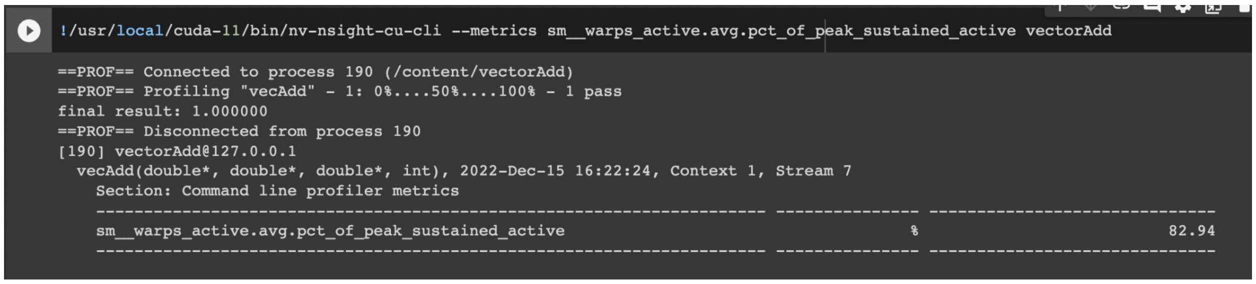
Roman Maksymiuk
27/12/2022

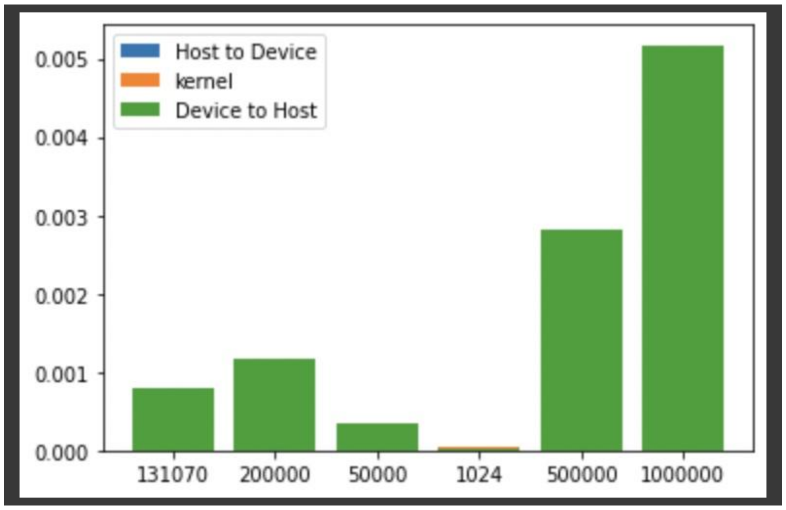
Assignment 3

Exercise 1:

1. To compile I used nvcc vectorAdd.cu -o vectorAdd, and then once I got the executable file I simply called it such as ./vectorAdd
2. For a vector of length N, the vectorAdd kernel performs N floating point operations, one for each element of the input vectors. Also 2*N global memory reads, 1 per each element a and b.
3. For the vector of length 1024, blockSize = 1024, gridSize = 1 since there's only one block of threads needed to process all the elements. After running the correct metrics to find the achieved occupancy I got the following result. Achieved occupancy is a measure of how effectively the GPU is being used by the kernel, we got 82.94%, which is pretty effective



