. Much of the measurements ended up negative or too high even **though pointer chasing was followed**

Fig1: Random Access – Read / Write



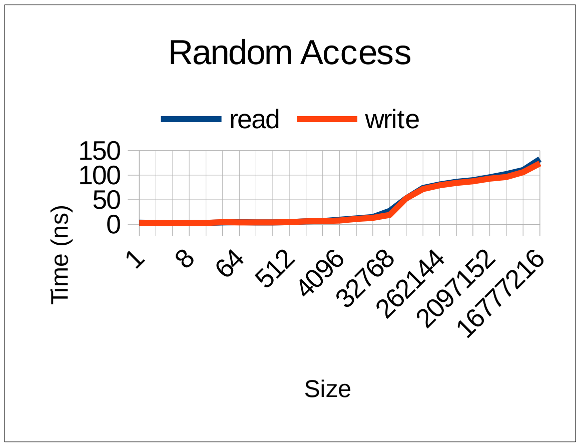
Cache Access Time = [(Complete_cycles / Freq) – (Overhead_cycles / Freq)] / [Array Size]

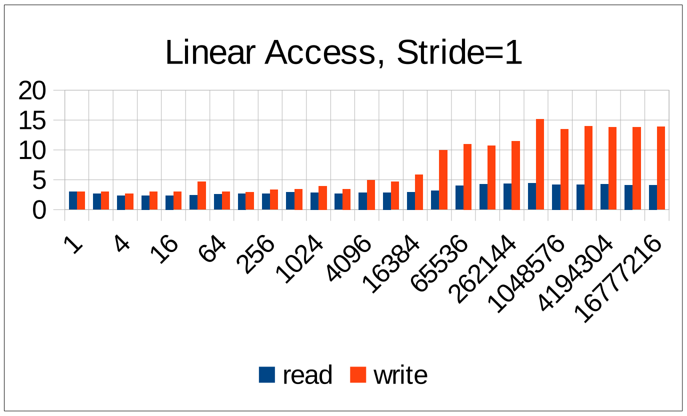
Method 2: Using Clock()

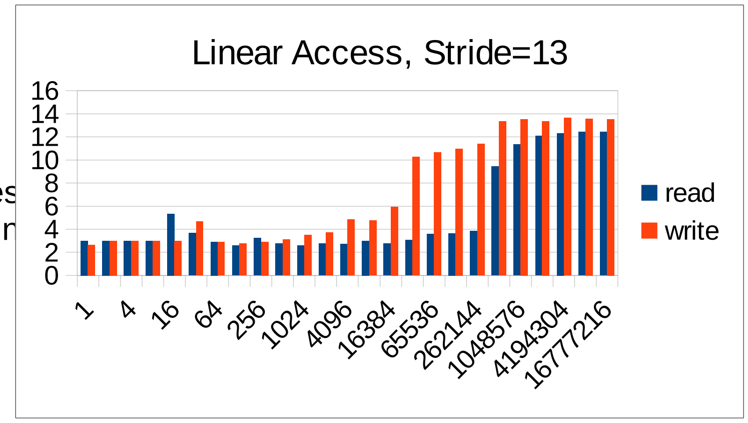
In this method we measured the clock cycles between the cache access operation and in order to tackle the 1 millisecond limit. We read through the array in repeation (atleast 100 repeatitions).

Thus, cache access time = (cycles count * stride)/ (array_size * repeatitions* cycles_per_sec) seconds

Using this method we had successfully measured cache access times for Random and Linear Reads and Writes.







Please note that for this PC
L1 cache is at 32kB, L2 cache is 256kB.

From this we can observe that the memory mountain is visible for Random and Linear Acces both. For linear, inorder to show further mountair we vary the stride parameter as shown in these figure.
As can be seen that since write operations are always time consuming than read, our write access time is more than the read time.

a) Is this average access time represents the latency or throughput of memory hierarchy? Why?
Yes, Average access time represents the latency of memory hierarchy because in memory hierarchy the slowest memory (caches followed by memory) are the lowest in the hierarchy and thus, in order to access information in it, it needs more time which in other words is the average access time.

How did you mitigate the overhead in measurement, or how did you improve the accuracy of your measurement?

1. In our program in order to mitigate the overheads in measurement we used **Cache warmup** before actually performing performance benchmarking. Search for //WARMUP in the code to verify it. This is reading through the array once so that our cache misses get reduced and we get accurate measurements.
2. Furthermore, we tried to measure with various tools and syntax codes in order to evaluate the measurements. (1. Perf tool . 2. Gem5 3. Clock())
3. We also used **taskset commands** to run the code specifically on a particular CPU to get better results

b) What makes the access time different for linear access pattern and random access pattern? Is there difference for read and write? Why?
In Catching, there is prefetching involved to deal with temporal locality which basically means you are likely to access locations around the ones currently accessed. Due to this, average access time shows less variance in values and comparitively less than the access times of the random access as can be seen in graph above for Linear Access with stride = 1. The greater the stride means its more like random access thus increased access times and closer to that of random access.

