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

Hanyang Xu

1. Program Documentation:

In general, the program works in the following way: 1) Create and randomized an array of pointers 2) Access this array in read-only or read-and-write mode with Pointer Chasing technique on each thread 3) Measure the duration of the program to complete Pointer Chasing of the entire array on each thread 4) Sum up the time of all thread 5) The access time is calculated as: Total time / Number of thread / Array Size. 6) Memory Size vs Latency output is outputted to csv file with name "t_thread number_r" for read only and "t_thread number_rw" for read and write.

The minimal memory size allocated with array is 64B and the maximum memory size allocated with array is 64MB. The memory allocation doubles for each measurement. Since the test computer has 12MB L3 cache, it is more than enough to expose all the limitation of the test machine cache system.

Pointer chasing refers to a common sequence of instructions that involves a repeated series of irregular memory access patterns that require the accessed data to determine the subsequent pointer address to be accessed, forming a serially dependent chain of loads. It is implemented by 1) Create an array of void* pointers with ordered the indices 2) Randomized the indices by swapping values of two array element randomly 3) Override the value of array element to be the address of array element indicated by its index value. Then, we access each element based on its stored address iteratively, liking chasing a pointer. [1]

This program has the following compile option:

- *-t $(number of thread)* This option specifies the number of threads to run this program. It is required. If the user need 1 thread to run the program, then the following instruction is needed: *./CacheAccess -t 1*
- *-rw* This option specifies whether the program does a read-only measurement or a read-modified-write measurement. It is not required. If the option is not provided, the default is a read-only measurement. If the user need 1 thread to run a read-write measurement, then the following instruction is needed: *./CacheAccess -t 1 -rw*

There is two known limitation to this program. 1) First, it is not fault proved. The program is only guaranteed to work if user follow the instruction correctly. 2) it requires third party multithread framework OpenMP, it could be difficult to setup for different systems.

