

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: $X = 800$, $Y = 600$, Grid = 32×16 blocks

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 $800 \% 32 = 0$, and that $600 \% 16 = 8$, which leads to there being no horizontal divergence, only possibly vertical one – and with vertical divergence of 8. Since warps are processed sequentially and their size is 32, which is the same size of the X-dimension of the block, each warp will contain one line of the block. The last 8 lines do not make part of the grid and thus will be processed all in the same way leading to having 0 warps 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.

The number of total wraps in the grid is $19 \times 50 \times 16 = 15,200$ wraps.

For the control divergence, the remainder in the X -dimension is $600 \% 32 = 4$ and in the Y -dimension is $800 \% 16 = 0$.

Each wrap will contain one line of the block as mentioned in the previous question. However, in the last block of the X -dimension, 4 columns will be outside of the grid, thus only some threads will perform computations.

Therefore, the number of wraps without control divergence is $\lfloor (50 \times 16 = 800) \rfloor$ wraps.

Question 3: $X = 600$, $Y = 799$

In the (X) -axis, we have $\lceil \frac{600}{32} \rceil = 19$ blocks, and $\lceil \frac{799}{16} \rceil = 50$ in the (Y) -axis.

The number of total wraps in the grid is $(19 \times 50 \times 16 = 15,200)$ wraps.

For the control divergence, the remainder in the X -dimension is $(600 \% 32 = 4)$ and in the Y -dimension is $(799 \% 16 = 15)$.

In this case, since there is one less row in the Y -dimension, the last block will contain 15 rows outside of the grid. However, as mentioned previously, each wrap will have one line of the block, thus the last 15 rows will be processed in the same manner. Therefore, there is no divergence control in the Y -dimension.

We will then have $\lfloor (50 \times 16 - 1 = 799) \rfloor$ wraps with control divergence. We subtract 1 because since we removed a line from the program, one wrap disappears.

