COMPSYS 304 Assignment 3 Eric Zhao

Task 1

1. Name of my processor: Intel Core i7-7700HQ

2. Cache size of my system:

LEVEL 1 Cache Size = 256 KB,

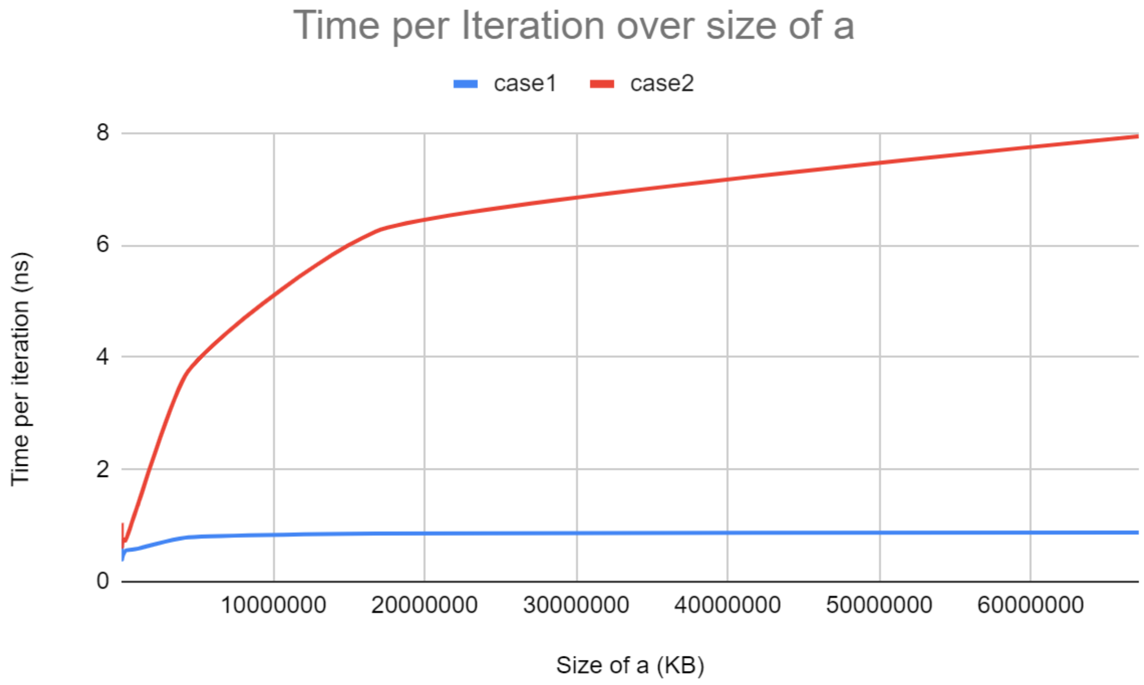
LEVEL 2 Cache Size = 1.0 MB,

LEVEL1 Cache Size = 6.0 MB

3. Table of measured time per iteration for each Case

N	Size of a (Bytes)	Case1: Time per iteration (ns)	Case 2: Time per iteration (ns)
1024	4096	0.7655471563	1.0449439287
4096	16384	0.415253453	0.6330665201
16384	65536	0.4565226845	0.7596390788
65536	262144	0.5514812074	0.7349299267
262144	1048576	0.5871379472	1.3480212147
1048576	4194304	0.784873464	3.6708740936
4194304	16777216	0.8605570656	6.2431570313
16777216	67108864	0.8785178807	7.9369493733

4. Chart of time-per-iteration over size of a:



5. As shown in the graph above, the time per iteration of Case 1 stays in a constant trend for the size of a in the range of 4KB to 64MB. The reason behind this can be caused by the different levels of caches. From 4KB to 16KB, only Level 1 cache changed, as Case 1 goes linearly through the array, the address of the following data will be predictable. Therefore, Case 1 required shorter time to finish every iteration.

In terms of Case 2, the curve shows us a unpredictable trend as when size of a changes from 1MB to 4MB, the time per iteration increased from 0.74 to 1.34ns which is a big increment in the graph. I believe this is because the processor needs to figure out values from lower memory locations without any prediction.

Overall Comparison between Case 1 and Case 2:

- Going linearly through the array in Cases 1 takes shorter time to finish iterations compared to random access method in Case 2.
- Case 2 is much slower than Case 1 mainly due to its unpredictable behaviour, based on the feature of randomly access, higher timing cost will be resulted.

Task 2

1. Straight forward implementation took : 8.13 seconds

My implementation used 4 for loops to evaluate the sum of the given equation. However, when k value increments, array b’s row number changes, and the programme gets slowed down significantly.

2. Temporary matrix implementation took : 4.70 seconds

A temporary matrix of the same size as the array a, b and c is created in order to sum up the value, volatile variables is therefore not used. As a result, total timing cost reduced by a lot after the creation of temporary matrix.

2. Blocking and Temporary matrix implementation took : 3.16 seconds

After checking the specifications of my computer, I decided to have a blocking size of 64. I implemented multiple layers of loops in order to go through the array within an appropriate size. The programme is optimised as a result of increasing hit rate.