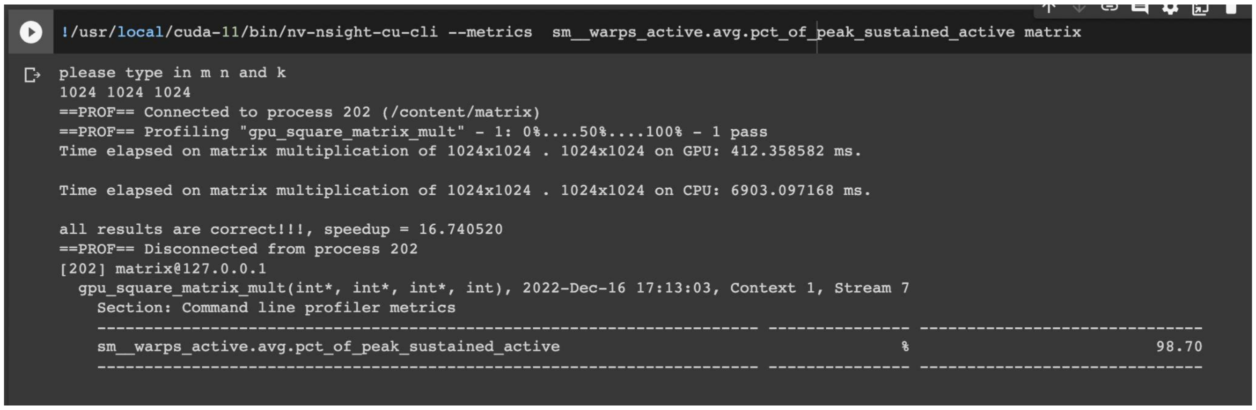
```
!/usr/local/cuda-11/bin/nv-nsight-cu-cli --metrics sm__warps_active.avg.pct_of_peak_sustained_active vectorAdd

==PROF== Connected to process 190 (/content/vectorAdd)
==PROF== Profiling "vecAdd" - 1: 0%....50%....100% - 1 pass
final result: 1.000000
==PROF== Disconnected from process 190
[190] vectorAdd@127.0.0.1
  vecAdd(double*, double*, double*, int), 2022-Dec-15 16:22:24, Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------------- --------------- ------------------------------
    sm__warps_active.avg.pct_of_peak_sustained_active                                    %                          82.94
    ---------------------------------------------------------------------- --------------- ------------------------------
```

4. Once increased to the size of 131070 the program still worked but the gridSize will be 131 since 131 will be needed to process 131070 elements. As well after profiling the program with achieved occupancy went down as well to 81.89% since the GPU will be less fully utilized.
5. 

As it’s clearly seen that the most time is taken by the time it takes to send the data back from the device to the host. Unfortunately the graph is a little bit hard to understand as I chose huge ranges for the vector size. But the general idea is understood.

Exercise 2:

1. Some application domains are: image processing, machine learning, scientific simulations.
2. There are 2n-1 floating point operations performed, as it depends on
3. The kernel reads n elements from matrix A and k elements from matrix B so in total it would be n+k elements
4. Grid: ((128 + BLOCK_SIZE - 1) / BLOCK_SIZE, (128 + BLOCK_SIZE - 1) / (BLOCK_SIZE) and block size would be (BLOCK_SIZE,BLOCK_SIZE). Size of 16, this would be a grid of dimensions (8,8) and block dimensions(16,16) for a total of 8*8*16*16 = 2048 threads.

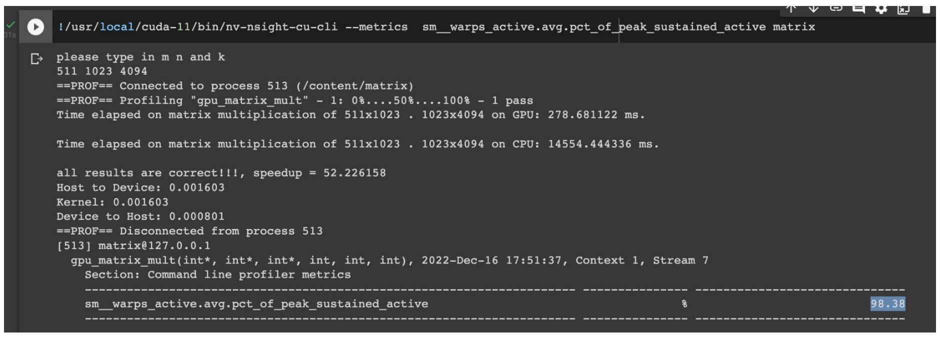


```
!/usr/local/cuda-11/bin/nv-nsight-cu-cli --metrics  sm__warps_active.avg.pct_of_peak_sustained_active matrix
```