2. Cache Performance Results

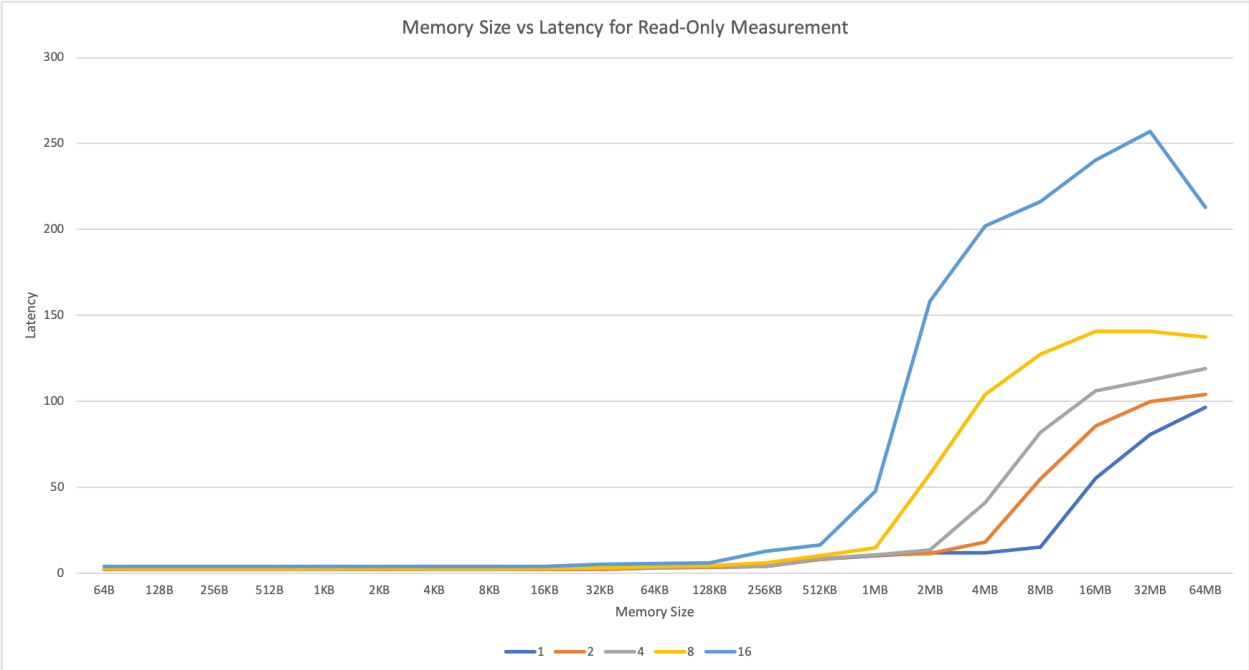

Figure 1 Read Only Latency for Different Data Size

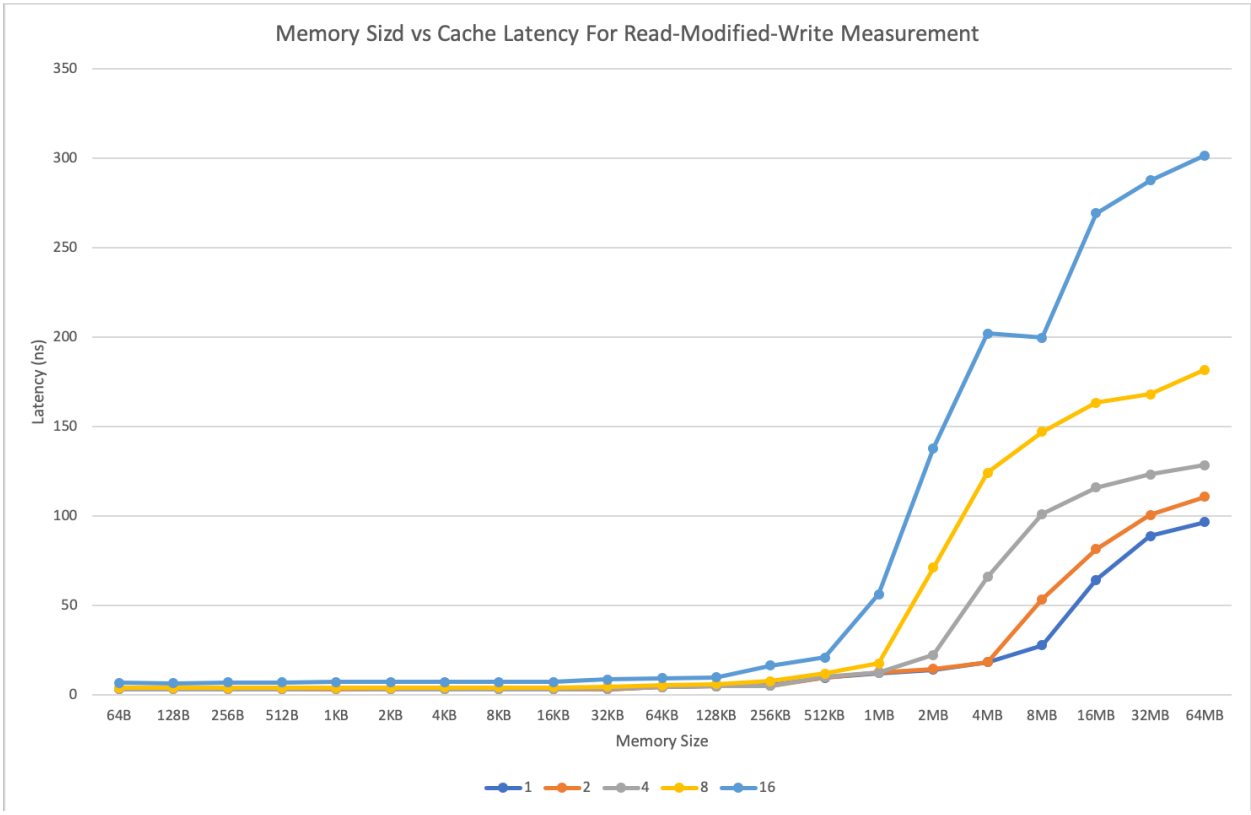

Figure 2 Read-Modified-Write Latency for Different Data Size

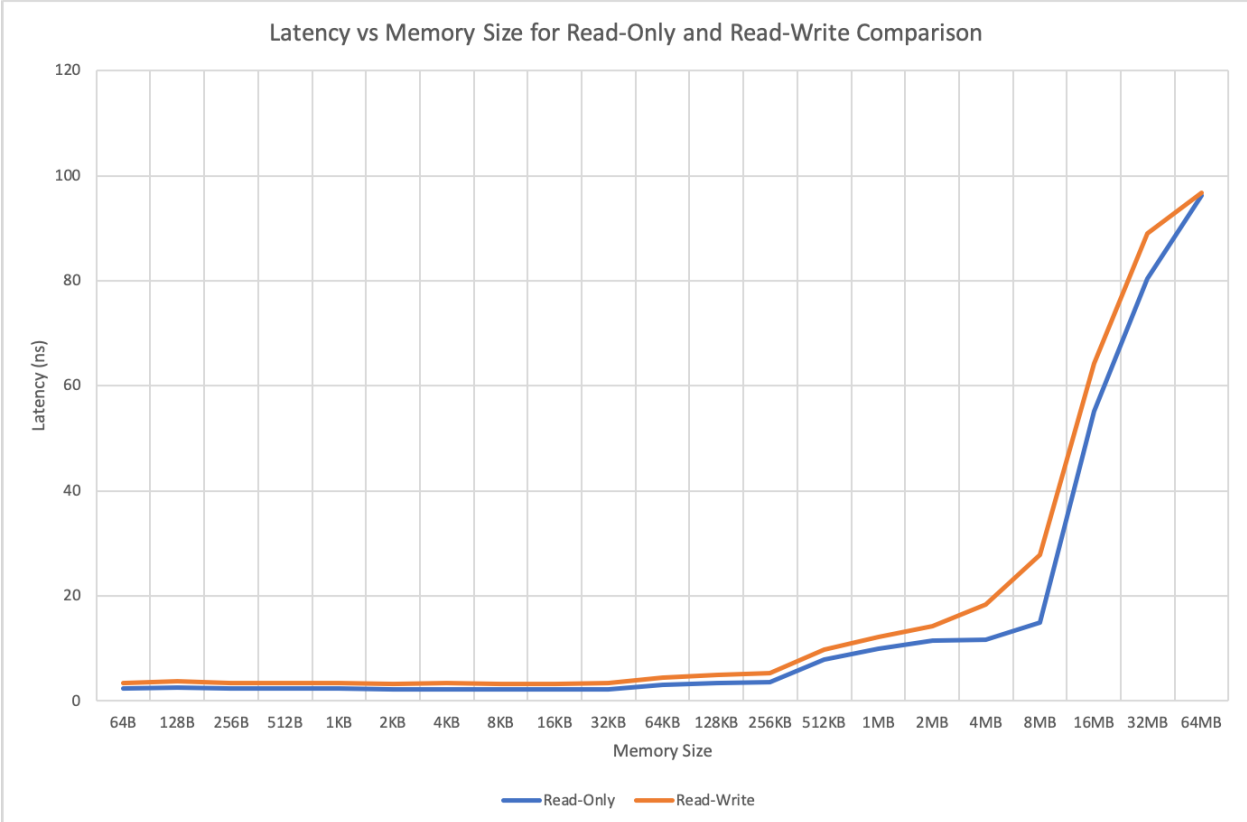

Figure 3 Comparison Between Read-Only and Read-Modified-Write Operation on Different Data Size for Single Thread

3. Result Analysis

For Read-only measurement, we can tell from Figure 3 that there are four area of interest. There are four distinct curves in Figure 3. The first one is from 64B to 256KB, the second one is from 256KB to 2MB, the third one is from 2MB to 8MB, and the last one is from 8MB to 64MB. These curves tell that the test system has three level of cache with the L1 cache being around 256KB, L2 cache being around 2MB and L3 cache being around 8MB. The table below shows the cache system of the test system. It marches pretty well with the output of the test program.

CPU Model:	Intel Core i7-9750H
L1 Cache:	192 KB for Data
L2 Cache:	1.5MB
L3 Cache:	12 MB

[2]

Comparing the Read-Only and Read-Modified-Write measurement, we can see one major differences. 1) The Read-Modified-Write has greater latency compared to Read-Only. This is because the write operation in Read-Modified-Write takes away some bandwidth while the Read-Only can put all cache bandwidth into read operation.

Read-Only operation seems to be slightly more stable than Read-Modified-Write. It is because the Read-Only has only one operation to be done while Read-Modified-Write has to interlevel two operations.

One behavior I noticed is that multithreading effectively reduces the cache size. For example, in Figure 1, The latency for 1 thread reading 8MB of data is equivalent to 2 thread reading 4MB of data. The latency of 2 thread reading 8MB of data is about twice of 1 thread reading the same amount of data.

Methods to sanity check my results include: 1) Running the test code on different machines and observe the relationship between the outputs and the actual cache system of the test machines. 2) Run the real program on the same test machine that includes a lot of random data R/W, graph the cache latency and compare with the result presented here.

4. Source:

[1] https://github.com/afborchert/pointer-chasing
[2] https://en.wikichip.org/wiki/intel/core_i7/i7-9750h