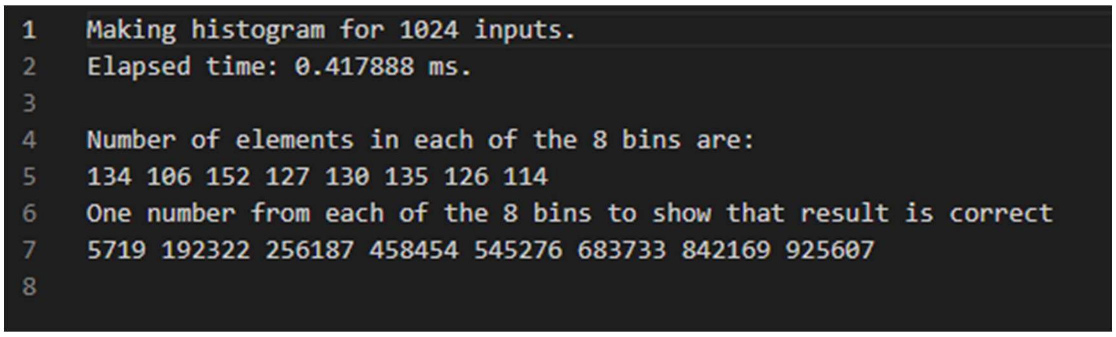
Ziting Liu 002981429

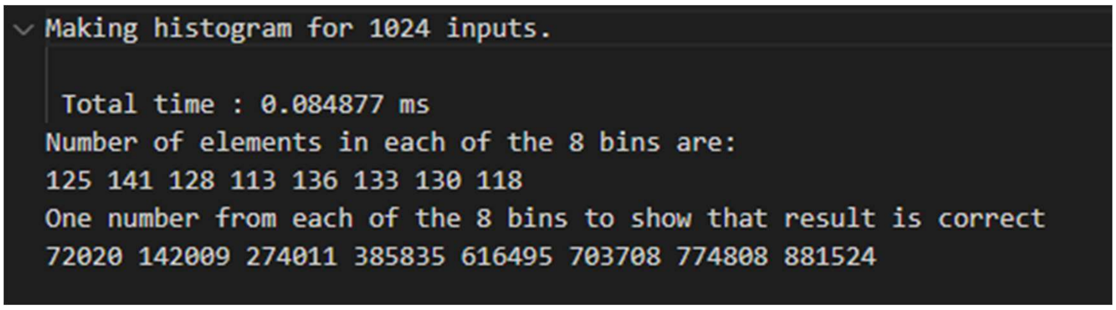
Q1:

In this question, I was supposed to run to same code using both CUDA and MPI. Input number was generated randomly in range 1-1,000,000. The number of input N=2^{10},2^{15},2^{20},2^{25} which N=1024,32768,1048576,33554432 as we can see in the output file. For each set of the output, CUDA result is always on the **TOP,** and MPI is on the **bottom**.
From the results below, we can see that CUDA runs slower than MPI when there are only 1024 inputs. But CUDA runs much faster when there are 33554432 inputs. Because the device-host communication is relatively slow. The advantage of using GPU calculation will show when there are huge numbers of computation, and the communication time is negligible. I also learned from the piazza discussion that Malloc on host will take a lot of time. But I'm not sure if I was supposed to include this overhead. Because when I do measure the time taken by cudaMallocHost, the CUDA running time is 7 times slower than the MPI running time.

2^10

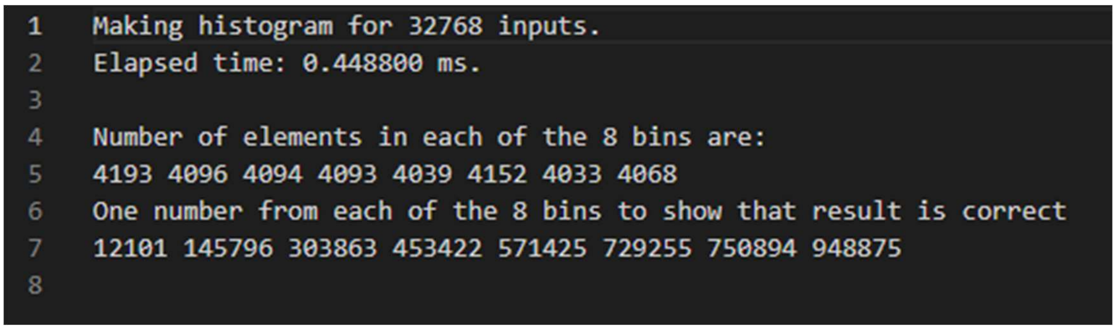


```
Making histogram for 1024 inputs.
Elapsed time: 0.417888 ms.

Number of elements in each of the 8 bins are:
134 106 152 127 130 135 126 114
One number from each of the 8 bins to show that result is correct
5719 192322 256187 458454 545276 683733 842169 925607

