Contribution: Yitong Li mainly finish the exercise 1 and 2 and Guoqing Liang mainly finish the exercise 2 and 3.
Code Link: CHU-2002/DD2360HT23 (github.com)

Exercise 1

1. Assume X=800 and Y=600. Assume that we decided to use a grid of 16X16 blocks. That is, each block is 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.

Because the grid size is 16X16 = 256, for X=800 and Y=600, considering the kernel function, the block number should be $\lceil 800/16 \rceil * \lceil 600/16 \rceil = 1900$. So the number of threads should be 1900X16X16 = 486400.
So the number of warps is 486400/32 = 15200.
Assuming the GPU use row-major order, for the blocks y from 601 to 608, ***Divergences*** occur. So the number is 4*800/16 = 200. (In these blocks, only half of warps get ***Divergences***)

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

Just as the first question. X from 601 to 608 ***Divergences*** occur.
Still assuming the GPU use row-major order, the number is 800/16 * 256/32 = 400. (In these blocks, all blocks get ***Divergences***)

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

For X = 600 and Y = 799, ***Divergences*** occur in threads get X from 601 to 608, Y in 800.
Still assuming the GPU use row-major order, for warps, the last row and blocks in question 2 will all lead to ***Divergences.*** So the number will be 800/16 * 256/32 + $\lceil 600/16 \rceil$ -1= 438.

Exercise 2 - CUDA Streams

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

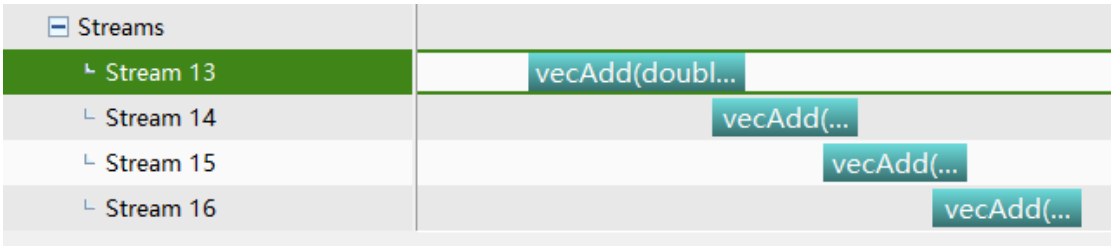
Non-streamed vector addition

```
==36602== NVPROF is profiling process 36602, command: /content/drive/MyDrive/Assignment/Assignment4/ex2/vectorAdd1 10000
Total time with streams: 0.000897 seconds
Results correct!
==36602== Profiling application: /content/drive/MyDrive/Assignment/Assignment4/ex2/vectorAdd1 10000
==36602== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   59.36%  18.464us         2  9.2320us  9.0560us  9.4080us  [CUDA memcpy HtoD]
                   26.75%  8.3200us         1  8.3200us  8.3200us  8.3200us  [CUDA memcpy DtoH]
                   13.89%  4.3200us         1  4.3200us  4.3200us  4.3200us  vecAdd(double*, double*, double*, int)
      API calls:   99.56%  234.07ms         3  78.023ms  4.6950us  234.05ms  cudaMalloc
                    0.18%  425.95us         1  425.95us  425.95us  425.95us  cudaLaunchKernel
                    0.09%  221.51us       114  1.9430us     197ns  79.078us  cuDeviceGetAttribute
                    0.09%  202.36us         3  67.454us  31.265us  96.879us  cudaMemcpy
                    0.07%  157.14us         3  52.380us  6.3730us  137.08us  cudaFree
                    0.01%  12.123us         1  12.123us  12.123us  12.123us  cuDeviceGetName
                    0.00%  7.9670us         1  7.9670us  7.9670us  7.9670us  cuDeviceGetPCIBusId
                    0.00%  6.6920us         1  6.6920us  6.6920us  6.6920us  cudaDeviceSynchronize
                    0.00%  5.9520us         1  5.9520us  5.9520us  5.9520us  cuDeviceTotalMem
                    0.00%  2.1270us         3     709ns     336ns  1.4090us  cuDeviceGetCount
                    0.00%  1.0950us         2     547ns     281ns     814ns  cuDeviceGet
                    0.00%     520ns         1     520ns     520ns     520ns  cuModuleGetLoadingMode
                    0.00%     390ns         1     390ns     390ns     390ns  cuDeviceGetUuid
```

