Question – 1

Assignment – 5

Let us revisit the histogramming problem assigned in Homework 4. As in Homework 4, your input to this program will be integers in the range 1-100,000 this time (use a random number generator that generates the numbers on the host). Your host-based data set should contain N integers, where N can be varied in the program.

a) This time you will implement a histogramming kernel in CUDA and run on a single GPU (P100 or V100). You can choose how to compute each class of the data set on the GPU. Attempt to adjust the grid size to get the best performance as you vary N. Experiment with $N = 2^{12} - 2^{23}$. When the GPU finishes, print one element from each class in class ascending order on the host (do not include the printing time in the timing measurements, though do include the device-to-host communication in the timing measurement). Plot your results in a graph.

b) Compare your GPU performance with running this same code on a CPU using OpenMP.

Answer:

a). N number of random integers need to be generated in the range 1 to 1000000, the number of bins is initialized to 10 and the performance of N is calculated from range from 2^{12} to 2^{23} :

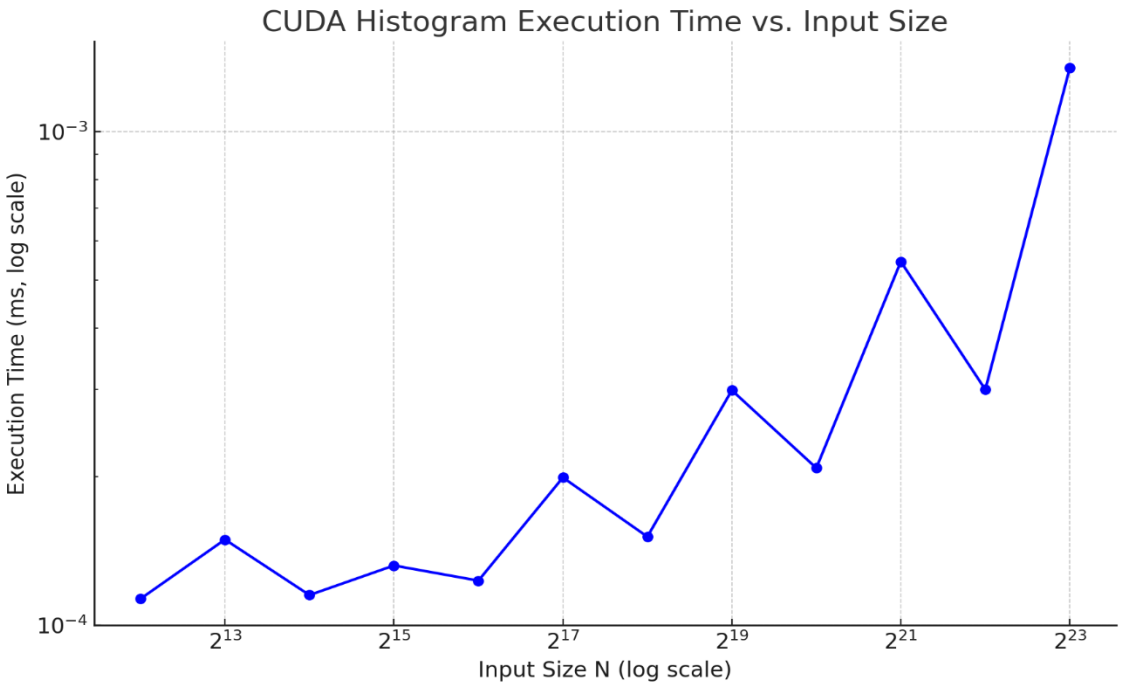
```
--- Histogram for N = 2^12 = 4096 ---
CUDA histogram completed in 0.001113 ms
Example input value from each class:
Class 0: 2271
Class 1: 15365
Class 2: 20406
Class 3: 32683
Class 4: 44161
Class 5: 54023
Class 6: 69634
Class 7: 72220
Class 8: 83273
Class 9: 97599

--- Histogram for N = 2^13 = 8192 ---
CUDA histogram completed in 0.000149 ms
Example input value from each class:
Class 0: 2759
Class 1: 11359
Class 2: 29051
Class 3: 34784
Class 4: 46887
Class 5: 57933
Class 6: 64082
Class 7: 79523
Class 8: 85125
Class 9: 98182

--- Histogram for N = 2^14 = 16384 ---
CUDA histogram completed in 0.000115 ms
Example input value from each class:
Class 0: 7414
Class 1: 19708
Class 2: 20230
Class 3: 36333
Class 4: 48568
Class 5: 50429
Class 6: 62755
Class 7: 71435
Class 8: 87443
Class 9: 96678
```

```
--- Histogram for N = 2^15 = 32768 ---
CUDA histogram completed in 0.000132 ms
Example input value from each class:
Class 0: 6249
Class 1: 14740
Class 2: 23613
Class 3: 34682
Class 4: 45341
