Part A

Question 1

Posix_memalign is a function in the POSIX standard library that is used to align the memory to ensure that data is stored in a way that corresponds to the cache's line size. If the data is not aligned, it may span multiple cache lines, resulting in more cache misses and slower performance.

Question 2

In naive matrix multiplication, its for loop for row cannot well utilize cache. When the dimension of the matrix is large, cache cannot store all elements in one row. Then, it keeps generating cache misses when computing the inner product between row and column. This is becuase after it iterates to the end of a row, the values previously fetched has been removed.

In addition, for each row, it computes the inner product with each columns. However, after computing inner product of row i with all columns, it needs to fetch all column valus from the main memory again when computing the inner product with row i+1.

Starting from matrix with size 256, the results show many noticeable differences. And this difference increases when size of matrix gets larger.

Question 3

When tile size is small, the megaflops are small and increases as tile size increases. When tile size is large, the megaflops are also small and decreases as tile size increases. The optimal tile size is 16. Details could see the following results:

```
Tile    Matrix      MFLOPS
========================================
Naive   128         497.69888260000000000000
Naive   256         387.77347260000000000000
Naive   512         312.29958180000000000000
Naive   1024        331.38493000000000000000
Naive   2048        99.93567560000000000000
========================================
4       128         336.59235420000000000000
4       256         323.61107060000000000000
4       512         321.11340440000000000000
4       1024        322.97265080000000000000
4       2048        235.86440740000000000000
========================================
8       128         362.84272000000000000000
8       256         373.39070600000000000000
8       512         373.41183800000000000000
8       1024        372.01568180000000000000
8       2048        350.17315900000000000000
========================================
16      128         756.82431240000000000000
16      256         764.83167720000000000000
16      512         656.76755540000000000000
16      1024        658.96175100000000000000
16      2048        627.13302140000000000000
========================================
32      128         868.63562680000000000000
32      256         638.35768860000000000000
32      512         540.73615500000000000000
32      1024        541.26108660000000000000
32      2048        434.85411280000000000000
========================================
64      128         605.75986940000000000000
64      256         576.35119860000000000000
64      512         483.58540780000000000000
64      1024        415.29168100000000000000
64      2048        398.29786260000000000000
```

Question 4

Yes it somehow changes the performance. But the differences are not significantly large.

Metric: MegaFlops/s.

Row-Column Order

16*16: 592.40

32*32: 268.43

512*512: 264.87

1024*1024: 316.66

2048*2048: 93.95

Column-Row Order

16*16: 241.97

32*32: 289.95

512*512: 284.29

1024*1024: 309.01

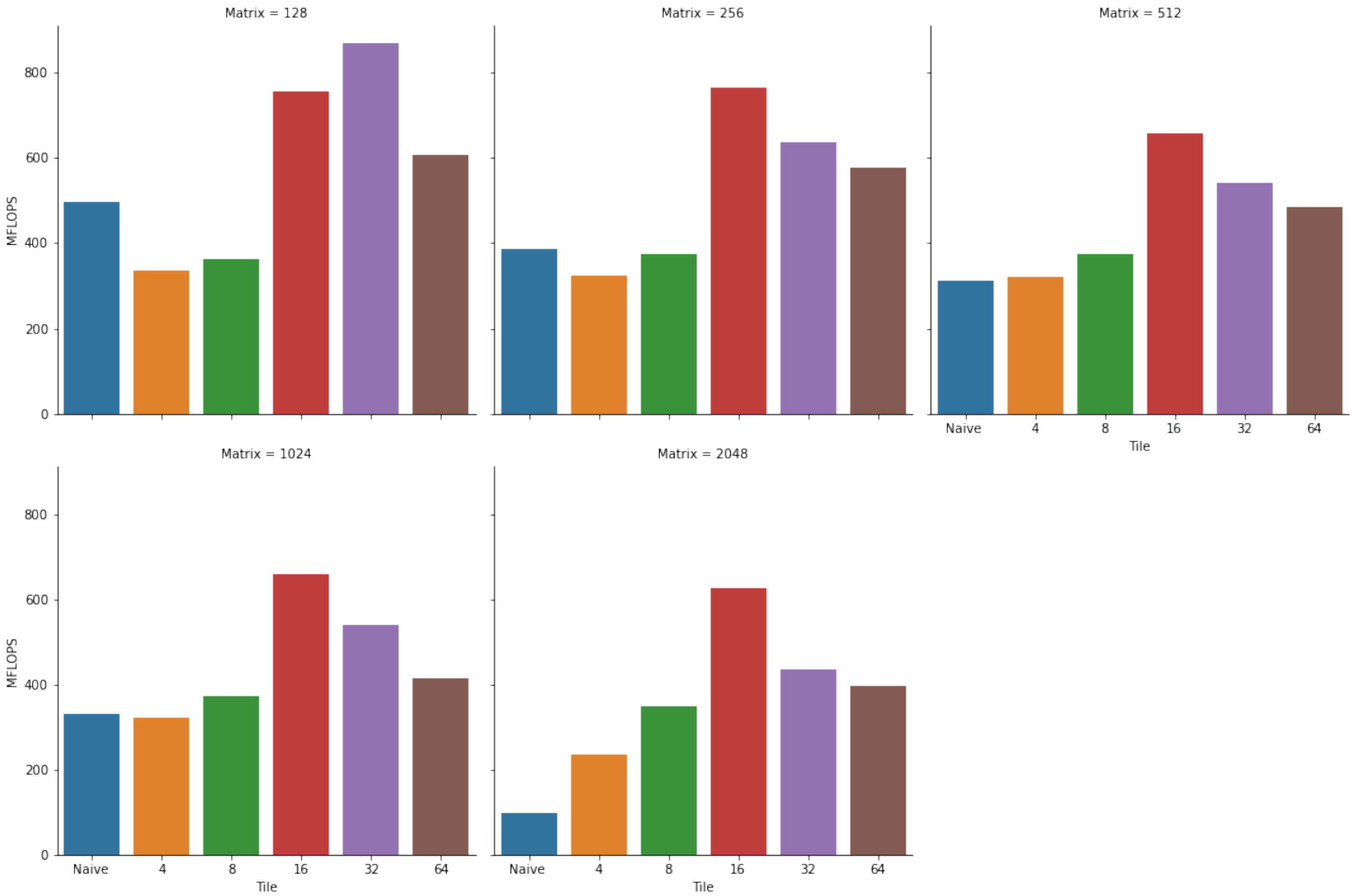
2048*2048: 86.34

Except when matrix is 16*16, row column-order is significantly faster than column-row order, in other small matrix multiplication, column-row order is slightly faster than row-column order.

However, as dimension of matrix is large, row column-order is consistently faster than column-row order.

I think this might due to spatial locality. When matrix dimension is small a cache line could store entire row and thus, fix column at outer loop is more efficient than fix row at outer loop. However, when dimension of matrix is large this advantage diminishes, because it cannot store the entire row in a cache block.

Part B



The above subplots are generated by seaborn.catplot function, which compare the MFLOP for different variants and tile number with the matrix size.