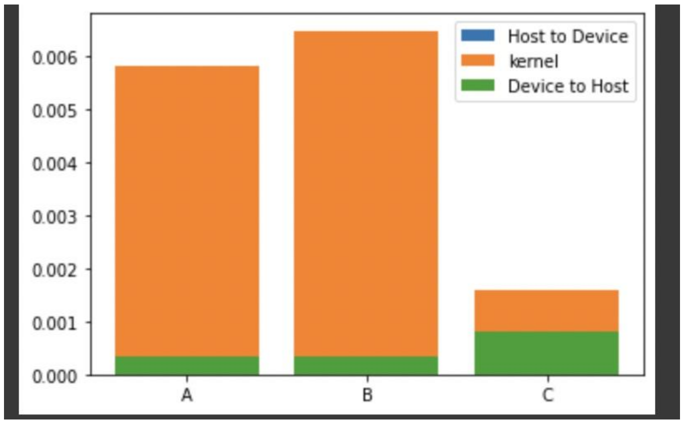
```
please type in m n and k
1024 1024 1024
==PROF== Connected to process 202 (/content/matrix)
==PROF== Profiling "gpu_square_matrix_mult" - 1: 0%....50%....100% - 1 pass
Time elapsed on matrix multiplication of 1024x1024 . 1024x1024 on GPU: 412.358582 ms.

Time elapsed on matrix multiplication of 1024x1024 . 1024x1024 on CPU: 6903.097168 ms.

all results are correct!!!, speedup = 16.740520
==PROF== Disconnected from process 202
[202] matrix@127.0.0.1
  gpu_square_matrix_mult(int*, int*, int*, int), 2022-Dec-16 17:13:03, Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------------- --------------- ------------------------------
    sm__warps_active.avg.pct_of_peak_sustained_active                                    %                          98.70
    ---------------------------------------------------------------------- --------------- ------------------------------
```

5. For this specific size:
   In the code, the grid and block dimensions are set as follows:

dim3 dimGrid((k + BLOCK_SIZE - 1) / BLOCK_SIZE, (m + BLOCK_SIZE - 1) / BLOCK_SIZE);
dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE)



```
!/usr/local/cuda-11/bin/nv-nsight-cu-cli --metrics  sm__warps_active.avg.pct_of_peak_sustained_active matrix

please type in m n and k
511 1023 4094
==PROF== Connected to process 513 (/content/matrix)
==PROF== Profiling "gpu_matrix_mult" - 1: 0%....50%....100% - 1 pass
Time elapsed on matrix multiplication of 511x1023 . 1023x4094 on GPU: 278.681122 ms.

Time elapsed on matrix multiplication of 511x1023 . 1023x4094 on CPU: 14554.444336 ms.

all results are correct!!!, speedup = 52.226158
Host to Device: 0.001603
Kernel: 0.001603
Device to Host: 0.000801
==PROF== Disconnected from process 513
[513] matrix@127.0.0.1
  gpu_matrix_mult(int*, int*, int*, int, int, int), 2022-Dec-16 17:51:37, Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------------- --------------- ------------------------------
    sm__warps_active.avg.pct_of_peak_sustained_active                                    %                          98.38
    ---------------------------------------------------------------------- --------------- ------------------------------
```

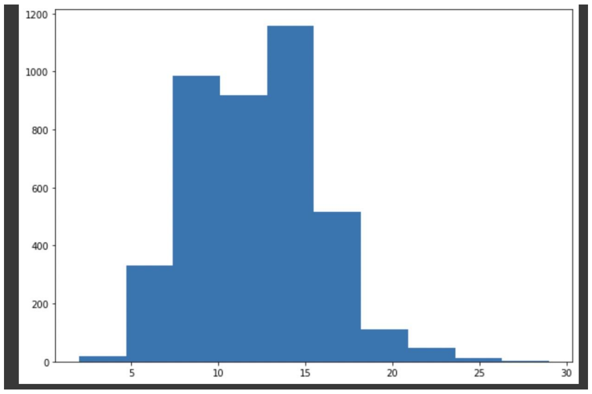
6. 