Class 5: 50996
Class 6: 69568
Class 7: 79946
Class 8: 89229
Class 9: 90115

--- Histogram for N = 2^16 = 65536 ---
CUDA histogram completed in 0.000123 ms
Example input value from each class:
Class 0: 2575
Class 1: 13324
Class 2: 23631
Class 3: 36992
Class 4: 46112
Class 5: 52942
Class 6: 65645
Class 7: 74005
Class 8: 80653
Class 9: 92963

--- Histogram for N = 2^17 = 131072 ---
CUDA histogram completed in 0.000199 ms
Example input value from each class:
Class 0: 372
Class 1: 16038
Class 2: 26381
Class 3: 30716
Class 4: 44747
Class 5: 50819
Class 6: 61174
Class 7: 76655
Class 8: 88344
Class 9: 93219
```

```
--- Histogram for N = 2^18 = 262144 ---
CUDA histogram completed in 0.000151 ms
Example input value from each class:
Class 0: 4976
Class 1: 18143
Class 2: 28779
Class 3: 39266
Class 4: 46039
Class 5: 57168
Class 6: 65087
Class 7: 74928
Class 8: 88727
Class 9: 90393

--- Histogram for N = 2^19 = 524288 ---
CUDA histogram completed in 0.000299 ms
Example input value from each class:
Class 0: 5652
Class 1: 12942
Class 2: 24392
Class 3: 34217
Class 4: 45407
Class 5: 55930
Class 6: 62810
Class 7: 73890
Class 8: 82487
Class 9: 94466

--- Histogram for N = 2^20 = 1048576 ---
CUDA histogram completed in 0.000208 ms
Example input value from each class:
Class 0: 5678
Class 1: 19614
Class 2: 20795
Class 3: 34379
Class 4: 44353
Class 5: 55699
Class 6: 67353
Class 7: 78439
Class 8: 86793
Class 9: 93684
```

```
--- Histogram for N = 2^21 = 2097152 ---
CUDA histogram completed in 0.000545 ms
Example input value from each class:
Class 0: 16
Class 1: 14426
Class 2: 22060
Class 3: 32257
Class 4: 42931
Class 5: 52648
Class 6: 61756
Class 7: 79624
Class 8: 82139
Class 9: 90902

--- Histogram for N = 2^22 = 4194304 ---
CUDA histogram completed in 0.0003 ms
Example input value from each class:
Class 0: 4551
Class 1: 11683
Class 2: 25468
Class 3: 35271
Class 4: 47847
Class 5: 53417
Class 6: 64930
Class 7: 70409
Class 8: 86110
Class 9: 94502

--- Histogram for N = 2^23 = 8388608 ---
CUDA histogram completed in 0.001345 ms
Example input value from each class:
Class 0: 3649
Class 1: 17381
Class 2: 27687
Class 3: 33345
Class 4: 44598
Class 5: 59699
Class 6: 67907
Class 7: 77854
Class 8: 87409
Class 9: 93498
```

