Applied GPU Programming - Assignment 4 - Group 19

Group members and contributions

We did the assignment together and in-person, therefore both of us contributed the same amount of work.

- Diogo Gaspar - dgaspar@kth.se (mailto:dgaspar@kth.se) - 50%
- Diogo Melita - diogogm@kth.se (mailto:diogogm@kth.se) - 50%

Exercise 1

Question 1: X = 800, Y = 600

Per grid, the number of blocks along, respectively, the horizontal (X) and vertical (Y) axis can be calculated as follows:

- X: $\lceil \frac{800}{32} \rceil = 25$
- Y: $\lceil \frac{600}{16} \rceil = 38$

Furthermore, each block holds 32×16 threads, and since each warp consists of 32 threads, as per the assignment's statement, we have 16 warps per block. As such, the total warps will be $25 \times 38 \times 16 = 15200$.

Now, as for control divergence, we have that, in each block, two rows of threads are in the same warp (since each row will have 16 threads, and each warp holds 32). We also have that $800 \% 32 = 0$, and that $600 \% 16 = 8$, which leads to there being no horizontal divergence, only possibly vertical one – and with vertical divergence of 8, we have that 8 rows execute one path, and the other 8 execute the alternative. However, since every two rows are in the same warps, there won't be control divergence here either! As such, in these conditions, there won't be any warp with control divergence.

Question 2: X = 600, Y = 800

Now, in the X-axis, we have $\lceil \frac{600}{32} \rceil = 19$ blocks, and $\lceil \frac{800}{16} \rceil = 50$ in the Y-axis.

Since the image width (600 pixels) does not align with the block width (32 pixels), the last block in each row will have partially filled warps, which converts to $24\%16 = 8$ warps. As such, the total warps with control divergence will be $50 \times 8 = 400$.