It seems likethe kernel taking most of the time during the run of the program.
7. The code seems to work the same, as when tried to replace the from double to float the time breakdown is the same.

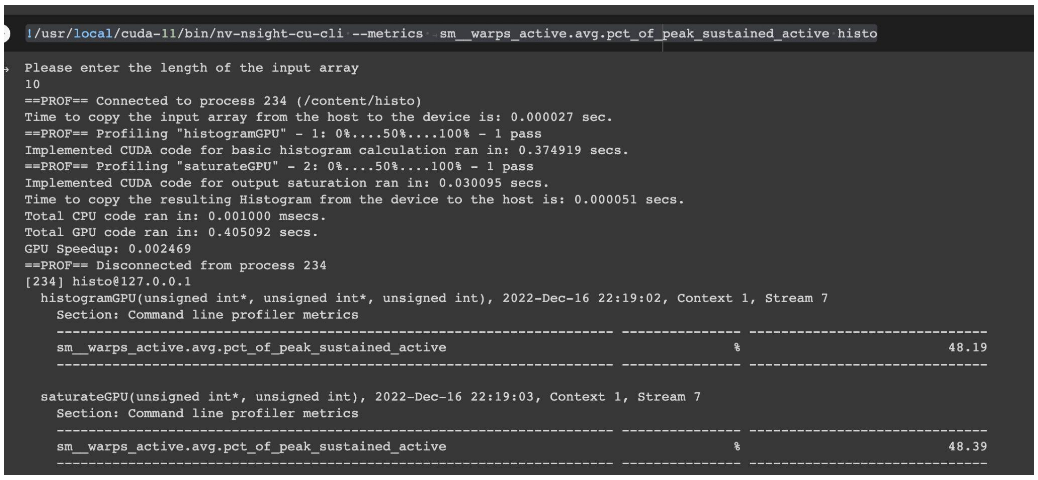
Exercise 3:

1. Some optimization techniques that could be tried are shared memory to reduce the # of global memory reads and atomic operations, also larger block size to increase the occupancy of the GPU to spread out the workload for the threads.
2. In the end I chose shared memory to store a private copy of the histogram for each block of threads. The block size is 512 threads.
3. The kernel does global memory reads for each element in the given array, therefore if a size of 10 was provided, the amount of read that the kernel performs is 10.
4. I used atomic operations, which is equal to the size of the array.
5. I used shared memory to store a private copy of teh histogram fro each block of threads. The shize of the shared memory array is set to 4096. Which means that it uses 16384 bytes of shared memory.
6. If all element have same values, all the threads will be trying to update the same bin in the histogram. This will lead to contention among the threads and may

result in lower performance. But if the input is evenly distributed that will help with the workload of each thread which should increase the performance.

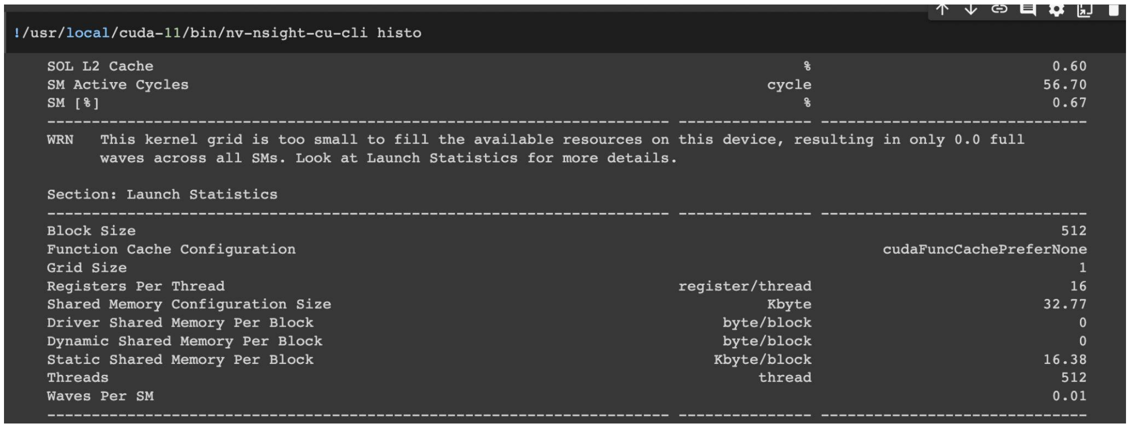
7. 