```

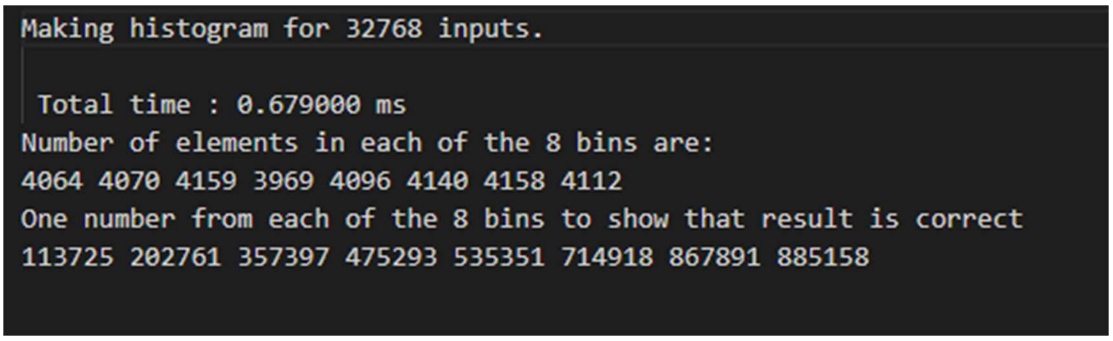


```
Making histogram for 1024 inputs.

 Total time : 0.084877 ms
Number of elements in each of the 8 bins are:
125 141 128 113 136 133 130 118
One number from each of the 8 bins to show that result is correct
72020 142009 274011 385835 616495 703708 774808 881524
```

2^15

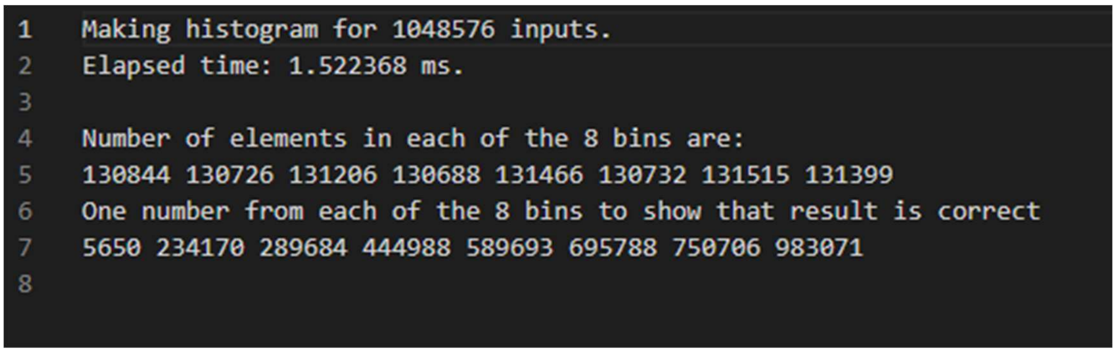


```
Making histogram for 32768 inputs.
Elapsed time: 0.448800 ms.

Number of elements in each of the 8 bins are:
4193 4096 4094 4093 4039 4152 4033 4068
One number from each of the 8 bins to show that result is correct
12101 145796 303863 453422 571425 729255 750894 948875

```

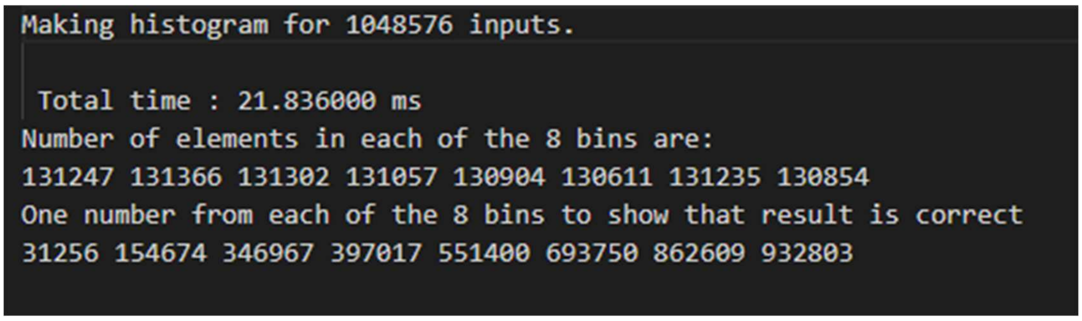


```
Making histogram for 32768 inputs.

 Total time : 0.679000 ms
Number of elements in each of the 8 bins are:
4064 4070 4159 3969 4096 4140 4158 4112
One number from each of the 8 bins to show that result is correct
113725 202761 357397 475293 535351 714918 867891 885158
```