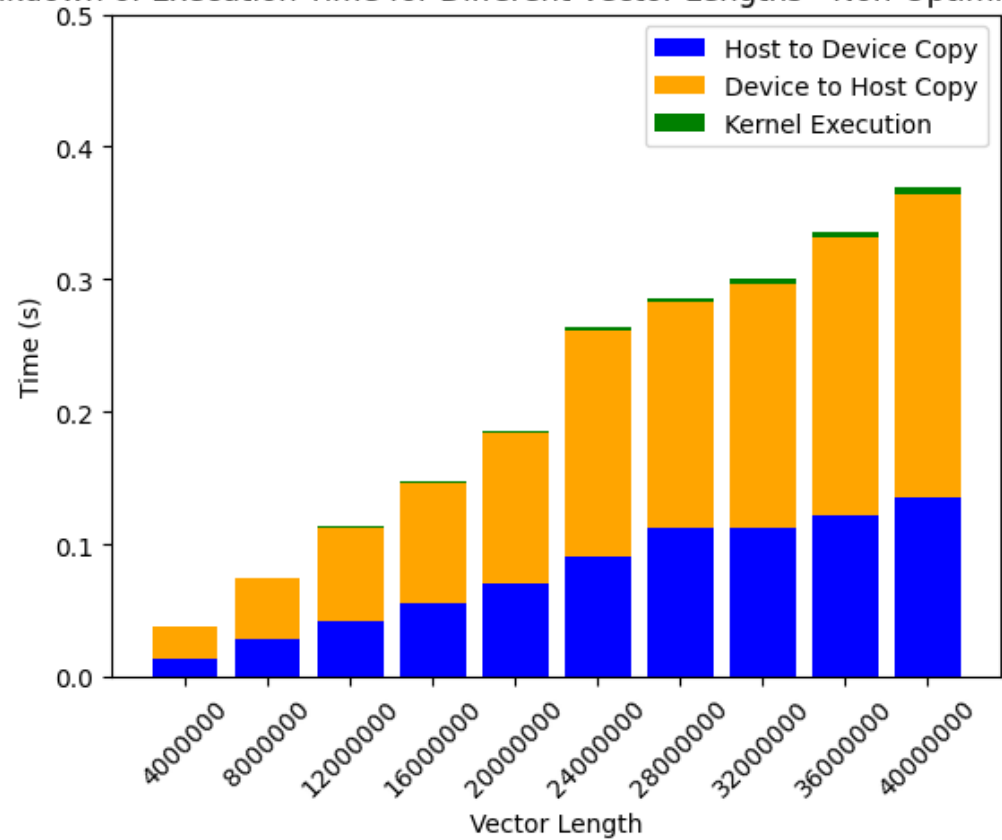
Exercise 2

Question 1

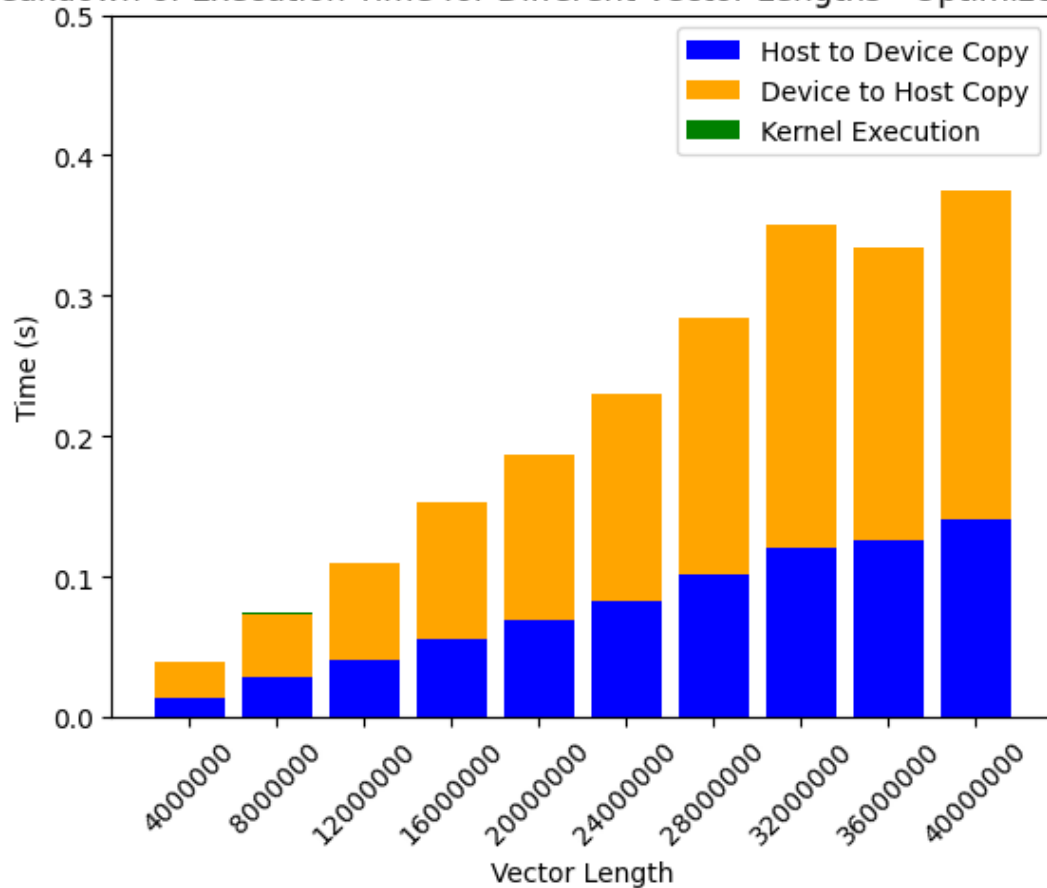
Below you can see both execution time breakdowns for different vector lengths:

