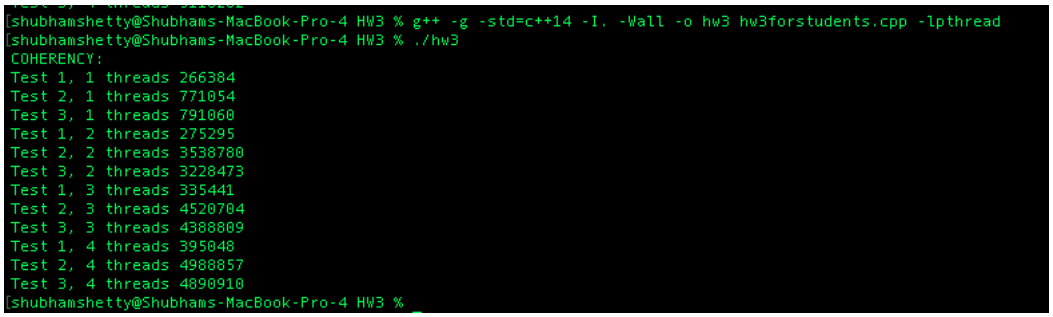
Systems for Data Science - HW3

Cache Coherency

Shubham Shetty

February 17 2021



Test #	1 Thread	2 Threads	3 Threads	4 Threads
Test 1	0.266384	0.275295	0.335441	0.395048
Test 2	0.771054	3.538780	4.520704	4.988857
Test 3	0.791060	3.228473	4.388809	4.890910

Table 1: Execution time in seconds for different tests and different number of threads

System Details -

- *Processor*: 2.5 GHz Dual-Core Intel Core i5
- *Max number of threads*: 4

As we can observe from the table, for each test the time taken for execution increases as the number of threads increases. This can easily be explained by cache coherency issue of parallel access as the overhead involved in ensuring that the data caches containing same data are all in sync keeps increasing as the number of resources accessing the cache increases. As threads increase, each thread will have its own cache containing the data being accessed and the associated overhead in keeping all these seperate cahces in sync will also increase.

In test 1, threads only read data in cache. Hence the relative change in execution time after addition of threads is not extremely significant as no overhead would be involved in maintaining cache coherency.

In test 2, threads are both reading and updating data in the data location. This involves both updating cache and writing back to the data location, hence the amount of overhead to maintain the cache coherency is higher. We can see a significant jump in processing speed between 1 thread and 2 threads (around a 360% increase).

For test 3, threads are accessing distinct elements in the array, hence similar amounts of overhead can be expected. As threads are added, there is a significant increase in processing time.

From these examples it is obvious that parallel processing invites significant overheads. It becomes important from a programming perspective to ensure these overheads are minimised in order to fully utilise the benefits of parallelism.