Jillian James

Marika Swanberg

CSCI 389, Computer Systems

CSCI 389 HW1

Files in Homework 1:

hw1.cpp - Program that takes two arguments "size" and "iters". It creates a buffer of length "size" and times how long it takes to perform "iters" number of accesses to the buffer. In order to measure the latency of memory accesses we needed to foil the compiler so it wouldn't prefetch the array values. To do this, we made a hash function to probe the buffer and access the values stored in memory. Run with the commands. See comment at the beginning of the file for the commands used to run the program and the specifications for the system it was run on.

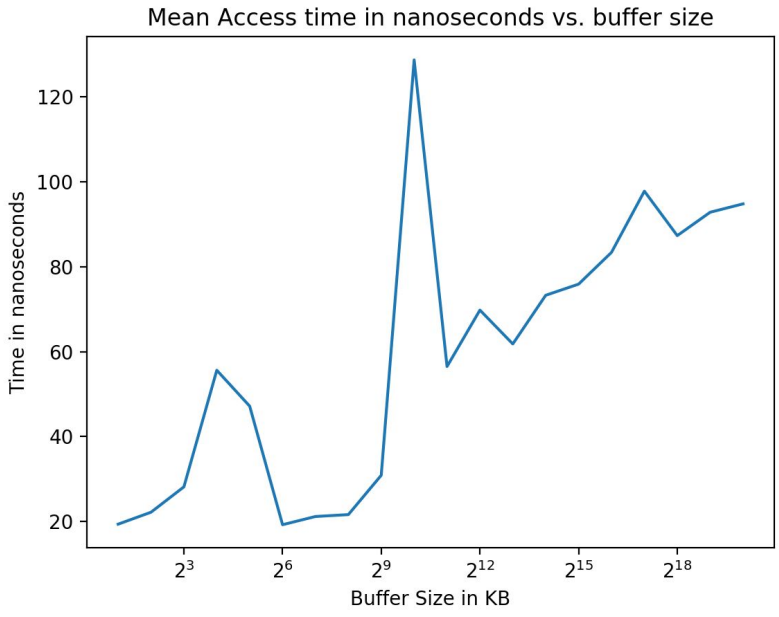
Hw1shell.sh - A shell program that contains all of the calls to run hw1.cpp with its “size” and “iters” arguments. To run, navigate to the directory that hw1shell.sh is in and enter the following into terminal:

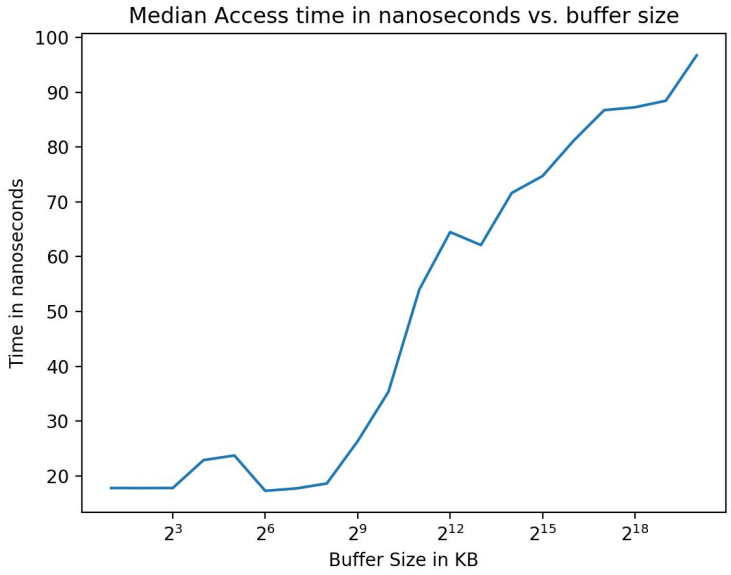
```
chmod +x hw1shell.sh
./hw1shell.sh
```

