Exercise 1 - Thread Scheduling and Execution Efficiency

1. Assume X=800 and Y=600. Assume that we decided to use a grid of 16X16 blocks. That is, each block i s organized as a 2D 16X16 array of threads. How many warps will be generated during the execution of the kernel? How many warps will have control divergence? Please explain your answers.

For each block which has 16x16=256 threads, it has 8 warps (256/32=8).
In X direction (i.e. row), we have 800/16=50 blocks, and there are no control divergence in this dir ection. In Y direction (i.e. column), we have (600/16)=37.5, since we can't have a portion block, w e are going to have 38 blocks in this direction. Although half of the warps in the last block are out of range, but these warps are all executed in the same statement, it won't have control divergence a s well.
So the total warps in this execution is 50x38x8=15200, and there is 0 warp will have control diverg ence.

2. Now assume X=600 and Y=800 instead, how many warps will have control divergence? Please explain y our answers.

In X direction (i.e. row), we have 600/16=37.5 blocks, and we are going to have 38 blocks in this d irection. Since the warp is assigned by the sequence of thread block index and thread block will fir st linearize into 1 dimension, so we will have control divergence in the right edge of the picture (i. e. the last column of the picture).
In Y direction (i.e. column), we have (800/16)=50, so there are 50x8=400 warps will have control divergence.

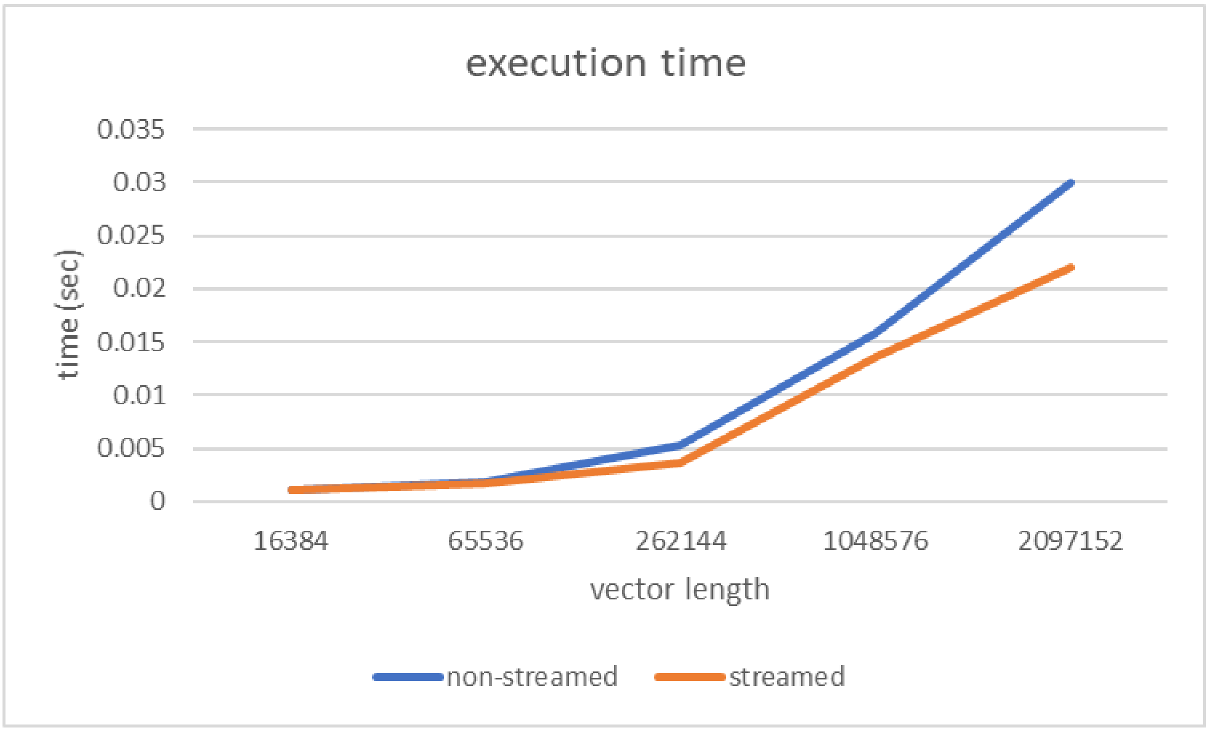
3. Now assume X=600 and Y=799, how many warps will have control divergence? Please explain your ans wers.