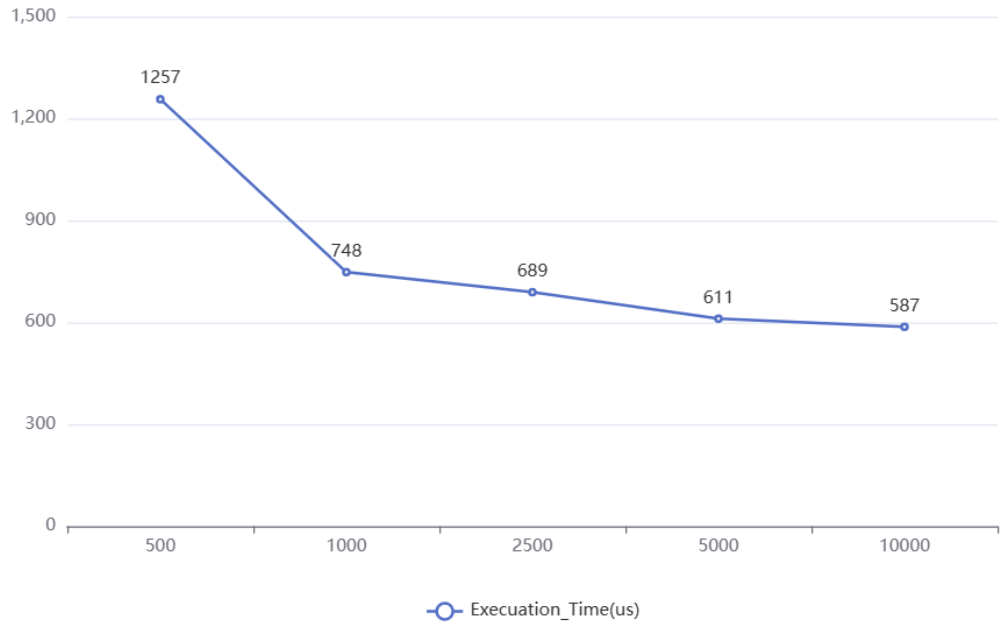
Streamed vector addition

```
==36654== NVPROF is profiling process 36654, command: /content/drive/MyDrive/Assignment/Assignment4/ex2/vectorAdd2 10000
Total time with streams: 0.000688 seconds
Results correct!
==36654== Profiling application: /content/drive/MyDrive/Assignment/Assignment4/ex2/vectorAdd2 10000
==36654== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   74.16%  157.43us        12  13.119us  6.0890us  28.533us  [CUDA memcpy HtoH]
                   25.84%  54.847us         4  13.711us  10.144us  16.992us  vecAdd(double*, double*, double*, int)
      API calls:   99.48%  173.18ms         3  57.727ms  4.0360us  173.16ms  cudaHostAlloc
                    0.27%  478.54us         4  119.64us  5.6200us  460.17us  cudaLaunchKernel
                    0.11%  191.22us        12  15.935us  7.8240us  33.498us  cudaMemcpyAsync
                    0.08%  133.43us       114  1.1700us     139ns  53.677us  cuDeviceGetAttribute
                    0.02%  41.406us         4  10.351us  2.3770us  33.902us  cudaStreamCreate
                    0.01%  20.945us         4  5.2360us  3.0380us  11.286us  cudaStreamDestroy
                    0.01%  11.881us         1  11.881us  11.881us  11.881us  cuDeviceGetName
                    0.00%  5.9790us         1  5.9790us  5.9790us  5.9790us  cuDeviceGetPCIBusId
                    0.00%  5.6220us         1  5.6220us  5.6220us  5.6220us  cudaDeviceSynchronize
                    0.00%  4.4390us         1  4.4390us  4.4390us  4.4390us  cuDeviceTotalMem
                    0.00%  4.2990us         3  1.4330us     712ns  2.8090us  cudaFree
                    0.00%  2.1690us         3     723ns     242ns  1.6610us  cuDeviceGetCount
                    0.00%  1.1800us         2     590ns     170ns  1.0100us  cuDeviceGet
                    0.00%     668ns         1     668ns     668ns     668ns  cuModuleGetLoadingMode
                    0.00%     216ns         1     216ns     216ns     216ns  cuDeviceGetUuid
```