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



Breakdown of Execution Time for Different Vector Lengths - Optimized Version

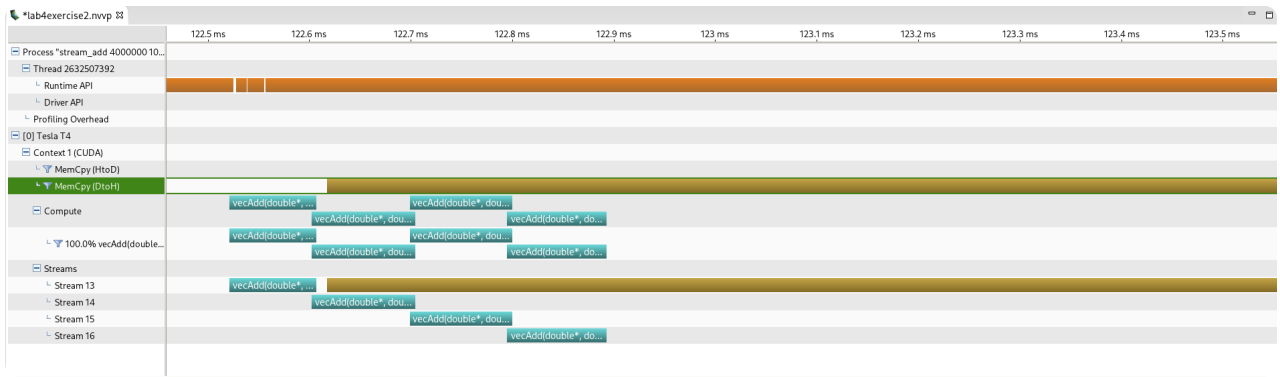


The optimized version seems to have one outlier (we ran the tests several times – these were not deterministic); however, ignoring those, we can see that time seems to linearly grow with increasing vector lengths. Additionally, we also verify that using streams, we reduce the Kernel Execution time to almost zero, however the copying times increase.

Question 2

To trace the performance, we used the following command (note that the vector size was set to 4000000, and the segment size to 1000000 below):

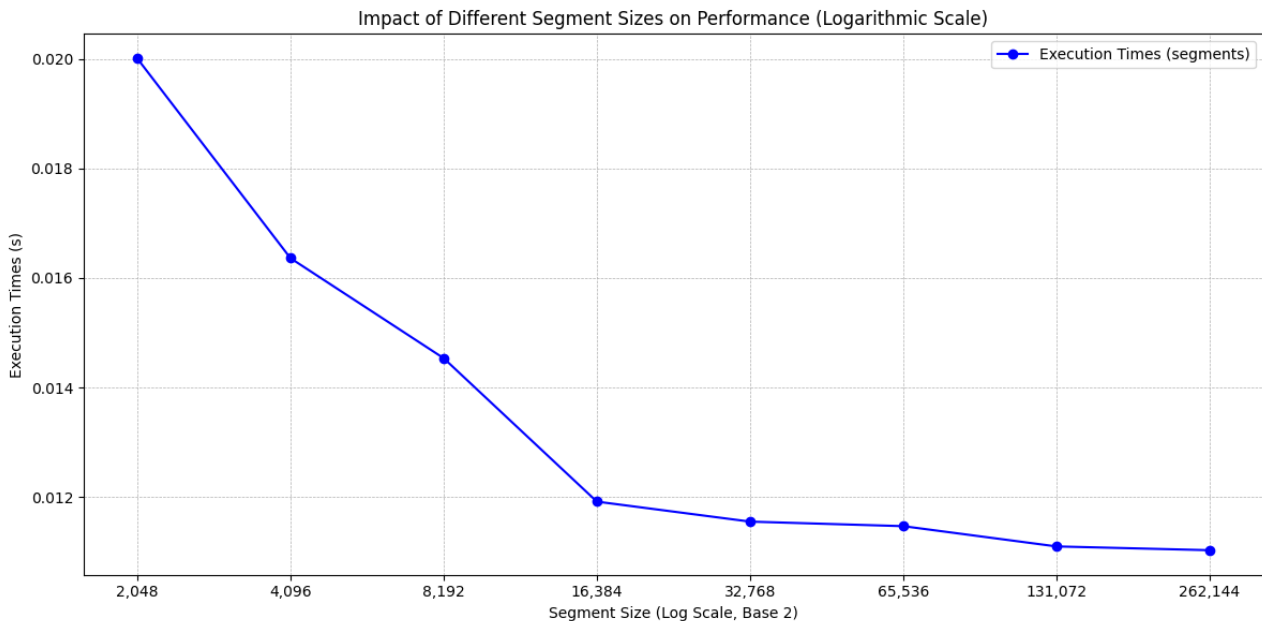
```
nvprof --output-profile lab4exercise2.nvvp -f ./hw4_ex2 4000000 1000000
```