Followed by question 2, the last column of the picture will have control divergence.
And for the Y direction, the (799/16)=49.93, so we will have control divergence in the last row as well. More specifically, the last warp in the block will have control divergence. Therefore, there ar e 50x8 (along the right edge of picture) + 37x1 (along the bottom edge of picture) = 437

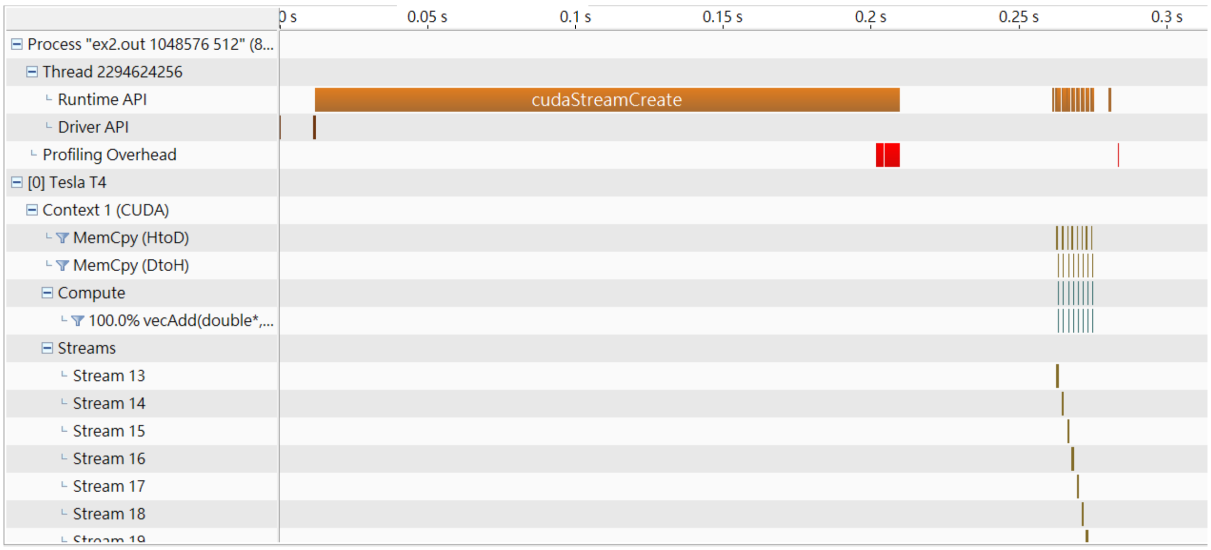
Exercise 2 - CUDA Streams

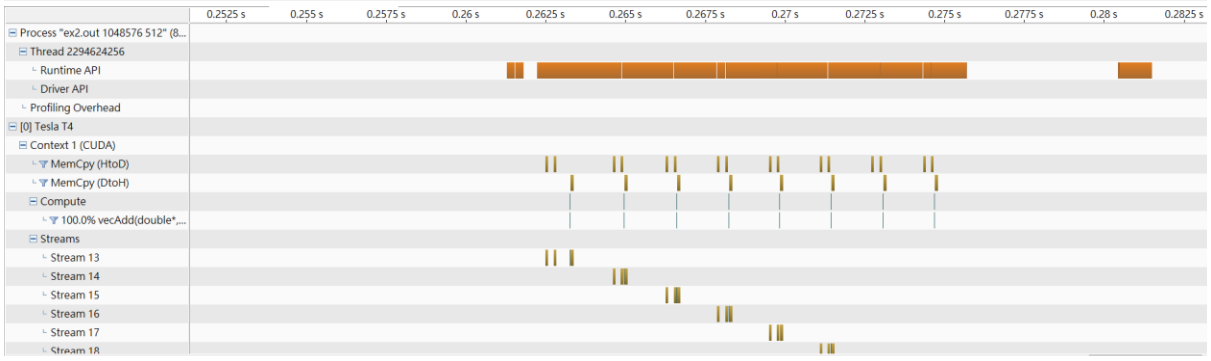
1. Compared to the non-streamed vector addition, what performance gain do you get? Present in a plot (yo u may include comparison at different vector length)

We try five different vector lengths. And we can observe that the larger the vector length, we can s ave more execution time compared to the non-streamed version.



