CSE240B Final Project Report

Zifeng Li

Characterize the Cache Performance of a CPU Under Shared and Non-shared Conditions

Documentation

1. Latency Measurement:
   The program measures both the read and writes latency. The read latency is measured by performing the operation that reads the value of each element of an array and does a simple arithmetic operation on them. The read and write latency is measured by performing the operation that reads the value of each element of an array, modifies them, and then stores them back to the array. Since the performance of such operations can vary due to many reasons. These operations are performed with many iterations, and the average latency is reported. We will also explore the impact of utilizing multi-threading on memory and cache performance.
2. Program options:
   a. Read or read & write: this is the option that switches between measuring only the read latency and measuring the read & write latency. (-m for read & write)
   b. The number of threads to use: this is the option that sets the number of threads to use to perform the read or write operations. Use "-t" followed by a number to set the number of threads to use. If not set, then the program will use a single thread.
   c. The size of the data array: this is the option that sets the size of the data array where the program will operate. It takes a single number n. The size of the data array will be 2^n.
3. Program Limitation:
   a. Multi-threading package: the program utilizes the POSIX Threads package. "pthread.h" must be present the system's include path. If using gcc to compile, use the "-lpthread" to link its library file.
   b. The maximum number of threads: currently, the maximum number of threads it supports if 16. If a larger number of

threads is desired, modify the predefined MAX_THREAD variable in the cachetime.c file.

Results:

1. Read Latency

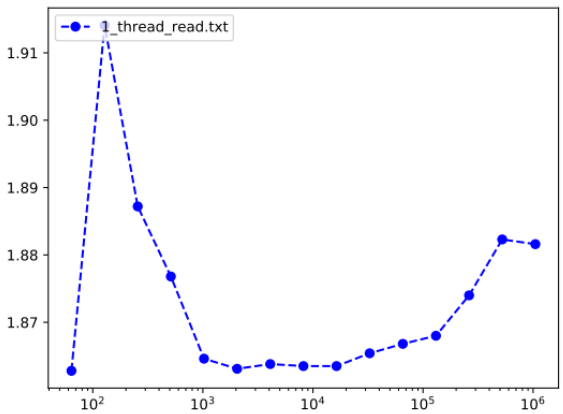


Figure 1 Read Latency of single thread.

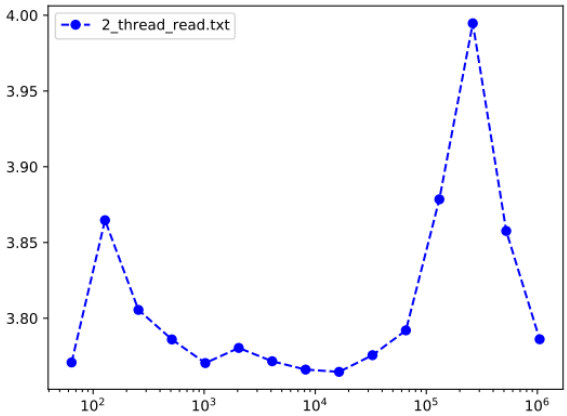


Figure 2 Read Latency of 2 threads.

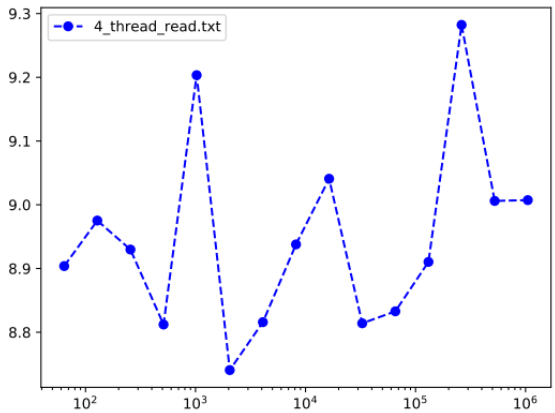


Figure 3 Read Latency of 4 threads.

Figure 4 Read Latency of 8 threads.

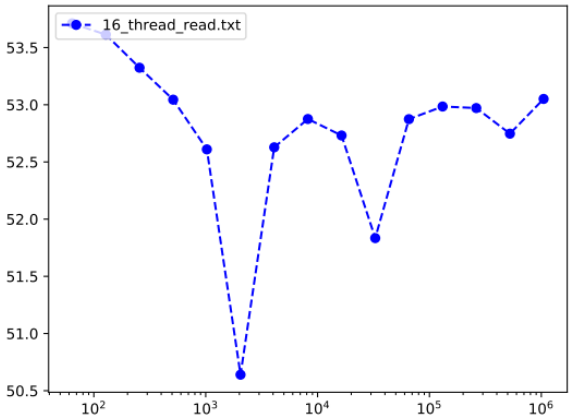


Figure 5 Read Latency of 16 threads.