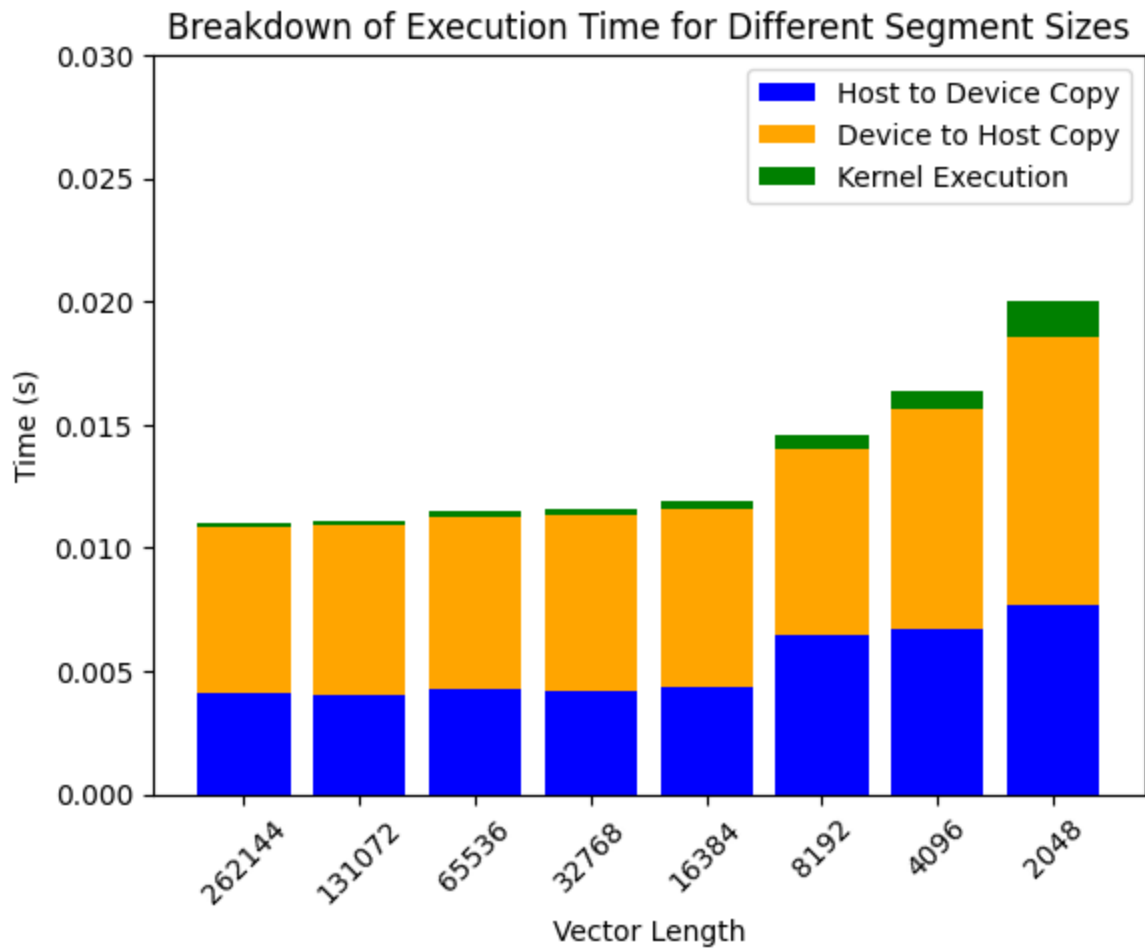
From the timeline, we observe concurrent execution across multiple streams (Streams 13 to 16), which indicates (partial) overlap of communication and computation.

Question 3

We chose to run this experiment with an input size of (2^{20}) which is (1048576) , and segment sizes between (2^{11}) to (2^{18}) . This output leads us to conclude that the segment size impacts on performance with bigger segment sizes leading to lower total execution times.