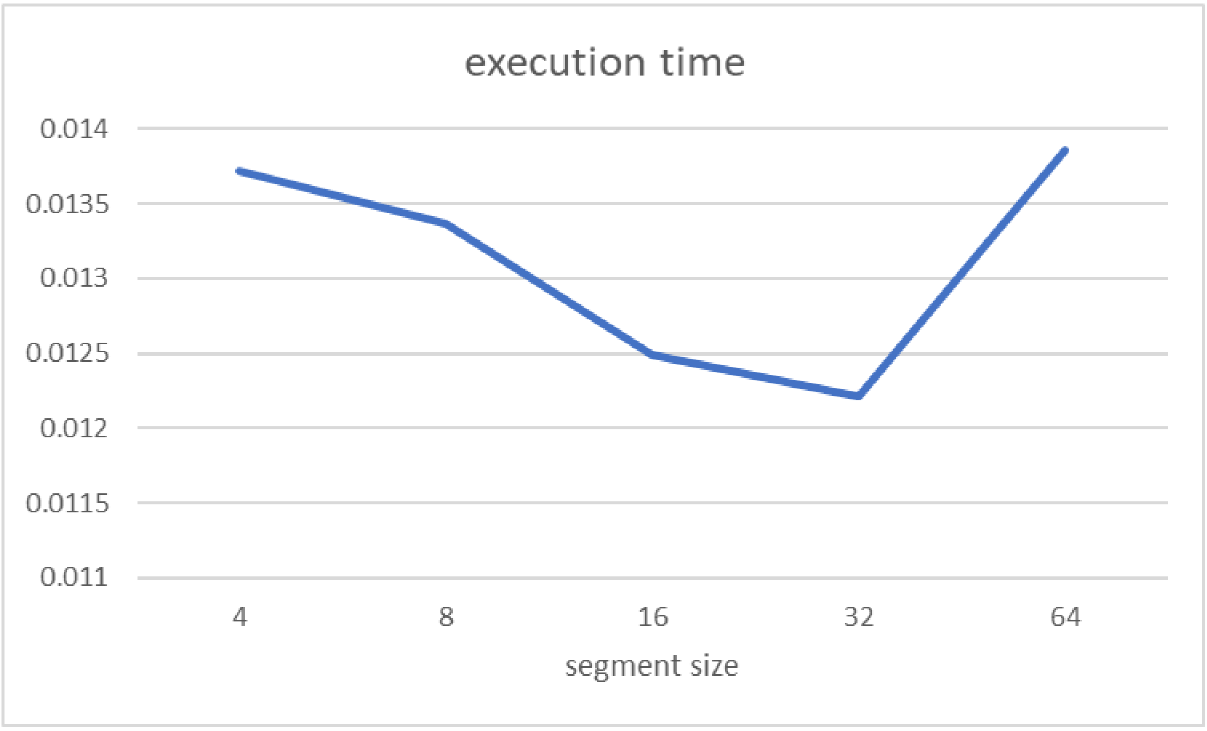
2. Use nvprof to collect traces and the NVIDIA Visual Profiler (nvvp) to visualize the overlap of communication and computation.