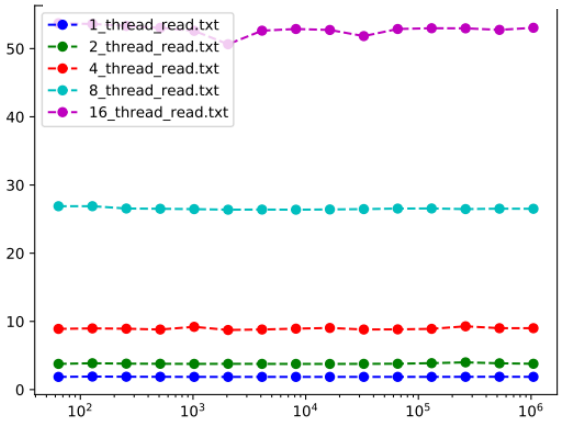


Figure 6 Read Latency of various number of threads.

2. Read & Write Latency:

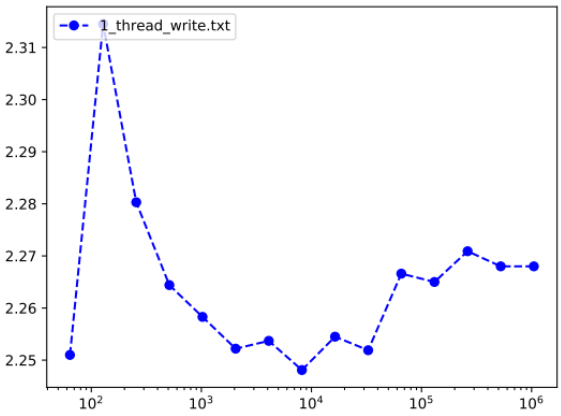


Figure 7 Write Latency of single thread.

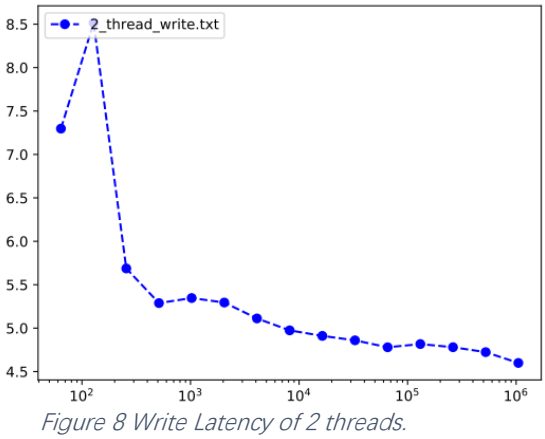


Figure 8 Write Latency of 2 threads.

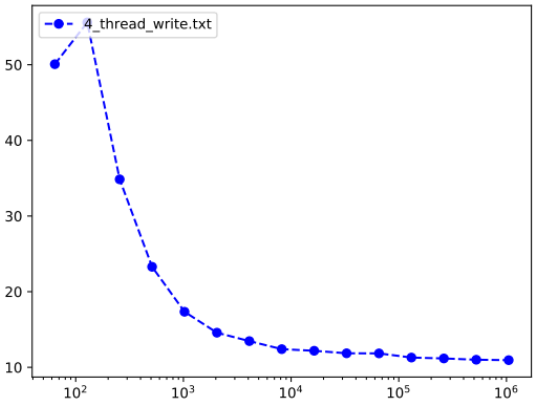


Figure 9 Write Latency of 4 threads.

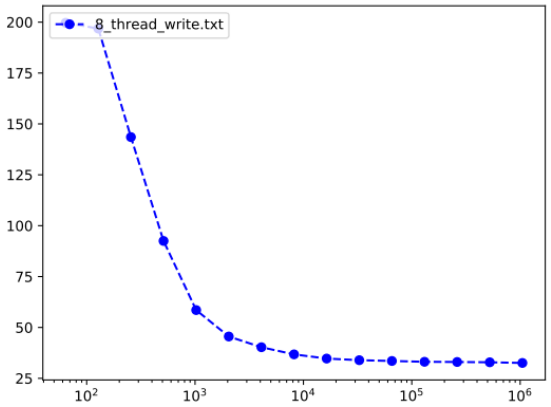


Figure 10 Write Latency of 8 threads.

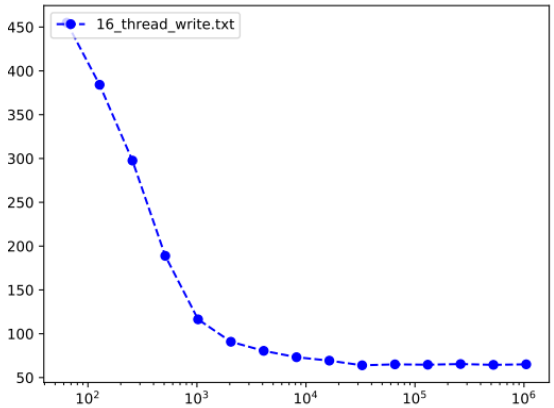


Figure 11 Write Latency of 16 threads.