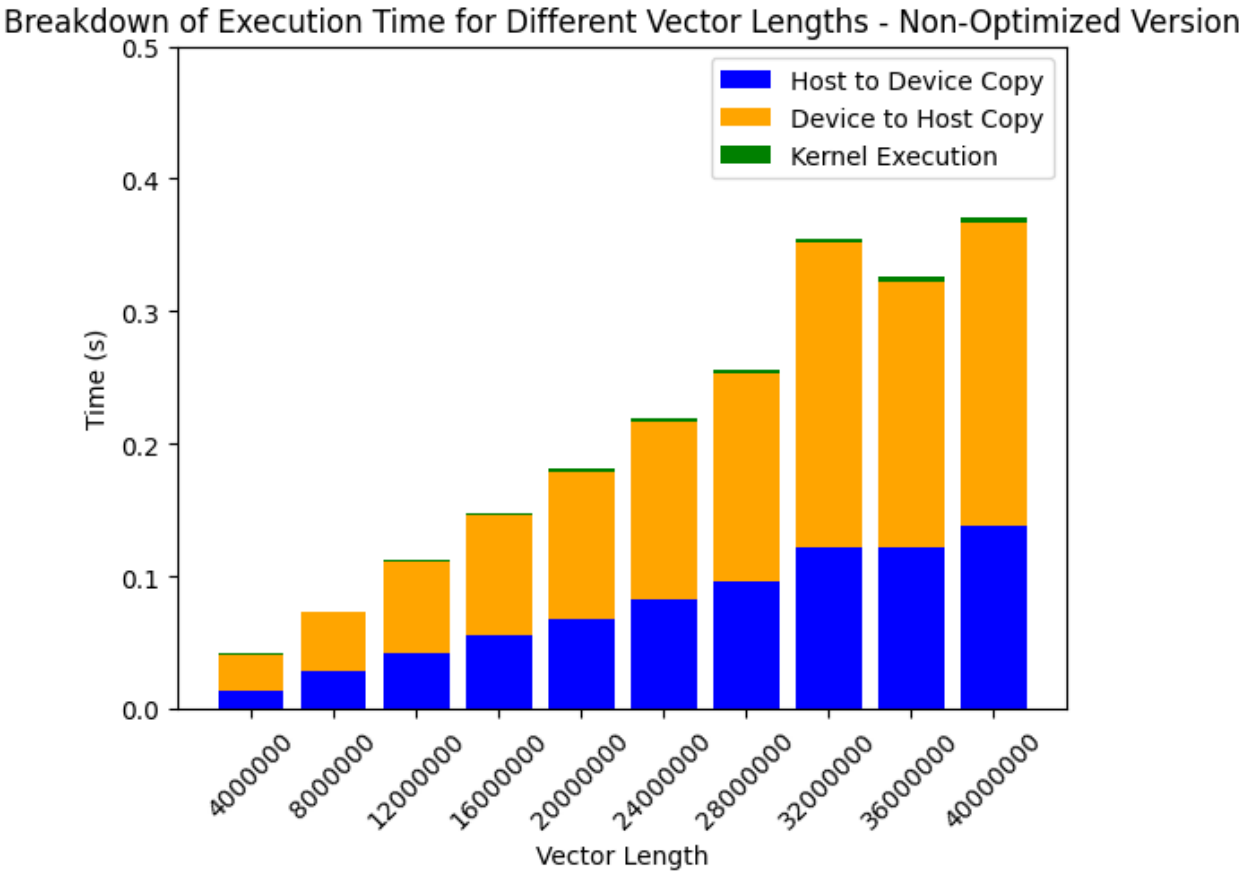
Question 3: X = 600, Y = 799

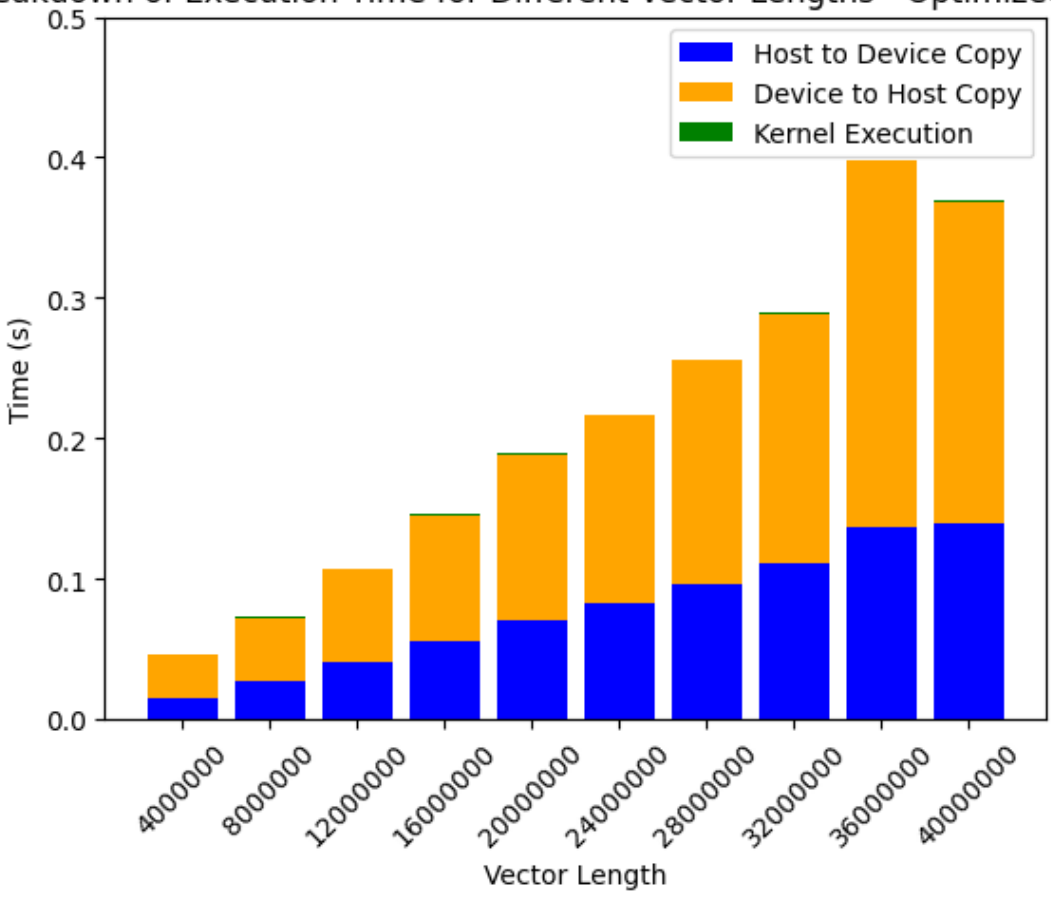
There will be, in total, the same amount of warps with control divergence as in the previous question, PLUS one additional row being added, since the height of the image now also doesn't exactly match with the block height. This additional row will now also be added to the total ($19 - 1$, minus one because the one block in the right corner has already been considered in the 400 warps part), thus the answer here amounts to 418.