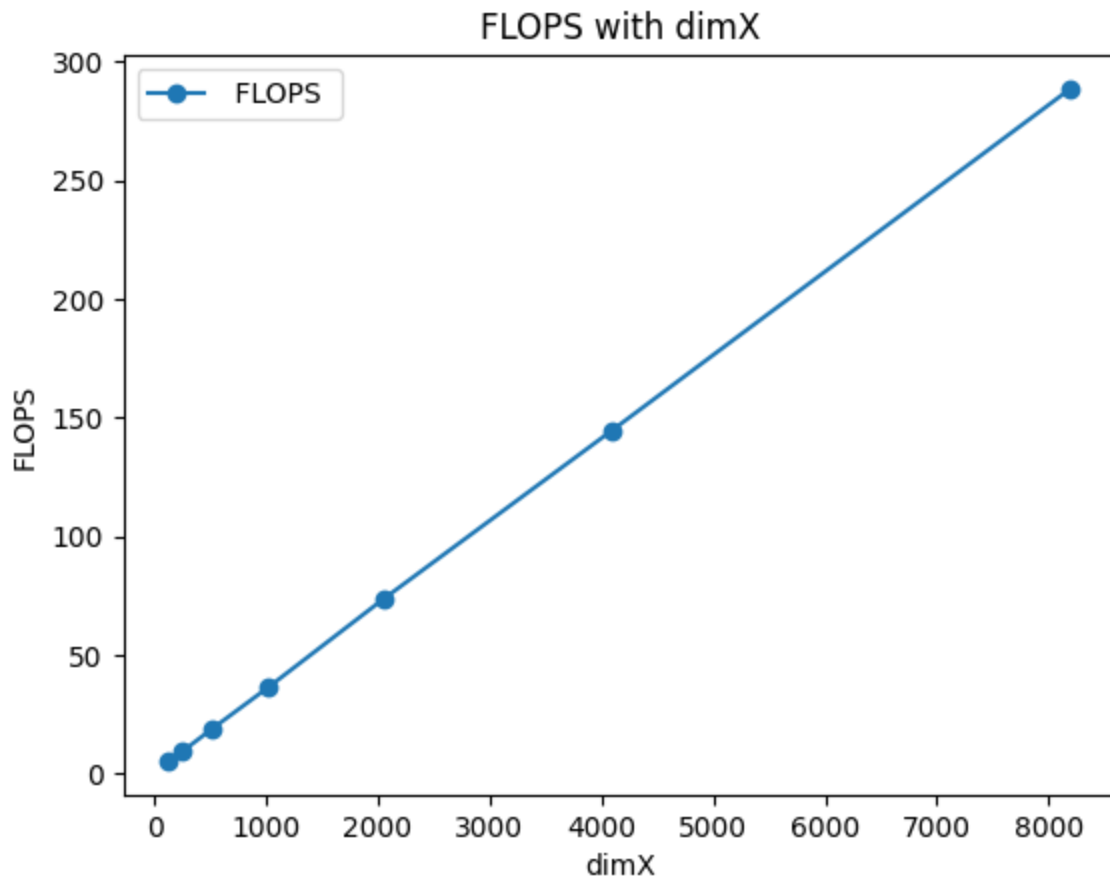
We can also see that smaller segment sizes lead to more time spent in copying from device to host and to bigger kernel execution times, as expected.



