Q1:

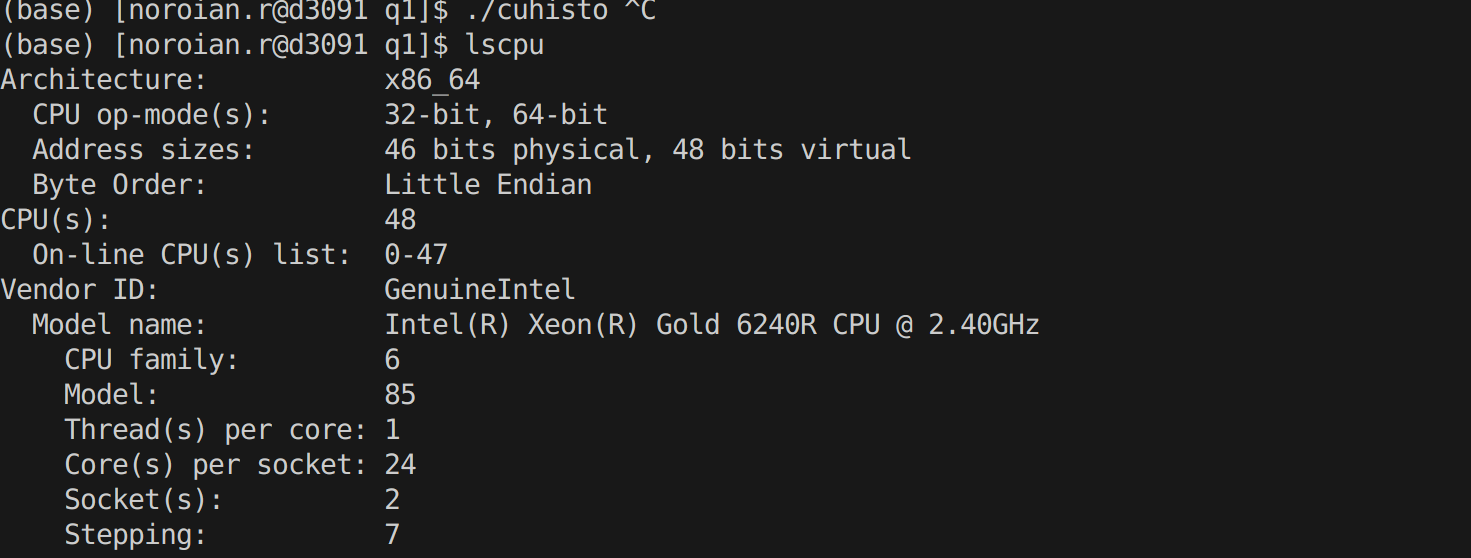
1. (40) Let us revisit the histogramming problem assigned in Homework 4. As in Homework 4, your input to this program will be integers in the range 1-100,000 this time (use a random number generator that generates the numbers on the host). Your host-based data set should contain N integers, where N can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