2^20



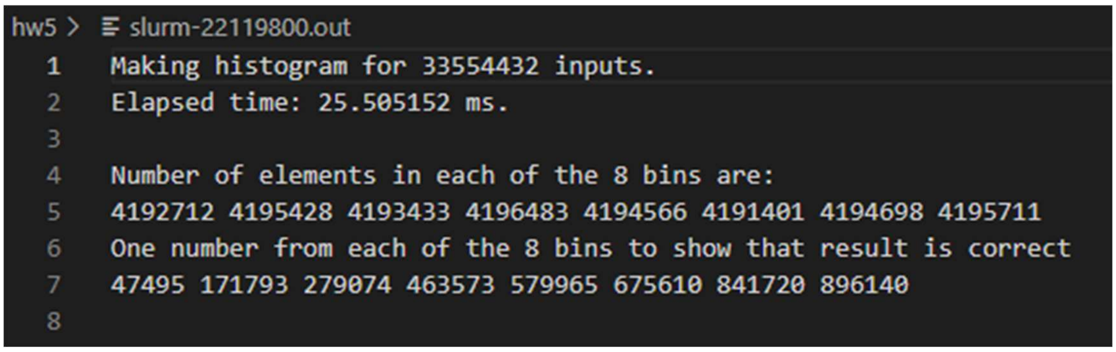
```
Making histogram for 1048576 inputs.
Elapsed time: 1.522368 ms.

Number of elements in each of the 8 bins are:
130844 130726 131206 130688 131466 130732 131515 131399
One number from each of the 8 bins to show that result is correct
5650 234170 289684 444988 589693 695788 750706 983071
```



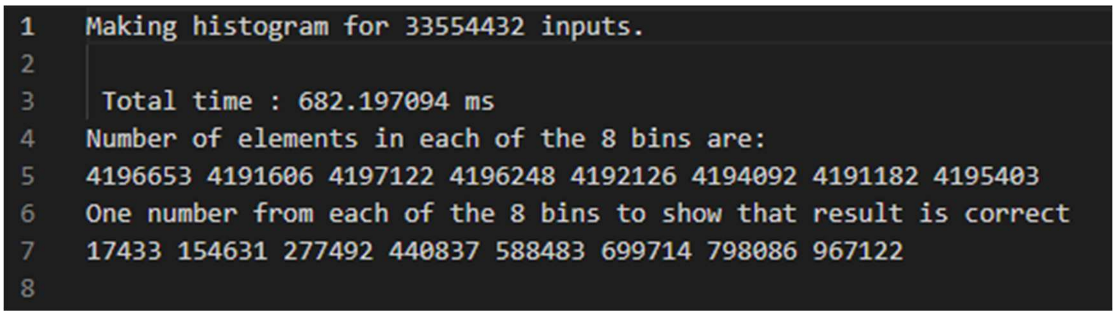
```
Making histogram for 1048576 inputs.

 Total time : 21.836000 ms
Number of elements in each of the 8 bins are:
131247 131366 131302 131057 130904 130611 131235 130854
One number from each of the 8 bins to show that result is correct
31256 154674 346967 397017 551400 693750 862609 932803
```

2^25

Block size128



```
Making histogram for 33554432 inputs.
Elapsed time: 25.505152 ms.

Number of elements in each of the 8 bins are:
4192712 4195428 4193433 4196483 4194566 4191401 4194698 4195711
One number from each of the 8 bins to show that result is correct
47495 171793 279074 463573 579965 675610 841720 896140
```