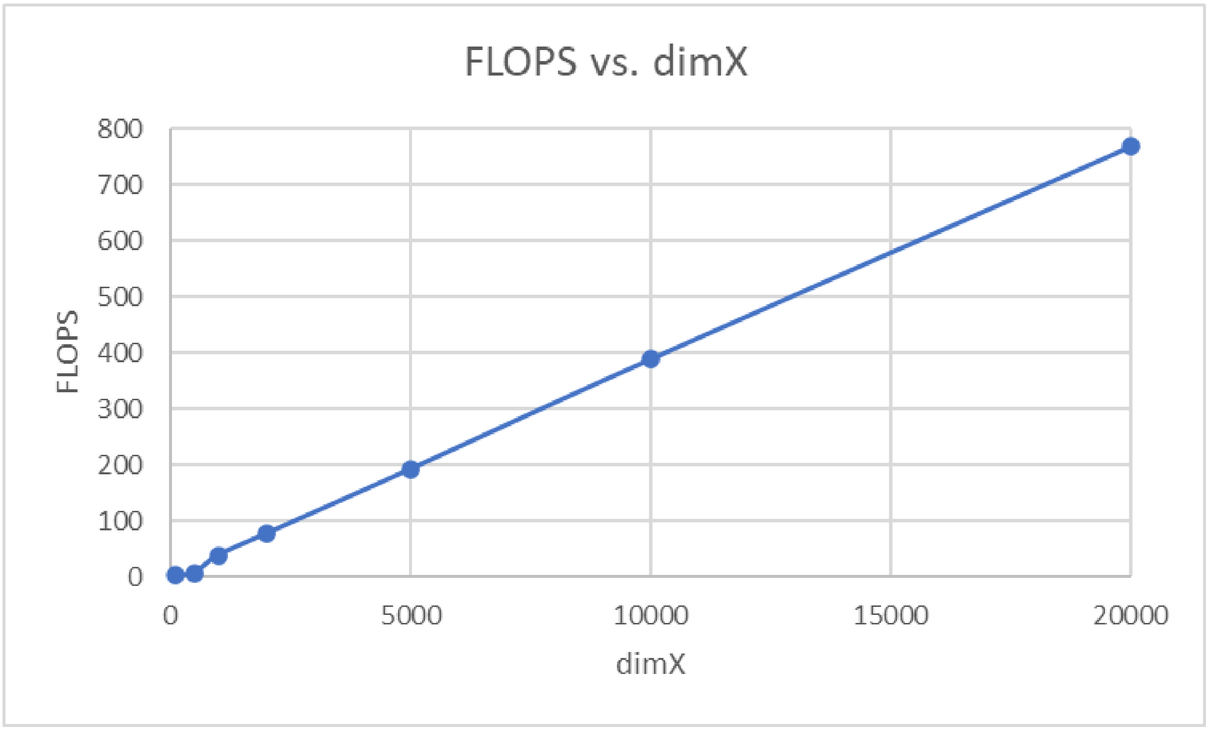
3. What is the impact of segment size on performance? Present in a plot (you may choose a large vector an d compare 4-8 different segment sizes)

We choose 1048576 as the fixed vector length and compare the execution time with 5 different seg ment sizes. Since the larger the segment size, it is more possible for it to parallel execution, so that it can take less time to finish the job. However, it still has upperbound, if there is not enough hard ware to support the execution of different streams, then it probably spends more time on waiting fo r the resource. That’ s why it takes more time when the segment size is larger than 32.



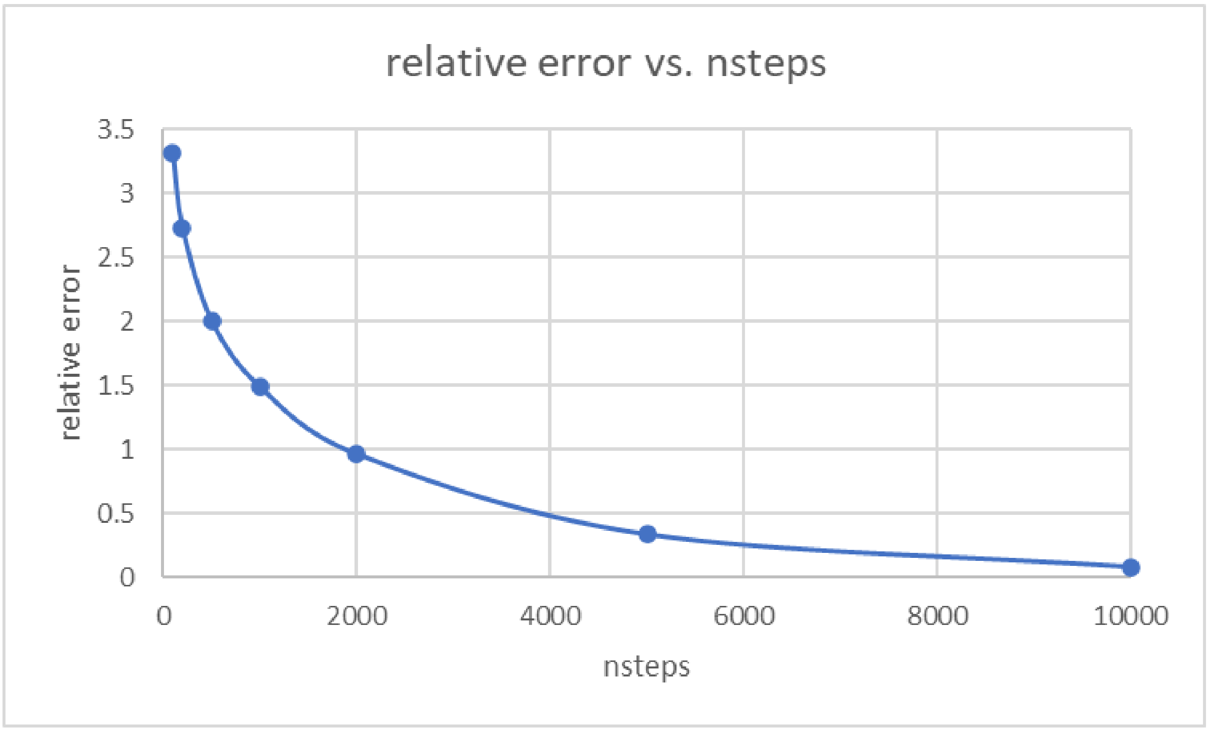
1. Run the program with different dimX values. For each one, approximate the FLOPS (floating-point operation per second) achieved in computing the SMPV (sparse matrix multiplication). Report FLOPS at different input sizes in a FLOPS. What do you see compared to the peak throughput you report in Lab2?