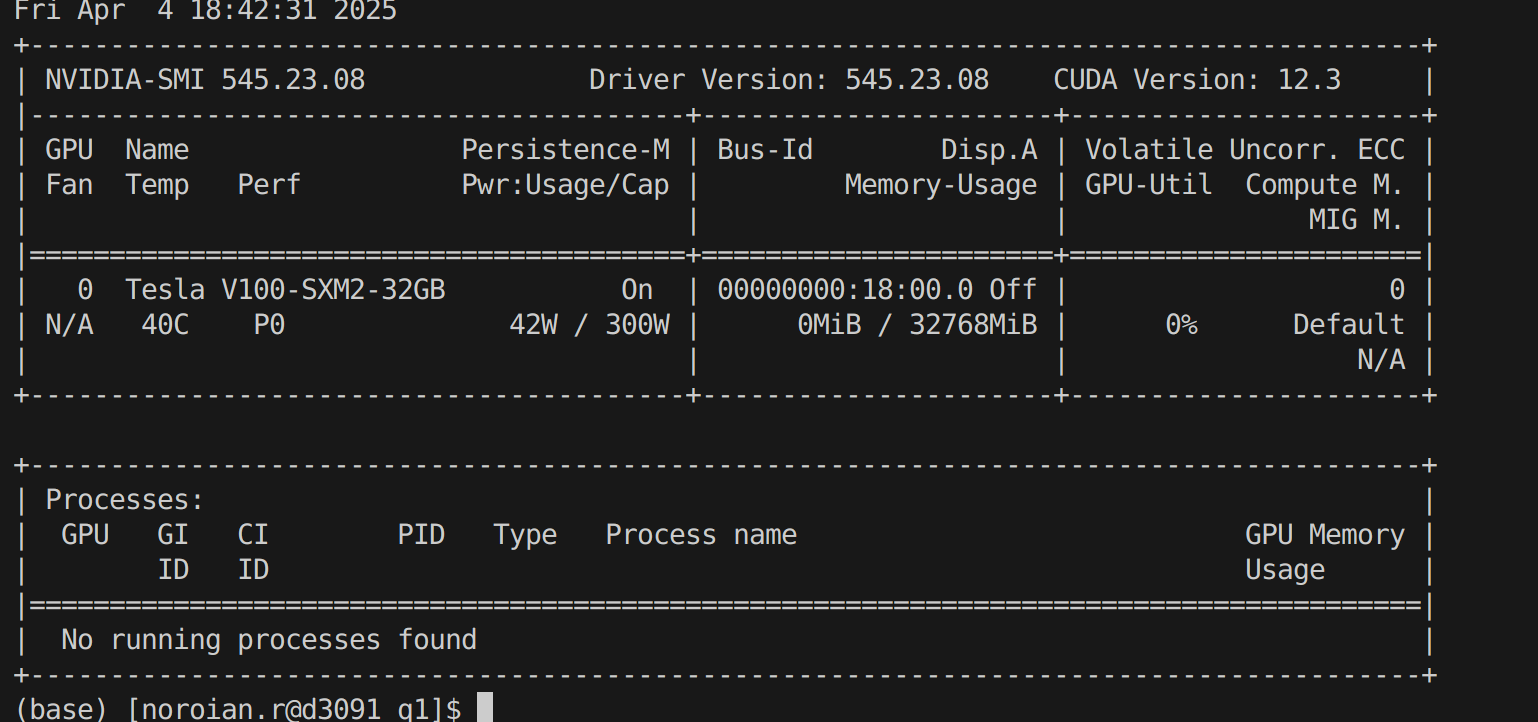
a. This time you will implement a histogram kernel in CUDA and run on a single GPU (P100 or V100). You can choose how to compute each class of the data set on the GPU. Attempt to adjust the grid size to get the best performance as you vary N. Experiment with N = 2 12 - 223 . When the GPU finishes, print 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 into a graph.

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

A and B are both included in this experimentation and analyzation. To build a histogram program for CUDA, I implemented atomic adds to a main histogram during kernel execution. Each thread within the GPU saves its own index of the dataset, and then synchronously adds it to the proper histogram bin. In OpenMP, I parallelized the bin classification and data collection, and synchronization ensured atomic updates to the main histogram. Each thread possessed its own histogram.