Exercise 2

Question 1

Below you can see both execution time breakdowns for different vector lengths: the first for the non-optimized version, with the second being related to the stream-optimized version.


Breakdown of Execution Time for Different Vector Lengths - Non-Optimized Version
Host to Device Copy
Device to Host Copy
Kernel Execution
Time (s)
0.0
0.1
0.2
0.3
0.4
0.5
4000000
8000000
12000000
16000000
20000000
24000000
28000000
32000000
36000000
40000000
Vector Length

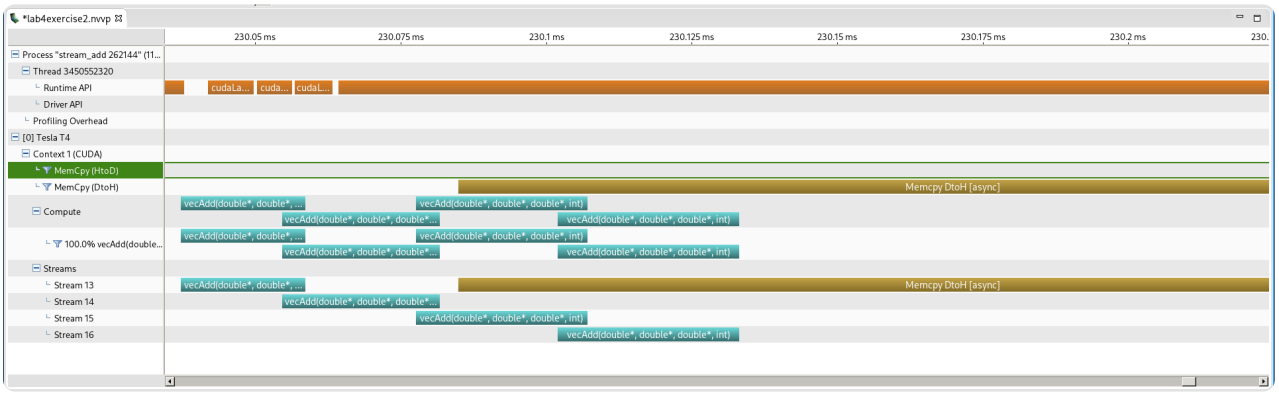

Breakdown of Execution Time for Different Vector Lengths - Optimized Version
Host to Device Copy
Device to Host Copy
Kernel Execution
Time (s)
0.0
0.1
0.2
0.3
0.4
0.5
4000000
8000000
12000000
16000000
20000000
24000000
28000000
32000000
36000000
40000000
Vector Length

Both versions seem to have one outlier per plot (we ran the tests several times – these were not deterministic); however, ignoring those, we can see that time seems to linearly grow with increasing vector lengths.