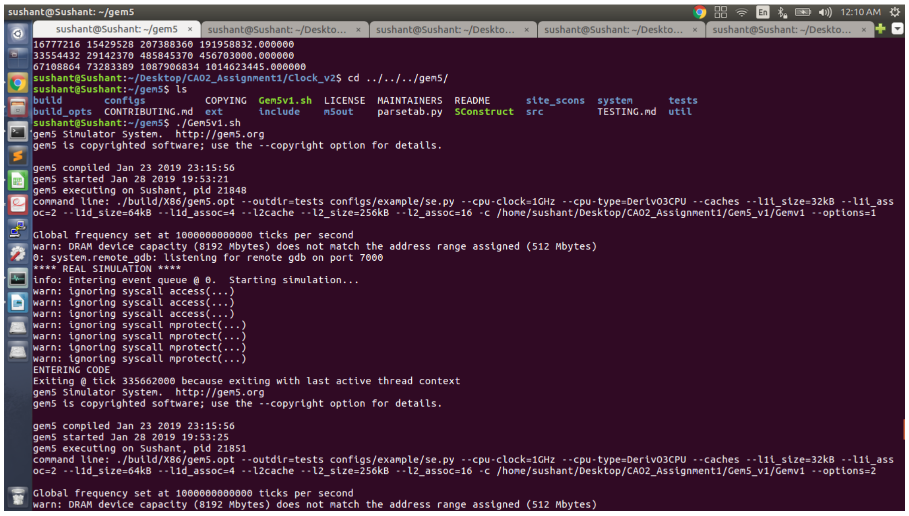
As can be seen from our graphs, is that the read times are usually smaller than the write times. This is due to the current transisitor technology limitation.

<u>TASK 2: Gem5 Simulator</u>

REFER THE GEM5 Folder for Execution code. The shell files have the configurations.
The m5out has the stat.txt, config.ini.
Please refer the excels for execution information after processing. The below image shows successful execution

The expected cache access times memory mountain has not been seen in the implementation as we received negative access times. The approach taken was similar to what perf tool required us to do.

I did notice that the is memory bus latency parameter in the stat.txt. Not completely sure as to whether this is cache access time, but I think it is.



```
16777216 15429528 207388360 191958832.000000
33554432 29142370 485845370 456703000.000000
67108864 73283389 1087906834 1014623445.000000
sushant@Sushant:~/Desktop/CAO2_Assignment1/Clock_v2$ cd ../../../gem5/
sushant@Sushant:~/gem5$ ls
build       configs          COPYING  Gem5v1.sh  LICENSE  MAINTAINERS  README      site_scons  system      tests
build_opts  CONTRIBUTING.md  ext      include    m5out    parsetab.py  SConstruct  src         TESTING.md  util
sushant@Sushant:~/gem5$ ./Gem5v1.sh
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Jan 23 2019 23:15:56
gem5 started Jan 28 2019 19:53:21
gem5 executing on Sushant, pid 21848
command line: ./build/X86/gem5.opt --outdir=tests configs/example/se.py --cpu-clock=1GHz --cpu-type=DerivO3CPU --caches --l1i_size=32kB --l1i_ass
oc=2 --l1d_size=64kB --l1d_assoc=4 --l2cache --l2_size=256kB --l2_assoc=16 -c /home/sushant/Desktop/CAO2_Assignment1/Gem5_v1/Gemv1 --options=1

Global frequency set at 1000000000000 ticks per second
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
0: system.remote_gdb: listening for remote gdb on port 7000
**** REAL SIMULATION ****
info: Entering event queue @ 0.  Starting simulation...
warn: ignoring syscall access(...)
warn: ignoring syscall access(...)
warn: ignoring syscall access(...)
warn: ignoring syscall mprotect(...)
warn: ignoring syscall mprotect(...)
warn: ignoring syscall mprotect(...)
warn: ignoring syscall mprotect(...)
ENTERING CODE
Exiting @ tick 335662000 because exiting with last active thread context
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Jan 23 2019 23:15:56
gem5 started Jan 28 2019 19:53:25
gem5 executing on Sushant, pid 21851
command line: ./build/X86/gem5.opt --outdir=tests configs/example/se.py --cpu-clock=1GHz --cpu-type=DerivO3CPU --caches --l1i_size=32kB --l1i_ass
oc=2 --l1d_size=64kB --l1d_assoc=4 --l2cache --l2_size=256kB --l2_assoc=16 -c /home/sushant/Desktop/CAO2_Assignment1/Gem5_v1/Gemv1 --options=2

Global frequency set at 1000000000000 ticks per second
warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
```