FLOPS increases while input sizes increase, but it did not reach the theoretical peak throughput due to various limitations.



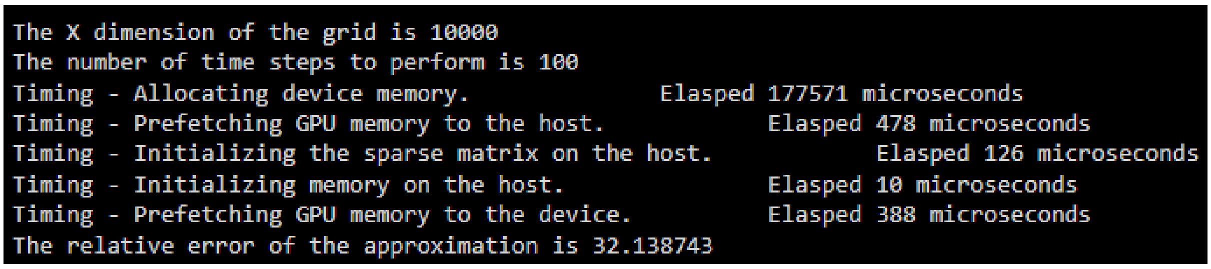
2. Run the program with dimX=128 and vary nsteps from 100 to 10000. Plot the relative error of the approximation at different nstep. What do you observe?

As expected, the relative error decreases as nsteps increases.



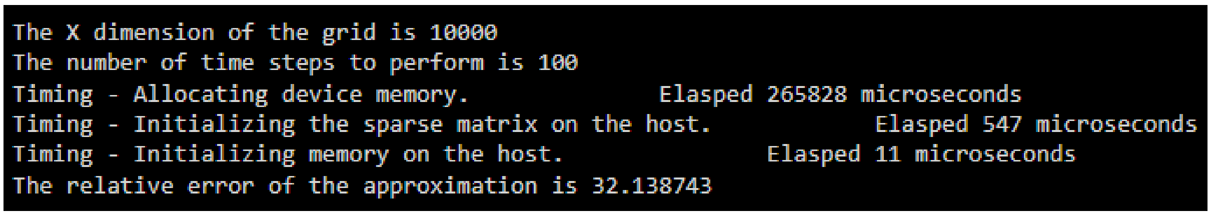
3. Compare the performance with and without the prefetching in Unified Memory. How is the performance impact? [Optional: using nvprof to get metrics on UM]

Output with prefetching:



```
The X dimension of the grid is 10000
The number of time steps to perform is 100
Timing - Allocating device memory.              Elasped 177571 microseconds
Timing - Prefetching GPU memory to the host.            Elasped 478 microseconds
Timing - Initializing the sparse matrix on the host.            Elasped 126 microseconds
Timing - Initializing memory on the host.               Elasped 10 microseconds
Timing - Prefetching GPU memory to the device.          Elasped 388 microseconds
The relative error of the approximation is 32.138743
```

Output without prefetching:



```
The X dimension of the grid is 10000
The number of time steps to perform is 100
Timing - Allocating device memory.              Elasped 265828 microseconds
Timing - Initializing the sparse matrix on the host.            Elasped 547 microseconds
Timing - Initializing memory on the host.               Elasped 11 microseconds
The relative error of the approximation is 32.138743
```

We can see that the program with prefetching outperforms the program without prefetching.

links: https://github.com/LaiYuHong/DD2360HT23/

contribution: we collaborated to finish both the implementation of code and answering questions.