As the value of N increases, it increases the execution time although not linearly. It does not increase linearly due to memory bandwidth saturation.



(b): This part involves CPU programming in OpenMP; it uses thread-local histograms to avoid race conditions and then merges them with #pragma.

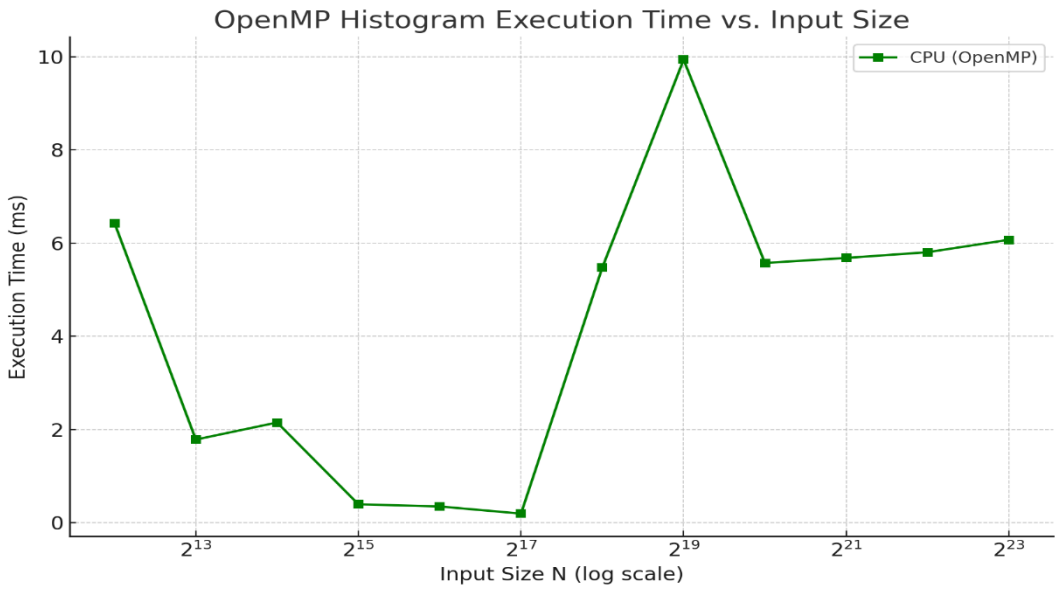
```
--- Histogram for N = 2^12 = 4096 ---
OpenMP histogram completed in 6.42752 ms
Example input value from each class:
Class 0: 864
Class 1: 19259
Class 2: 25709
Class 3: 33124
Class 4: 49610
Class 5: 52373
Class 6: 64435
Class 7: 79009
Class 8: 88368
Class 9: 95357

--- Histogram for N = 2^13 = 8192 ---
OpenMP histogram completed in 1.78246 ms
Example input value from each class:
Class 0: 8622
Class 1: 11990
Class 2: 20671
Class 3: 31703
Class 4: 46023
Class 5: 52790
Class 6: 66634
Class 7: 73984
Class 8: 88659
Class 9: 94463

--- Histogram for N = 2^14 = 16384 ---
OpenMP histogram completed in 2.14875 ms
Example input value from each class:
Class 0: 9323
Class 1: 18255
Class 2: 23265
Class 3: 31278
Class 4: 41559
Class 5: 55747
Class 6: 65425
Class 7: 76525
Class 8: 82326
Class 9: 94637
```

```
--- Histogram for N = 2^15 = 32768 ---
OpenMP histogram completed in 0.393518 ms
Example input value from each class:
Class 0: 3837
Class 1: 15117
Class 2: 20717
Class 3: 31648
Class 4: 49578
Class 5: 56436
Class 6: 62913
Class 7: 72089
Class 8: 80555
Class 9: 92871

--- Histogram for N = 2^16 = 65536 ---
OpenMP histogram completed in 0.347386 ms
Example input value from each class:
Class 0: 5266
Class 1: 19861
Class 2: 20909
Class 3: 38294
Class 4: 49167
Class 5: 57899
Class 6: 63742
Class 7: 77715
Class 8: 82082
Class 9: 94182

--- Histogram for N = 2^17 = 131072 ---
OpenMP histogram completed in 0.193808 ms
Example input value from each class:
Class 0: 5721
Class 1: 14187
Class 2: 25192
Class 3: 35410
Class 4: 40197
Class 5: 51355
Class 6: 62283
Class 7: 79792
Class 8: 87972
Class 9: 90562
```

```
--- Histogram for N = 2^18 = 262144 ---
OpenMP histogram completed in 5.47382 ms
Example input value from each class:
Class 0: 4075
Class 1: 10756
Class 2: 29804
Class 3: 33506
Class 4: 43478
Class 5: 54331
Class 6: 64098
Class 7: 77995
Class 8: 83307
Class 9: 98696

--- Histogram for N = 2^19 = 524288 ---
OpenMP histogram completed in 9.93213 ms
Example input value from each class:
Class 0: 111
Class 1: 18506
Class 2: 21655
Class 3: 38791
Class 4: 48260
Class 5: 50579
Class 6: 69775
Class 7: 79784
Class 8: 86326
Class 9: 97024

--- Histogram for N = 2^20 = 1048576 ---
OpenMP histogram completed in 5.57086 ms
Example input value from each class:
Class 0: 2402
Class 1: 16514
Class 2: 22927
Class 3: 31958
Class 4: 43226
Class 5: 55520
Class 6: 64177
Class 7: 74129
Class 8: 82018
Class 9: 90207
```

```
--- Histogram for N = 2^21 = 2097152 ---
OpenMP histogram completed in 5.67921 ms
Example input value from each class:
Class 0: 3390
Class 1: 16962
Class 2: 26176
Class 3: 36566
Class 4: 42418
Class 5: 51427
Class 6: 60660
Class 7: 77219
Class 8: 84179
Class 9: 91279

--- Histogram for N = 2^22 = 4194304 ---
OpenMP histogram completed in 5.80222 ms
Example input value from each class:
Class 0: 6393
Class 1: 11842
Class 2: 27250
Class 3: 38386
Class 4: 44662
Class 5: 53599
Class 6: 60248
Class 7: 76114
Class 8: 81673
Class 9: 95609

--- Histogram for N = 2^23 = 8388608 ---
OpenMP histogram completed in 6.06876 ms
Example input value from each class:
Class 0: 5505
Class 1: 14023
Class 2: 26756
Class 3: 32318
Class 4: 45873
Class 5: 51869
Class 6: 62278
Class 7: 72180
Class 8: 85008
Class 9: 98954
```

This is the graph of the CPU programming using OpenMP:



Comparison:

- The GPU's capacity to manage massive volumes of data with little increase in execution time is demonstrated by the consistently low GPU timings across all tested input sizes.
- When workload grows linearly with data size, CPU times rise with input size, exhibiting normal performance behavior. There is a noticeable amount of variation, though, which might be brought on by the CPU's inefficient management of extremely concurrent workloads.
- The decision between GPU and CPU programming is contingent upon the task's specific requirements, such as the nature of the problem, budget constraints, and performance objectives. GPUs provide unparalleled performance for tasks that are well-suited to high parallelism, whereas CPUs provide a wider range of applications with greater flexibility and simplicity of programming.