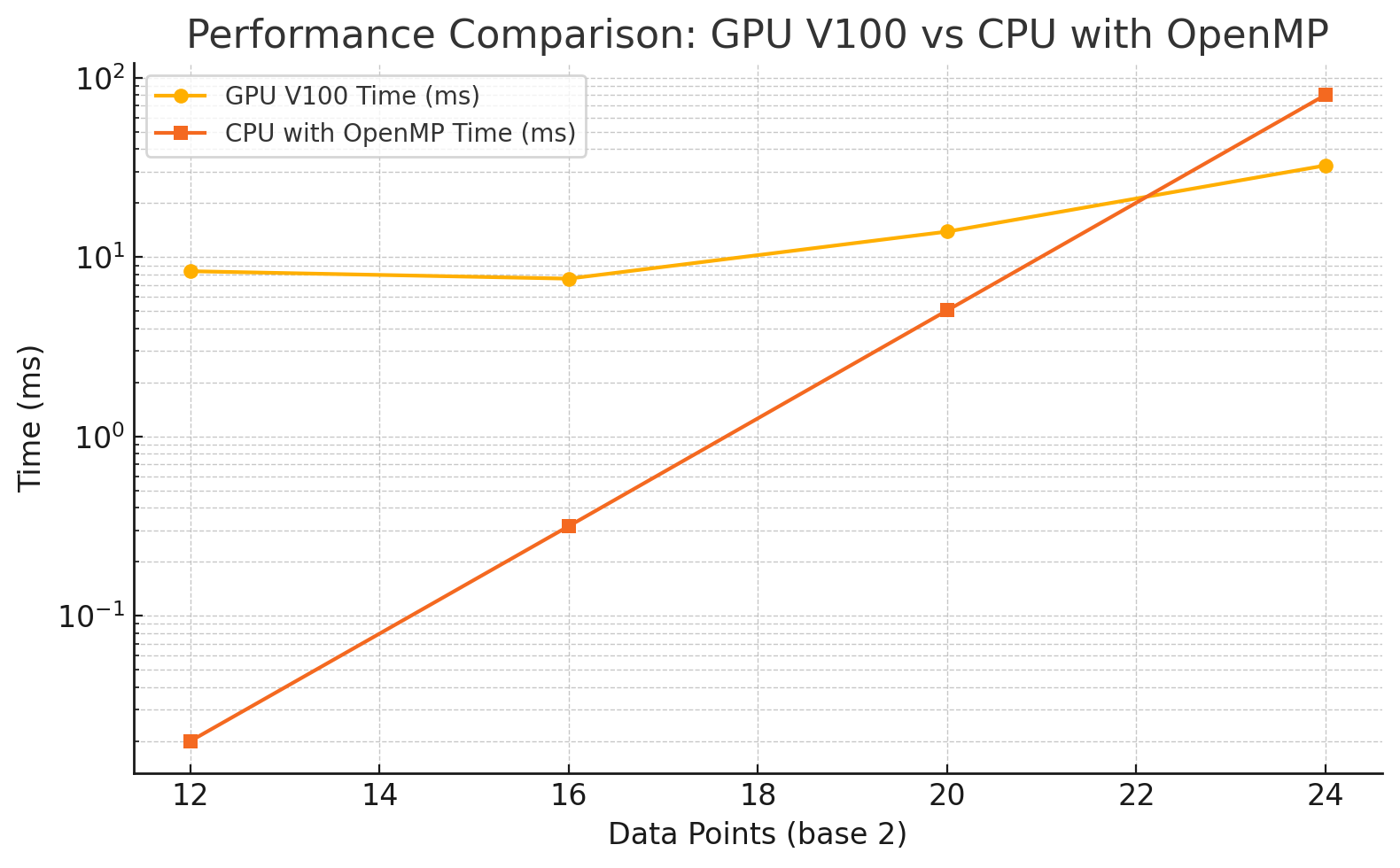
CPU (Used for OpenMP):

 V100 GPU node:



|  |  |  |
| --- | --- | --- |
| # Data points base 2 | GPU V100Time(ms) | CPU with OpenMP (ms) |
| 12 | 8.34 | 0.02 |
| 16 | 7.58 | 0.316 |
| 20 | 13.87 | 5.04 |
| 24 | 32.43 | 80.23 |

Graph of GPU & CPU performance:



Based on this graph, it can be concluded that the GPU processing power is more valuable for greater workloads. At 2 to the 24th power, it performs significantly better due to its ability to run the process in a massively parallel manner. This enables greater speedup compared to OpenMP, which does scale better. The associated overhead of the GPU and device communication leads to greater latency that disrupts the efficiency of the process. Though the GPU’s kernel execution may be similar in efficiency to OpenMP for fewer data points than shown in the graph, the overhead introduced with communicating with the CPU leads to inefficiencies as shown.

With fewer data points, OpenMP does perform better, as deployment on the GPU causes overhead, with the CPU and GPU communication introducing latency not present in OpenMP’s implementation. As the scale of histogramming increases, this becomes less of a challenge, as OpenMP isn’t able to efficiently parallelize the processing. This is where GPU kernel execution is incredibly fast at a massively parallel scale, outperforming OpenMP.