Exercise 3

Question 1

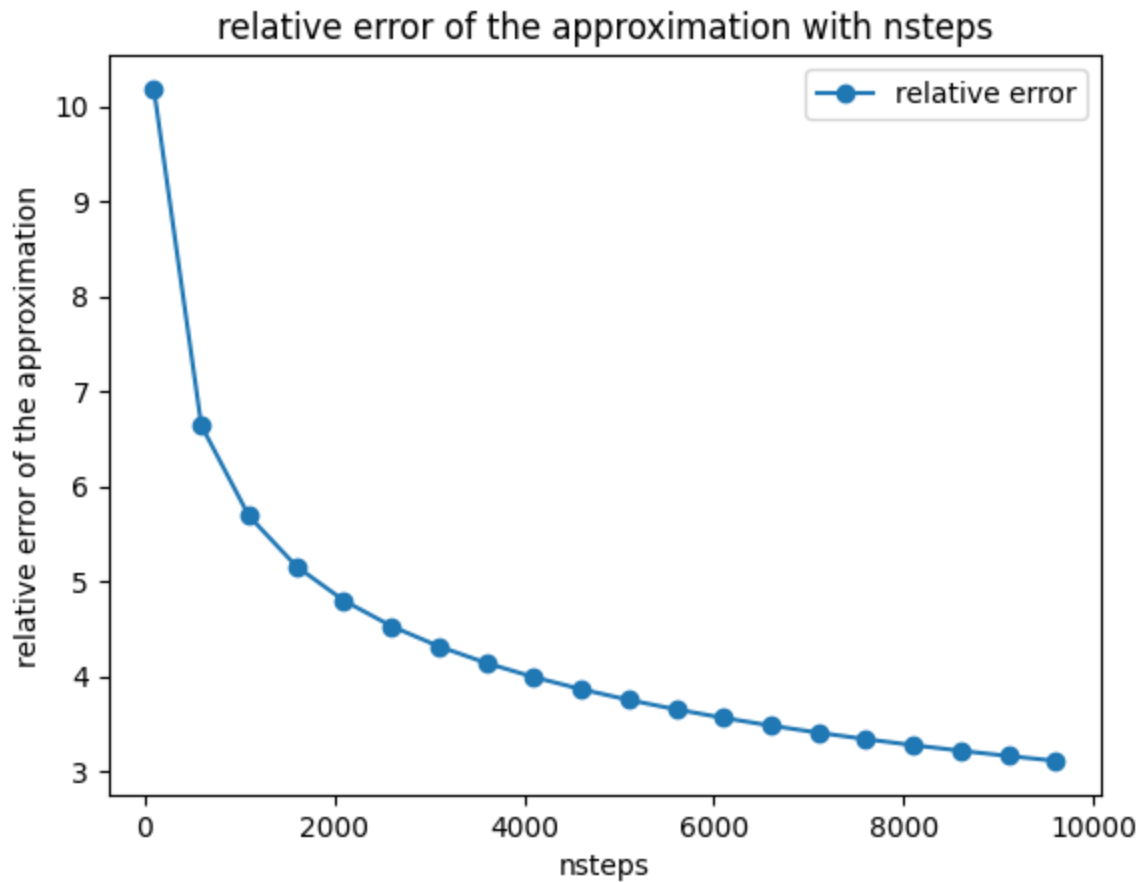
We fixed the step at (100) , and continuously increased the step by doubling it, from (128) to (12800) (not inclusive).



