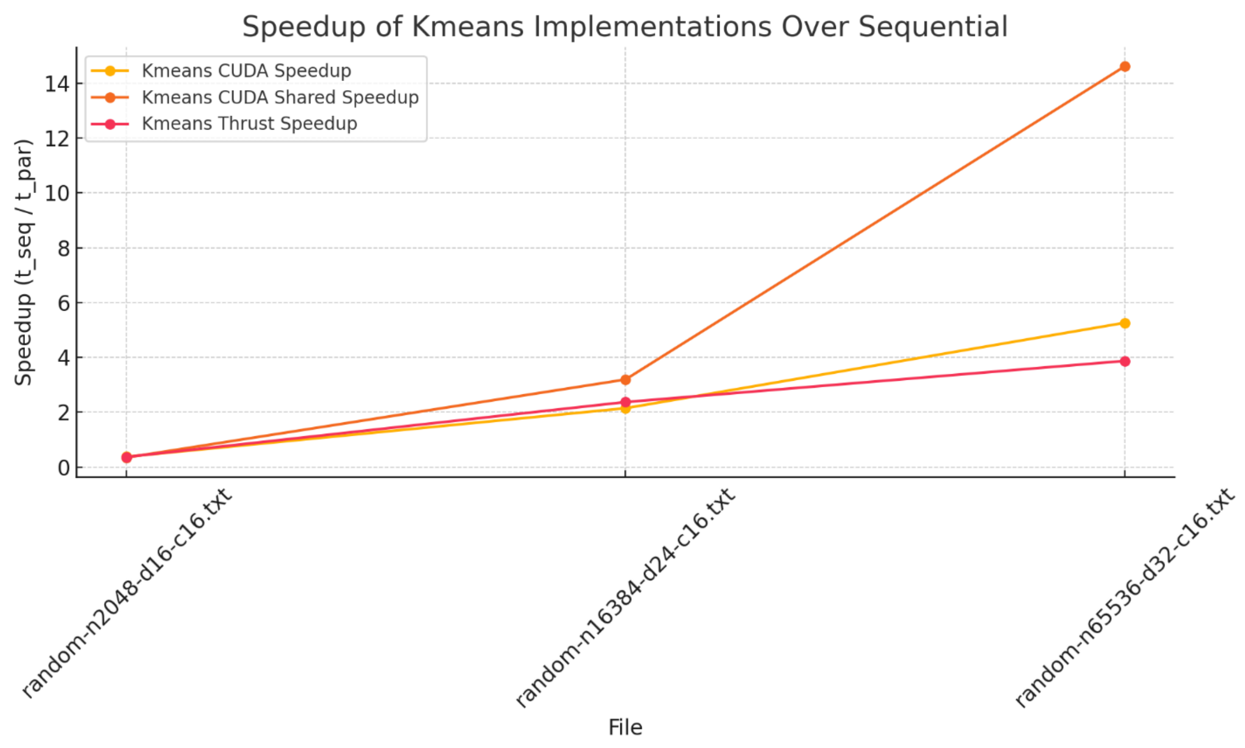
Abraham Zamora
AZ284
October 12, 2024

Lab 2 Report

This lab focused on using the GPU to achieve parallel performance on the K means algorithm. In order. To complete the programming portion for each, I wrote the sequential implementation first (kmeans.cpp). For each subsequent implementation, I used the structure of the sequential implementation and just adapted the program for each parallel implementation respectively. (CUDA, CUDA shared. and Thrust). As such, each implementation appears very similar but each focuses on the particulars of the implementation. To test each implementation, the assignment provided three files, and the output was provided for us to check our implementation. After testing each file, I noted the time it took each one to complete. There is a caveat that needs to be shared. The thrust implementation is not fully functional, and debugging this one proved to be quite difficult. The documentation was not ideal, and it truly was a challenge. With this caveat in mind, I still felt that the implementation should be tested in case the issue was something minor and required a minimal fix. The results are below. Times are in microseconds

File	Kmeans Sequential	Kmeans CUDA	Kmeans CUDA Shared	Kmeans Thrust
random-n2048-d16-c16.txt	109	288	313	294
random-n16384-d24-c16.txt	1022	476	320	431
random-n65536-d32-c16.txt	4966	944	340	1283

In addition to testing the time, it was also important to demonstrate the benefit of the parallel implementations. As such, I decided to graph the speedup similar to how we did for assignment one. The results can be seen below.



Observations

Going into the assignment, I expected that one of the CUDA implementations might perform the best. Given the tech specifications on Codio and the use of 256 threads per block, in the best-case scenario, a 200x or better can be expected. There is still overhead due to the transferring of data from the CPU to the GPU, but at a higher workload, it becomes a better idea to use the GPU over the CPU. Of course, in practice, the 200x might not be achievable. I believed that the shared memory implementation might perform the best. Sharing memory reduces the overhead associated with accessing global memory in the GPU. It is too easy to say that I expected the sequential to perform worse, especially as the workload increased. This can be seen on the table and the graph. The sequential overperformed early, but then underperformed as the workload increased. This was on par with my expectations especially after the first lab. Of the parallel implementations, I expected the thrust version to underperform as the workload increased. Thrust is designed for ease of use rather than maximum performance. It abstracts many low-level optimizations that CUDA and shared memory implementations can fine-tune. As such, the heavier workload would likely not improve the speed compared to the other two CUDA implementations. This proved to be the case. The graph demonstrates that the shared memory performed the best over the sequential implementation. The regular CUDA implementation came in second when the workload increased. In third was the Thrust implementation. It achieved faster speed at moderate and lower workloads before being surpassed, but it did not achieve the 14x improvement that the CUDA shared memory implementation had. Still, a nearly 5x improvement over sequential implementation is substantial. The improvement for the shared memory implementation is substantially higher than any of the improvements from our previous lab. For both CUDA implementations, I also tracked how much time it took to transfer data from the CPU to the GPU and back to the CPU. For the shared memory implementation, about 1.2% of the total elapsed time came from transferring the data between the CPU and GPU. This amounted to about 3 microseconds or less each time. The same for the standard CUDA implementation. Time spent transferring data amounted to about 3 microseconds or less. This is less than 1% of the total elapsed time. This is quite fast for both, but it isn't unsurprising that the shared memory takes a bit longer to as a percentage of the total elapsed time.

This lab was challenging and a step up from the first lab. In total, this assignment took about 25 hours total time. The programming and debugging took about 22-23 total hours to complete. The last portion of the time was dedicated to writing out the report and noting the observations after running each implementation.

Works Cited

NVIDIA. *K-Means Clustering Algorithm*. Retrieved September 20, 2024, from https://www.nvidia.com/en-us/glossary/k-means/

Goldsborough, P. (2017, September 10). *Exploring K-Means in Python, C++, and CUDA*. Retrieved September 25, 2024, from http://www.goldsborough.me/c++/python/cuda/2017/09/10/20-32-46-exploring_k-means_in_python,_c++_and_cuda/

NVIDIA. (n.d.). *Thrust: The C++ Parallel Algorithms Library*. Retrieved October 6, 2024, from https://nvidia.github.io/cccl/thrust/