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 and compare 4-8 different segment sizes)



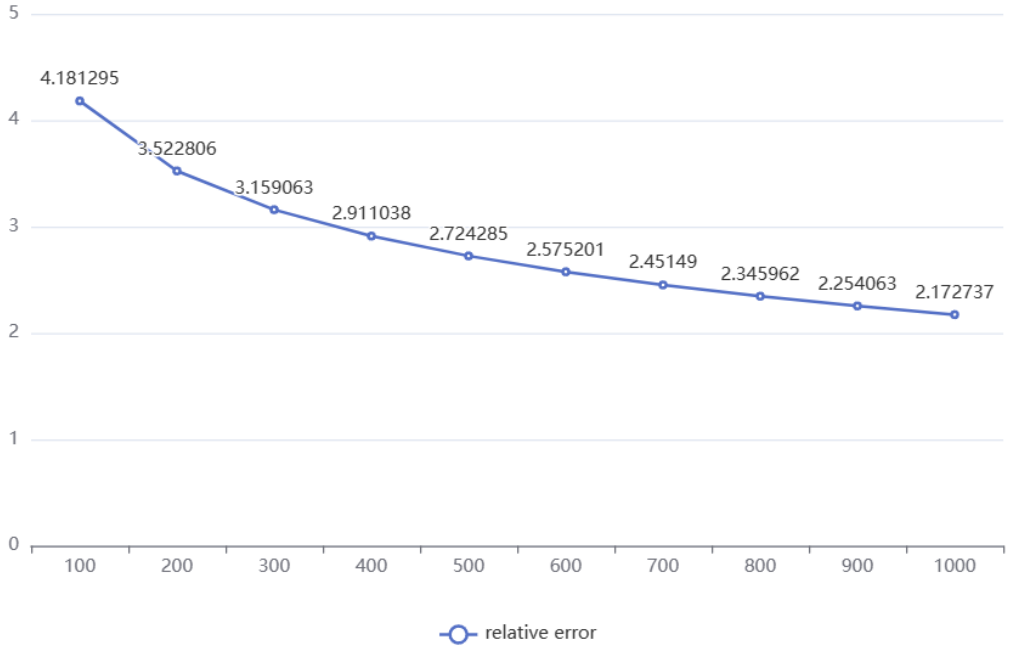
As segment Size increase, the performance of Streamed vector addition is getting better.

Exercise 3 - Heat Equation with using NVIDIA libraries

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?

As dimX increases, the size of the matrix and the number of non-zero elements grow, which cases more GPU cores can be used simultaneously and increases FLOPS

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 nsteps increases, the relative error decreases, indicating a more accurate approximation with a higher number of steps.

3.Compare the performance with and without the prefetching in Unified Memory. How is the performance impact?

```
The X dimension of the grid is 128
The number of time steps to perform is 1000
Timing - Total Execuation TIme with Prefetch.          Elasped 105452 microseconds
The X dimension of the grid is 128
The number of time steps to perform is 1000
Timing - Total Execuation TIme without Prefetch.              Elasped 124704 microseconds
```

Prefetching in Unified Memory can enhance performance when dimx is not large. However, when dimx become larger, the performance will be reduced.This maybe because prefetching can affect the efficiency of GPU memory and leads to an imbalance in memory bandwidth utilization or a decrease in cache hit rate