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

Question 2

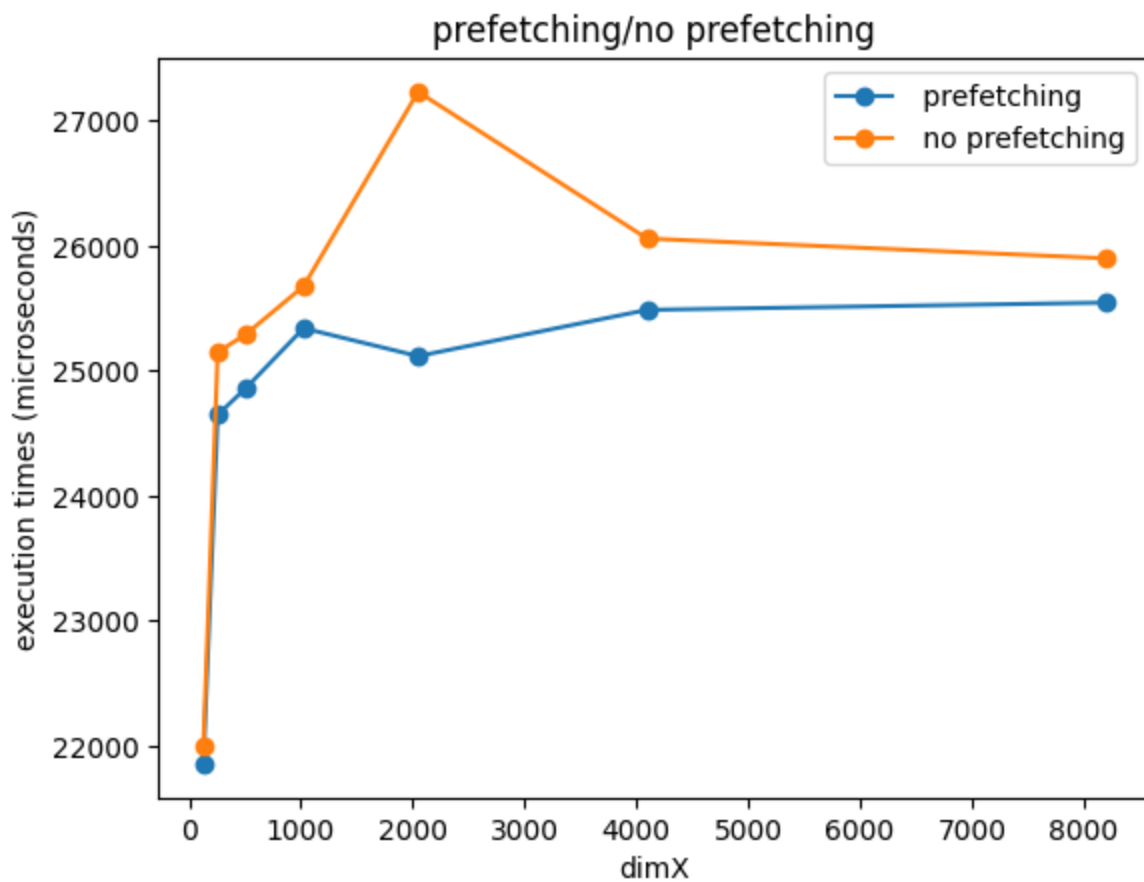
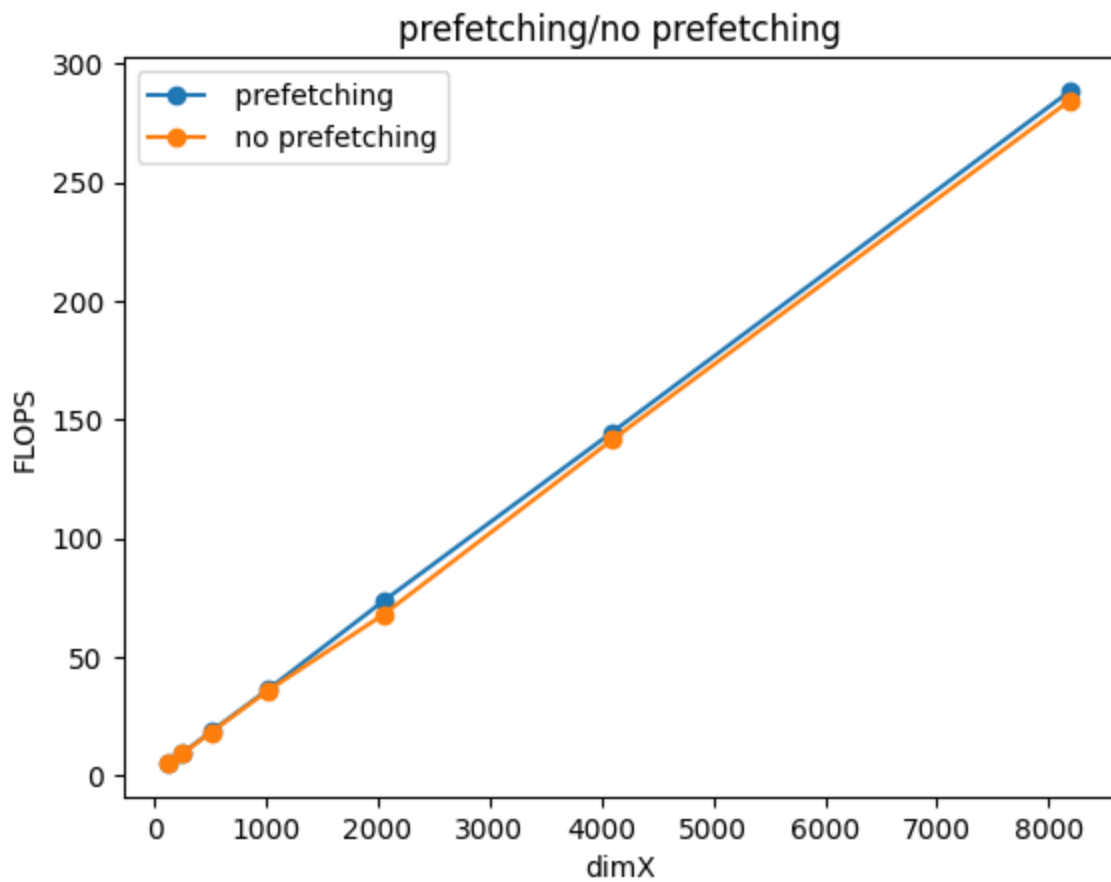
We fixed, as mentioned in the question's statement, dimX at (1024) , and the steps range from (100) to (10000) , increasing by (500) .



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):



As we can see, the amount of FLOPS is kept more-or-less similar with increasing dimX values; however, the execution time decreases a lot with prefetching, as expected!