Question 2

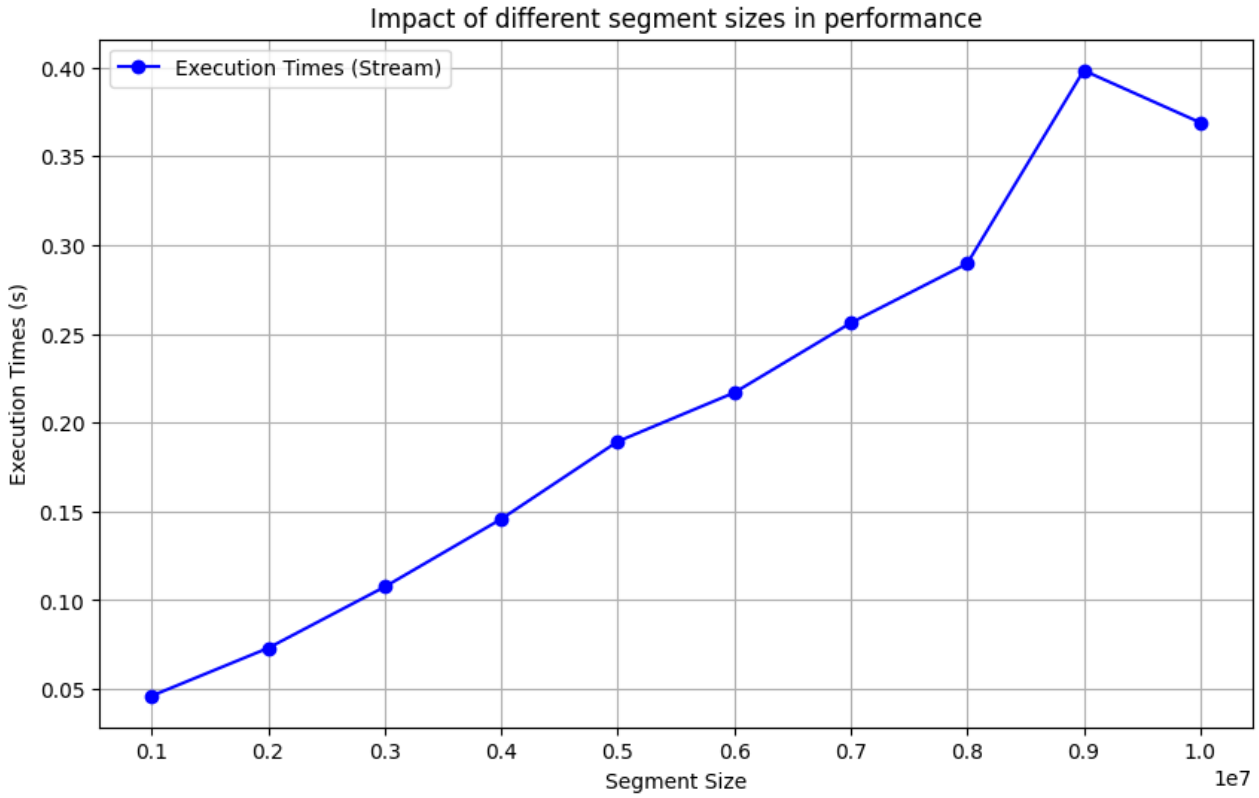
To trace the performance, we used the following command (note that the vector size was set to 262144):

```
nvprof --output-profile lab4exercise2.nvvp -f ./stream_add 262144
```



From the timeline, we observe concurrent execution across multiple streams (Streams 13 to 16), which indicates (partial) overlap of communication and computation.