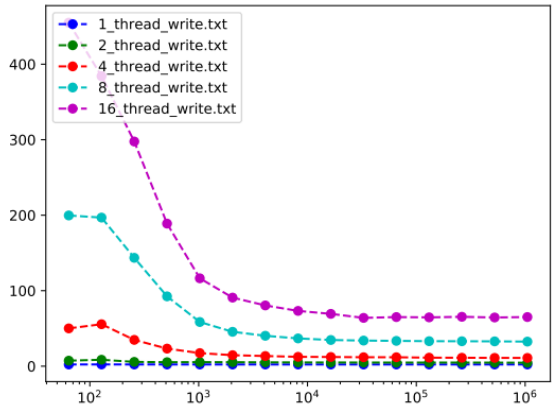
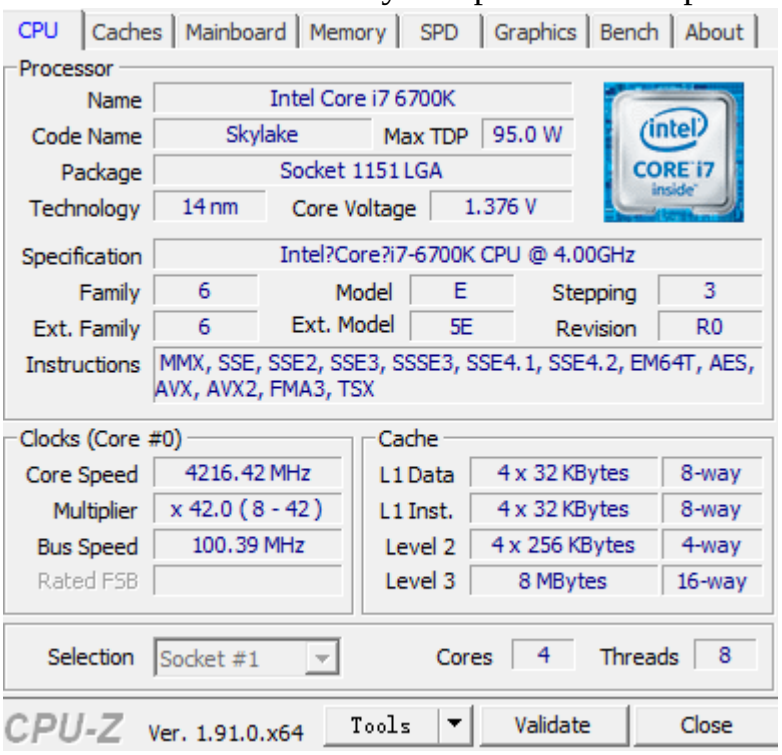


Figure 12 Write Latency of various number of threads.

3. Analysis:
   In the reading performances using a single thread and using two threads, two possible cache sizes exist in the memory hierarchy are around 32KB and 256KB. There is one noticeable latency jump between the array size of 128KB and 256KB (around 10^5). Besides, there is also a small latency improvement between the array size of 16KB and 64KB (a bit after 10^4 point). Therefore, a cache hierarchy may exist with these size configuration. The reading behaviors using the array size smaller than 2K are not as clear as the later part. In the result of using a single thread and two threads, we can see an abnormal latency jump with an array size of 128B and then a gradual decrease. One possible explanation may be that the overhead of the for loops may overshadow the latency produced by the array data reading.

   Here is a screenshot of my computer's CPU specification:



   We can see that the L1 cache for one CPU has a size of 32 KB, and the L2 cache has a size of 256 KB. These cache size roughly conform with the two jumps showed in the single-thread reading performance and the two-thread reading performance. The first latency jump happens because the L1 level cache cannot fit the whole data array, which results in many L1 cache misses. The second latency jump happens because of the same reason, but the problem is the L2 cache

is not large enough.

From the combined graph (figure 6), we can see the average read latency using a different number of threads increases almost linearly for using one, two, and four threads. But, since my computer only supports a maximum number of 8 threads and other applications are running on the machine, the performance of using 8 threads and 16 threads seems not in a linear relationship with the previous ones.

In the read/write performances, the single-thread performance, to some extent, also shows the cache hierarchy configuration showed in the read-only part. Two noticeable jumps happen around using 32KB data array and 256 KB data array. But for the performances reading with multi-threading, this feature doesn't exist. But, another interesting feature is that the latencies show a decreasing trend with growing data array size.

One possible explanation for such a behavior is that for multi-threading read/write operation. The OS must enforce store atomicity. So, if two or more threads want to write to the same cacheline, then the write operations will be serialized. For using a small data array, the possibility of two threads are writing to the same cacheline is larger than using a larger data array. This can results in a higher write latency when using a small data array. Besides, this possibility will also increase with the number of threads.

So, if we fix the number of threads, we can see that the write latency decreases with larger and larger data array. If we fix the data array size, the average read/write latency increases with the number of threads.