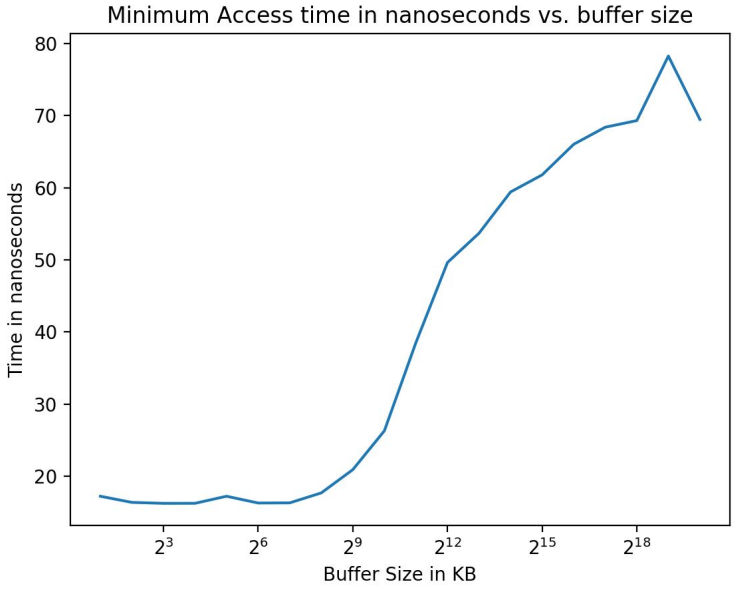
Data - Directory with 11 csv files that have the contain the output of a run of a run of ./hw1shell.sh

Plot_buffer.py - Python program that uses pyplot to create the graphs in the Graph directory using the csv files in the Data directory.

Graph - Directory containing the following graphs made with the eleven csv files in Data:




Median Access time in nanoseconds vs. buffer size
Time in nanoseconds
100
90
80
70
60
50
40
30
20
2^3
2^6
2^9
2^{12}
2^{15}
2^{18}
Buffer Size in KB


Minimum Access time in nanoseconds vs. buffer size
Time in nanoseconds
80
70
60
50
40
30
20
2^3
2^6
2^9
2^{12}
2^{15}
2^{18}
Buffer Size in KB

Part 3 - Analysis

1. Try to identify the sizes of your CPUs different caches from the graph and explain your reasoning.

First note that as each value in the buffer is between 0 and RAND_MAX (32767), the compiler will make each integer of type int16, meaning each element of the array is 2 bytes in memory. So, an array of size 2^10 is 2KB and an array of size 2^29 is 1048576 KB, which you will note is the range of the x-axis on our minimum graph. The minimum graph also has several notable step increases that may correspond to the different cache sizes. The time per memory access in nanoseconds is lowest in the range 2KB to 128KB. Therefore, it is likely that the L1 cache is probably <128KB. Then the graph appears to step increase in the range 128 KB to 512 KB. Thus L2 is likely <512KB. Then the graph appears to step increase in range 512KB to 2048KB so L3 is likely to be <2048KB. Finally, the graph steps again at 8192KB and appears to slope off before increasing sharply 131072KB. That last sharp increase may be caused by an access to DRAM. So we're inclined to believe L4 is <131072KB.

2. Did you get approximately the same performance as these numbers? Explain why or why not.

Not exactly. Our access time for the L1 and L2 caches is a bit high and the access time for main memory is a bit low. According to the link provided, L1 accesses should be about 0.5 nanoseconds but in our tests they took a bit under 10 nanoseconds. Similarly, accesses for the L2 cache should have taken 7 seconds but in our tests they took a bit under 25 seconds. This is likely due to due branch misprediction and the hash value our code computes. Our code purposely tries to foil the prefetcher, so it is likely that the occasional branch misprediction added penalty to our L1 and L2 latency measurements. According to the link, these penalties are about 5 ns. Additionally, a hash value is computed in our loop, and although it doesn't reference memory or call another function, a typical instruction is 1 ns so a few nanoseconds were probably added to the latency. As for why our memory access was lower than 100 seconds, the minimum graph shows the best case of the observed latencies, so this may just demonstrate the best case performance in which some values of the >131072KB size buffer were stored in cache while the rest were only in memory.

3. Look up which specific CPU you have in your test computer and what its cache sizes are supposed to be, and compare them to your guesses above in number one.

   The terminal command "sysctl machdep.cpu.brand_string" tells us that the CPU we used is the Intel(R) Core(TM) i5-7267U CPU @ 3.10GHz. After looking it up we found that the cache sizes were:

L1$	128 KiB
L2$	512 KiB
L3$	4 MiB
L4$	64 MiB

   This confirms my observations in number 1. In each case we remarked the cache was less than the size noted here when we should have said it was less than or equal to the size noted here, but we did observe that a step increase occurred at each of these points along the x-axis indicating a change in latency indicative of an access to a new level of the cache.