Question 3

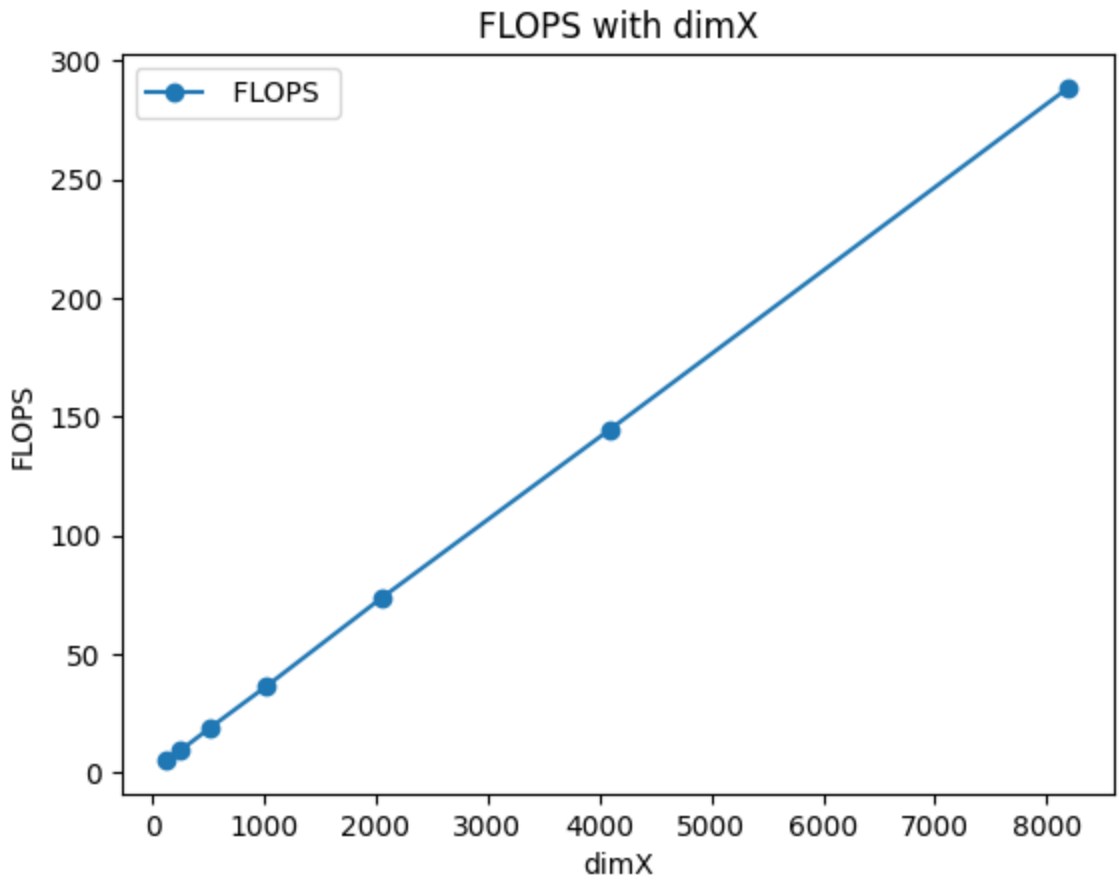


The plot has an outlier (for size 0.9E7). Other than that, the general outcome seems to lead to the fact that segment size has no strong impact on performance – that is, execution time linearly grows with segment size.

Exercise 3

Question 1

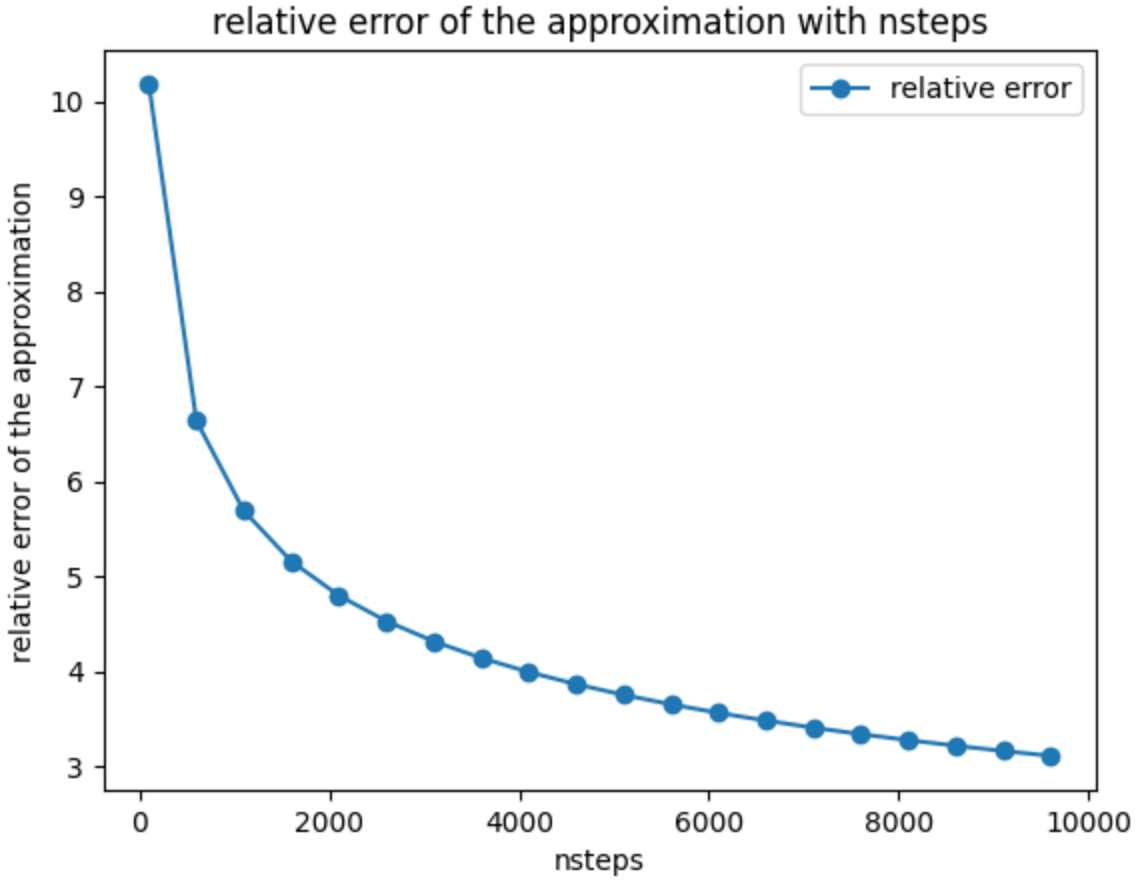
We fixed the step at 100, and continuosly increased the step by doubling it, from 128 to 12800 (not inclusive).