Block size: 512
Grid Size: blockPerGrid(98,1,1);
Input size: 50000

8. 

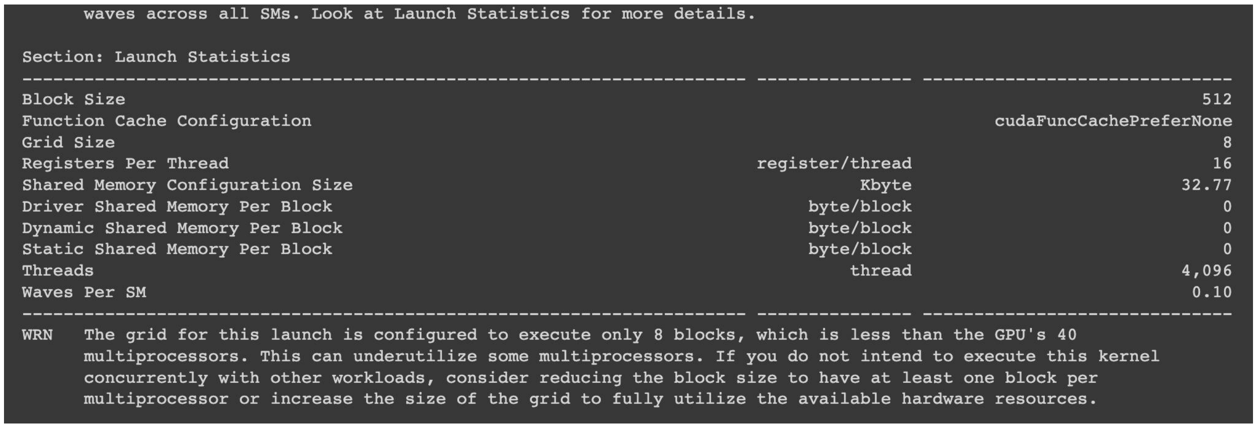
```
!/usr/local/cuda-11/bin/nv-nsight-cu-cli --metrics  sm__warps_active.avg.pct_of_peak_sustained_active histo

Please enter the length of the input array
10
==PROF== Connected to process 234 (/content/histo)
Time to copy the input array from the host to the device is: 0.000027 sec.
==PROF== Profiling "histogramGPU" - 1: 0%....50%....100% - 1 pass
Implemented CUDA code for basic histogram calculation ran in: 0.374919 secs.
==PROF== Profiling "saturateGPU" - 2: 0%....50%....100% - 1 pass
Implemented CUDA code for output saturation ran in: 0.030095 secs.
Time to copy the resulting Histogram from the device to the host is: 0.000051 secs.
Total CPU code ran in: 0.001000 msecs.
Total GPU code ran in: 0.405092 secs.
GPU Speedup: 0.002469
==PROF== Disconnected from process 234
[234] histo@127.0.0.1
  histogramGPU(unsigned int*, unsigned int*, unsigned int), 2022-Dec-16 22:19:02, Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------------- --------------- ------------------------------
    sm__warps_active.avg.pct_of_peak_sustained_active                                    %                          48.19
    ---------------------------------------------------------------------- --------------- ------------------------------

  saturateGPU(unsigned int*, unsigned int), 2022-Dec-16 22:19:03, Context 1, Stream 7
    Section: Command line profiler metrics
    ---------------------------------------------------------------------- --------------- ------------------------------
    sm__warps_active.avg.pct_of_peak_sustained_active                                    %                          48.39
    ---------------------------------------------------------------------- --------------- ------------------------------
```