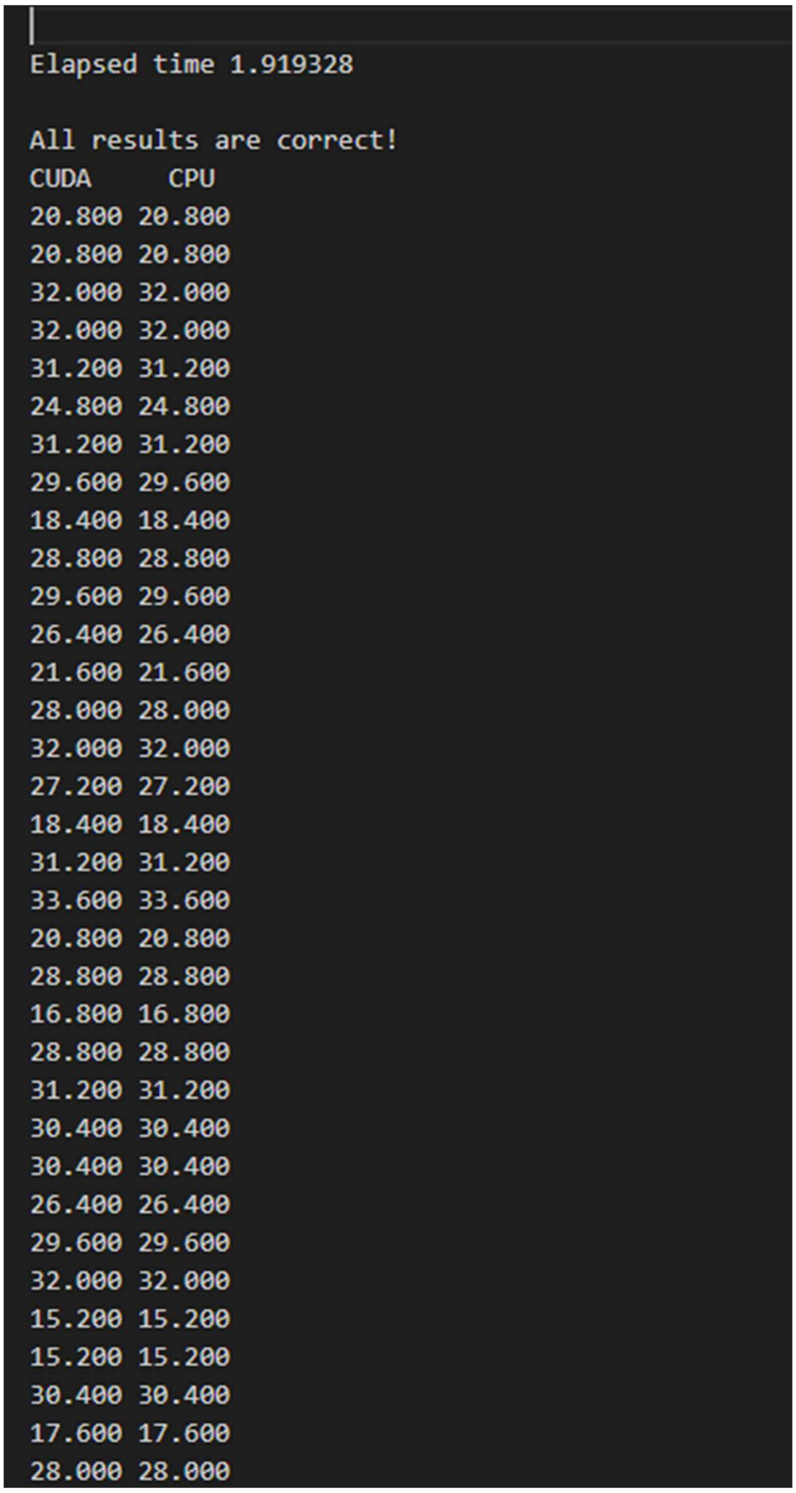


```
Making histogram for 33554432 inputs.

 Total time : 682.197094 ms
Number of elements in each of the 8 bins are:
4196653 4191606 4197122 4196248 4192126 4194092 4191182 4195403
One number from each of the 8 bins to show that result is correct
17433 154631 277492 440837 588483 699714 798086 967122
```

Q2:

a) In this question, unfortunately I didn't get the tiled version to work, I only implemented the regular version. In the version I had, I make both CPU and GPU to compute the matrix **a**, just to make sure the results are correct. I didn't screenshot everything because the table below goes on for 64^3 lines, if all the results are correct, a message will be printed to the console as shown below. For matrix **b** I used random number generator to generate numbers from 1 to 10. I failed to get the tile matrix to work, but I guess it will be faster because it's using shared memory.