From the plot, we noticed that the amount of floating point operations is basically linearly related to the increase of input data.

Question 2

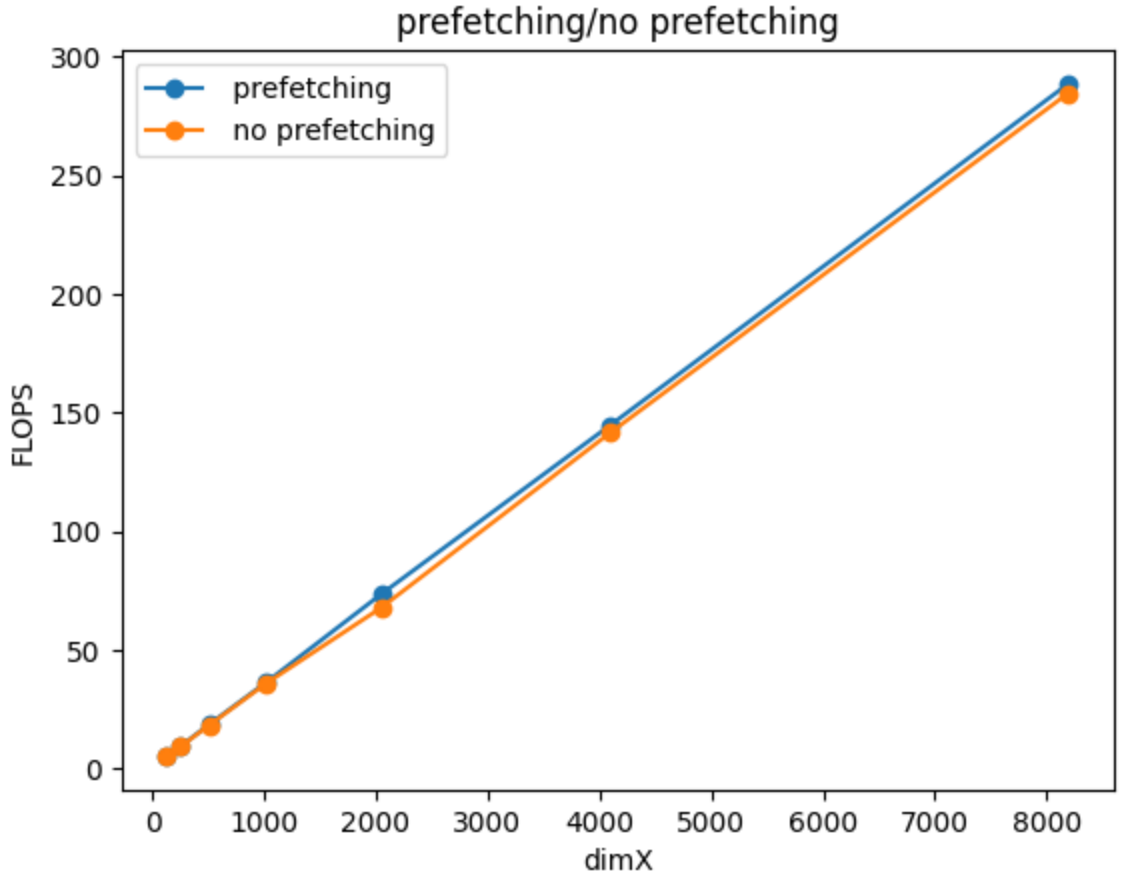
We fixed, as mentioned in the question's statement, dimX at 1024, and the steps range from 100 to 10000, increasing by 500.