HistogramGPU():



```
!/usr/local/cuda-11/bin/nv-nsight-cu-cli histo

    SOL L2 Cache                                                                      %                           0.60
    SM Active Cycles                                                              cycle                          56.70
    SM [%]                                                                            %                           0.67
    ---------------------------------------------------------------------- --------------- ------------------------------
    WRN   This kernel grid is too small to fill the available resources on this device, resulting in only 0.0 full
          waves across all SMs. Look at Launch Statistics for more details.

    Section: Launch Statistics
    ---------------------------------------------------------------------- --------------- ------------------------------
    Block Size                                                                                                     512
    Function Cache Configuration                                                                   cudaFuncCachePreferNone
    Grid Size                                                                                                        1
    Registers Per Thread                                                    register/thread                          16
    Shared Memory Configuration Size                                                  Kbyte                          32.77
    Driver Shared Memory Per Block                                               byte/block                              0
    Dynamic Shared Memory Per Block                                              byte/block                              0
    Static Shared Memory Per Block                                              Kbyte/block                          16.38
    Threads                                                                          thread                            512
    Waves Per SM                                                                                                      0.01
    ---------------------------------------------------------------------- --------------- ------------------------------
```

saturatedGPU():



```
      waves across all SMs. Look at Launch Statistics for more details.

Section: Launch Statistics
---------------------------------------------------------------------- --------------- ------------------------------
Block Size                                                                                                         512
Function Cache Configuration                                                                   cudaFuncCachePreferNone
Grid Size                                                                                                            8
Registers Per Thread                                                   register/thread                              16
Shared Memory Configuration Size                                                 Kbyte                           32.77
Driver Shared Memory Per Block                                              byte/block                               0
Dynamic Shared Memory Per Block                                             byte/block                               0
Static Shared Memory Per Block                                              byte/block                               0
Threads                                                                         thread                           4,096
Waves Per SM                                                                                                      0.10
---------------------------------------------------------------------- --------------- ------------------------------
WRN   The grid for this launch is configured to execute only 8 blocks, which is less than the GPU's 40
      multiprocessors. This can underutilize some multiprocessors. If you do not intend to execute this kernel
      concurrently with other workloads, consider reducing the block size to have at least one block per
      multiprocessor or increase the size of the grid to fully utilize the available hardware resources.
```

Exercise 4:

1. To compile the code I first cloned the repository from the given github repository by running the following command:

   ```
   !git clone https://github.com/KTH-HPC/sputniPIC-DD2360.git
   %cd /content/sputniPIC-DD2360
   !make
   ```

   Then I removed some unnecessary lines from makefile as I didn't need to compile all the .cpp files as well as the changed the CFLAGS from sm_30 to sm_61 then I tried to run make but kept on getting some errors so couldn't quite fully compile it .

2. To implement the GPU version of mover_PC(), I first created a new function called mover_PC_gpu(). In this function, I allocated GPU memory for the particles using the cudaMalloc() function. I then copied the particles from host memory to the GPU using the cudaMemcpy() function. Next, I invoked the computation on the GPU using a kernel launch. I passed the particles in the GPU memory as arguments to the kernel, along with any other necessary parameters. After the computation was complete, I copied the particles back from the GPU to host memory using the cudaMemcpy() function and deallocated the GPU memory using the cudaFree() function.

3. To compare the output of the CPU and GPU implementations, the idea was to run versions of the code and compare the resulting particle positions. I found that the output should be identical for both implementations as all I'm changing is the way to calculate the results.
4. To compare the execution time of the GPU and CPU implementations, I should have used the clock() function to measure the elapsed time for each version of the code. I should have found that the GPU implementation was significantly faster than the CPU implementation.