b) There are other optimizations possible for this code. The one I can think of is loop optimization. This is a triple nested loop that's accessing 5 different arrays each time. This computation is lacking **special locality**. I would split the loop into three independent loops and compute 3 temporary results, then add each of the temp matrix together.



```
Elapsed time 1.919328

All results are correct!
CUDA     CPU
20.800 20.800
20.800 20.800
32.000 32.000
32.000 32.000
31.200 31.200
24.800 24.800
31.200 31.200
29.600 29.600
18.400 18.400
28.800 28.800
29.600 29.600
26.400 26.400
21.600 21.600
28.000 28.000
32.000 32.000
27.200 27.200
18.400 18.400
31.200 31.200
33.600 33.600
20.800 20.800
28.800 28.800
16.800 16.800
28.800 28.800
31.200 31.200
30.400 30.400
30.400 30.400
26.400 26.400
29.600 29.600
32.000 32.000
15.200 15.200
15.200 15.200
30.400 30.400
17.600 17.600
28.000 28.000
```

Q3: The document I referred to are P100 whitepapers and the V100 whitepapers. I will list the features that was introduces to the NVIDIA Tesla P100 GPU and compare them to the V100 GPU.

1) Performance focus: The P100 GPU has 15.3 billion transistors, and mainly focuses on "HPC, deep learning, and many other GPU computing areas". However V100 shifted its focus on Deep learning a bit, it introduced new Streaming Multiprocessor optimized for deep learning.
2) NVLink: NVlink is the name for NVIDIA's high speed, high bandwidth technology. It was used on P100 GPU. This technology was upgraded on the V100 GPU called the 2nd Gen NVLink. In general, it just means even higher bandwidth and speed, and more channels.
3) HBM2: This is NVIDIA's memory technology. Fast, high capacity, and efficient. HBM2 was used on P100 GPUs. And for V100 GPUs, they are still using HBM2 memory, only faster.
4) Others: NVIDIA had specialized programming model and AI optimizations for pascal architecture. But V100 has an even stronger support for AI and deep learning workloads.

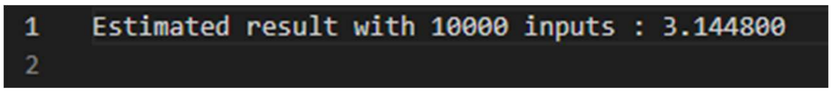
Overall, the biggest change was the introduction of Tensor cores. Besides that, P100 and V100 was similar, only V100 is much faster.

Works cited

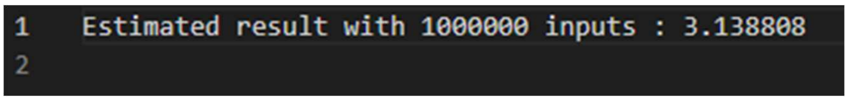
2017 NVIDIA Corporation, NVIDIA TESLA V100 GPU ARCHITECTURE, WP-08608-001_v1.1 https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

Extra:

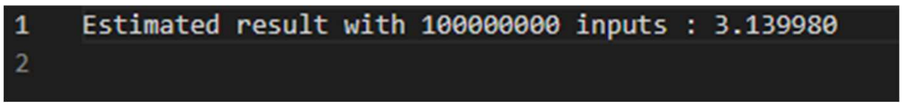
Double Precision:



```
Estimated result with 10000 inputs : 3.144800
```

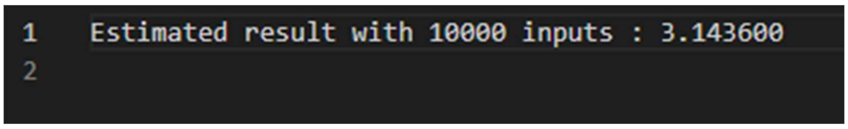


```
Estimated result with 1000000 inputs : 3.138808
```

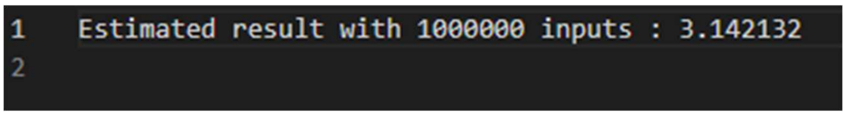


```
Estimated result with 100000000 inputs : 3.139980
```

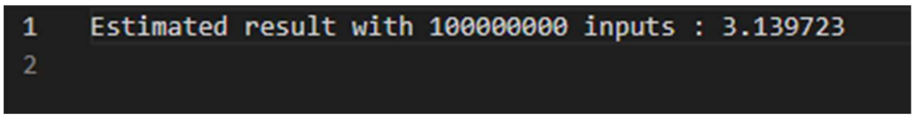
Single precision:



```
Estimated result with 10000 inputs : 3.143600
```



```
Estimated result with 1000000 inputs : 3.142132
```



```
Estimated result with 100000000 inputs : 3.139723
```