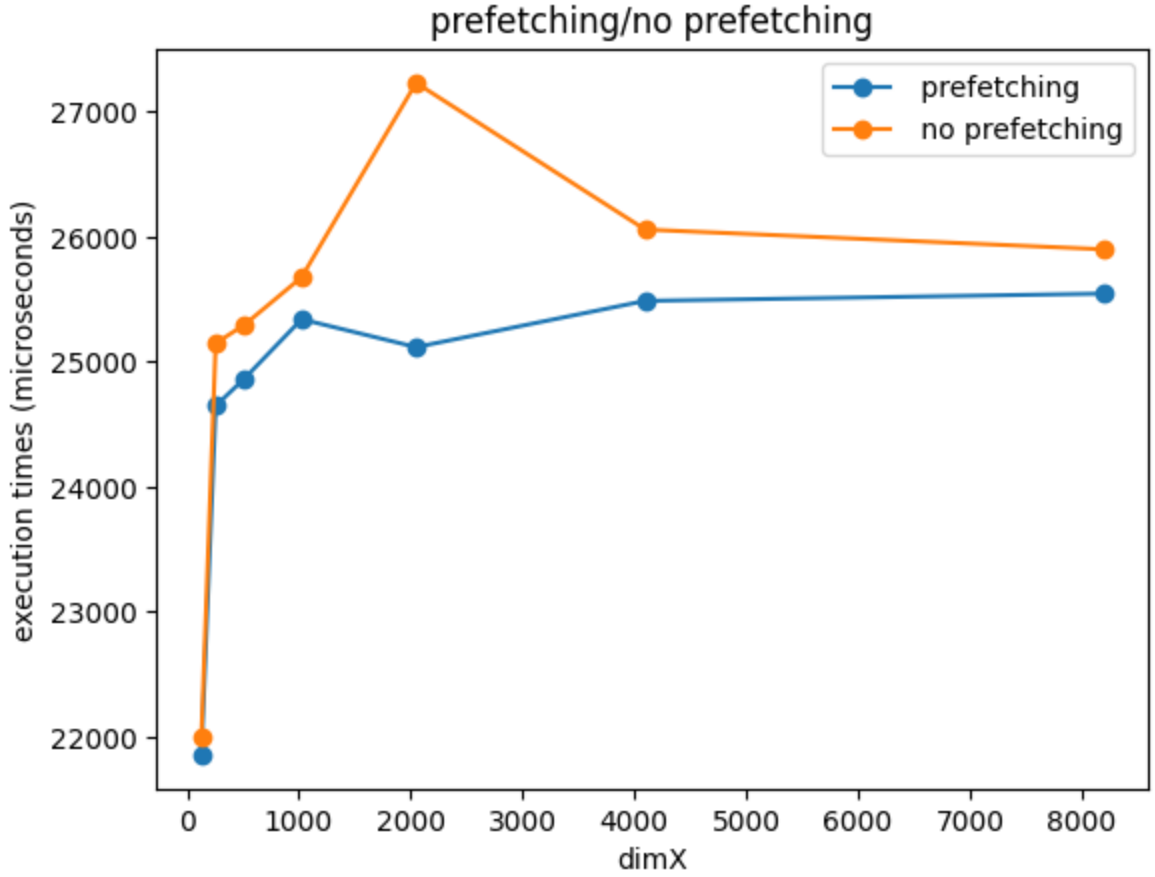


From the figure, we can find that the relative error decreases exponentially as the number of iterations increases.

Question 3

We plotted two plots for this question: one comparing FLOPS, and one comparing execution time, for each configuration (with and without prefetching):


prefetching/no prefetching
prefetching
no prefetching
FLOPS
dimX
0
50
100
150
200
250
300
0
1000
2000
3000
4000
5000
6000
7000
8000


prefetching/no prefetching
prefetching
no prefetching
execution times (microseconds)
dimX
22000
23000
24000
25000
26000
27000
0
1000
2000
3000
4000
5000
6000
7000
8000

As we can see, the amount of FLOPS is kept more-or-less similar with increasing $\mathrm{dim}X$ values; however, the execution time decreases a lot with prefetching, as expected!