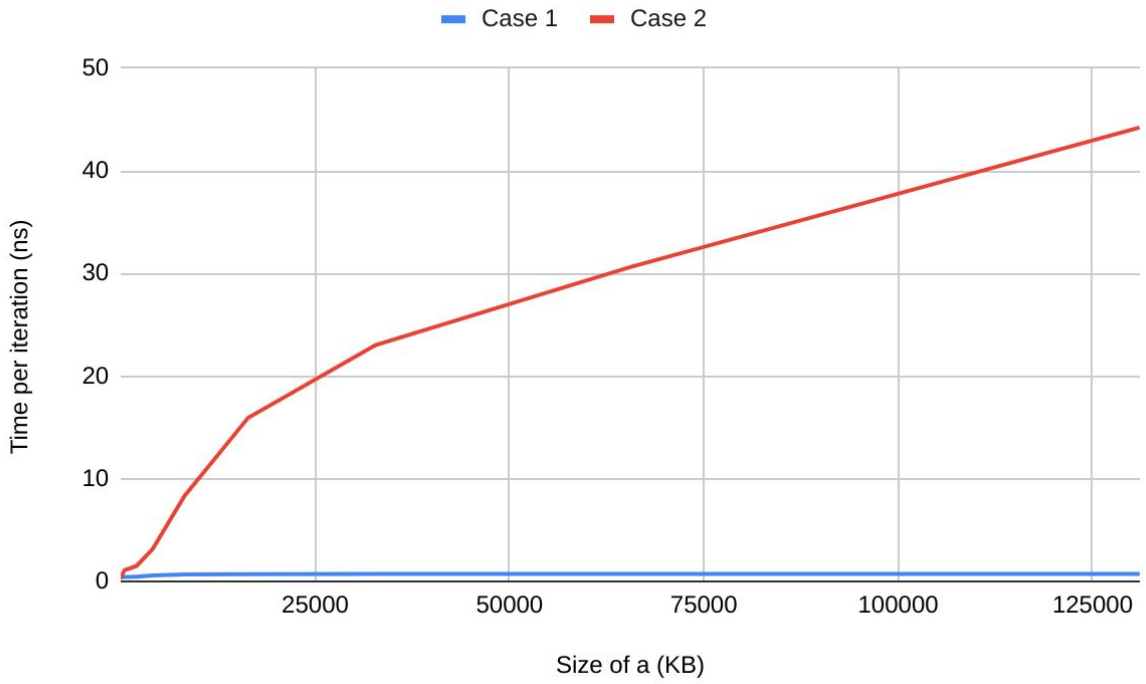
Compsys 304 Assignment 3

Harpreet Singh (hsin849)

Task 1

Processor: Intel Core i5 8250U
L1d cache: 128 KB
L1i cache: 128 KB
L2 cache: 1 MB
L3 cache: 6 MB

N	Size of a	Time per iteration Case1	Time per iteration Case2
2048	8 KB	0.40305	0.3842
4096	16 KB	0.38479	0.38363
8192	32 KB	0.39446	0.40421
16384	64 KB	0.3908	0.48815
32768	128 KB	0.39214	0.56901
65536	256 KB	0.41353	0.7808
131072	512 KB	0.42673	1.09614
262144	1 MB	0.42127	1.2368
524288	2 MB	0.44853	1.51207
1048576	4 MB	0.56756	3.13334
2097152	8 MB	0.67682	8.32452
4194304	16 MB	0.70511	15.94315
8388608	32 MB	0.71566	23.03173
16777216	64 MB	0.71552	30.62748
33554432	128 MB	0.71885	44.2193



In the above graph, it can be seen that case 1(sequential) takes significantly less time as compared to case 2(random). This is because when looping sequentially, adjacent elements which are to be accessed in the near future will be loaded in the cache. This increases the probability of a cache hit significantly. Whereas in case 2, the elements are accessed randomly which means storing adjacent elements in the cache would have no benefit. The probability of a hit is very low and there would mostly be misses.

It can also be seen that the time per iteration in case 1 stays the same whereas it increases in case 2. This is because as the size of a increases, there would be more cache misses for case 2. The miss to hit ratio would increase and the probability to get a hit would significantly decrease . Case 1 would have a consistent ratio of misses and hits and therefore will not be affected. The probability of receiving a miss would still stay low.

Task 2

Case1: 3.91023 secs

- In this case, both the matrices a and b are arranged in a row major order. In the matrix multiplication formula, b is accessed in a column major order which would decrease the probability of getting a hit and therefore be slower.

Case2: 2.64802 secs

- Transposing the matrix b improved the speed. This is because, the new transposed matrix is now arranged in a row major order and therefore will be faster.

Case3: 1.62872 secs

- L1 cache line size = 64
  Total floats it can store = 64 / 8 = 8
  Therefore k in a k by k block should be 8 for this particular computer.
  Dividing the above matrices into smaller sub matrices and computing them would improve the performance and speed. The length of the sub matrix should be decided based on the line size of the cache. The length of the sub matrix will be equal to the number of floats the cache line can store. Required elements would therefore be in the cache and improve fetching. Hence, this would improve speed and performance.