Student Name: - ATHARVA DESHPANDE

Student Email: - deshpana@oregonstate.edu

Project No: - Project#5

Project Name: - CUDA: Monte Carlo Simulation

ATHARVA DESHPANDE CS575 deshpana@oregonstate.edu

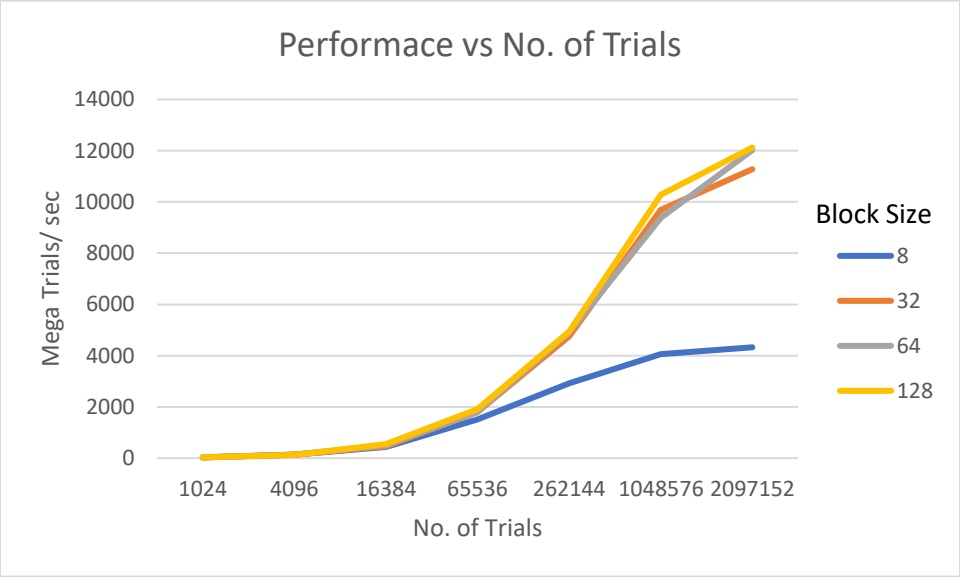
1. Tell what machine you ran this on.
   ➔ I ran the program on windows 11 system using OSU DGX server. I wrote the code using vim, imported other helper function files and executed it on the server to get the Monte Carlo Simulation results using CUDA.

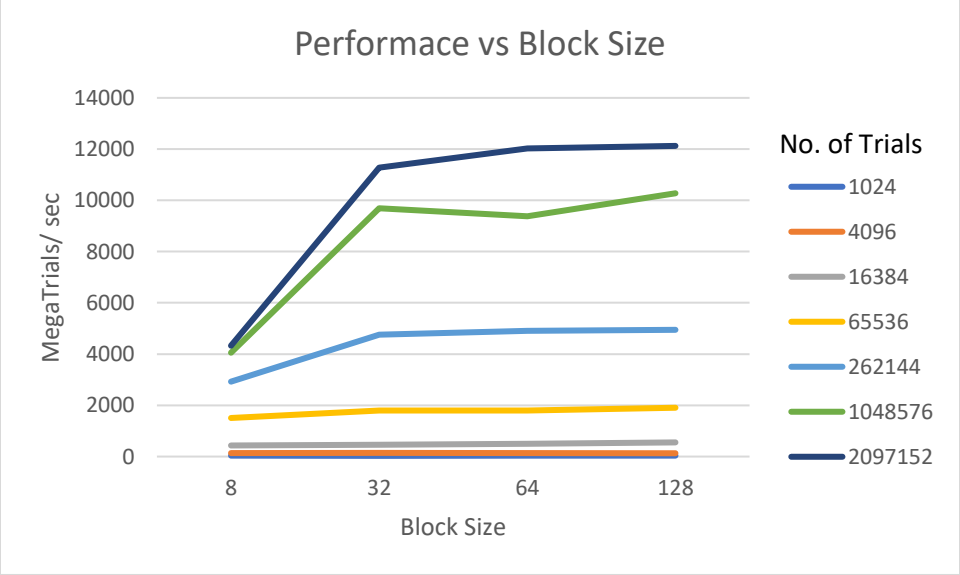
2. What do you think this new probability is?
   ➔ The average probability value that was seen for the highest number of trials (2097152) across multiple block sizes was 74.70.

3. Show the table and two graphs.
   ➔ The following is the result table obtained:

No of trials	Block Size	Mega Trials/ sec	Probability
1024	8	31.25	75.2
1024	32	26.3158	76.07
1024	64	34.4828	75
1024	128	31.25	75.49
4096	8	137.931	73.29
4096	32	142.8571	74.34
4096	64	137.931	73.17
4096	128	129.0323	74.85
16384	8	432.4324	74.69
16384	32	457.1428	74.77
16384	64	500	74.72
16384	128	551.7241	74.57
65536	8	1505.882	74.71
65536	32	1796.491	74.42
65536	64	1801.231	74.5
65536	128	1905.116	74.5
262144	8	2924.67	74.86
262144	32	4754.498	74.64
262144	64	4902.454	74.81
262144	128	4946.86	74.71
1048576	8	4053.94	74.63
1048576	32	9694.675	74.71
1048576	64	9378.363	74.71
1048576	128	10275.32	74.75
2097152	8	4326.951	74.74
2097152	32	11277.92	74.65
2097152	64	12024.95	74.73
2097152	128	12125.07	74.69

The corresponding graphs for performance vs No. of Trials and performance vs block size are:





4. What patterns are you seeing in the performance curves?
   ➔ In the first graph, we can see that performance improves as we increase the number of trials for different block sizes. However, the best performance occurs when the block size is 128 and the number of trials is at its maximum of 2097152. When the block size is increased, it allows more threads to run at the same time on the GPU, which leads to better performance. But this also requires a larger dataset to make up for the extra processing time. So, larger block sizes and more trials result in good performance.
   In the second graph, we can see that the best performance is achieved when the number of trials is 2097152. It's also important to note that this performance improvement happens when the block size is 128. This is because having more trials allows the GPU to use its resources more effectively, which leads to better performance due to increased parallel processing.

5. Why do you think the pattern looks this way?
   ➔ The GPU has the ability to run multiple blocks at the same time, which means that many threads can be executed simultaneously. This results in a significant level of parallelism, leading to improved performance. When we increase the number of trials and the block size, we also enhance the utilization of the GPU's resources. On the other hand, if the block size and number of trials are small, the GPU is not fully utilized, and this leads to a performance overhead.
   In both graphs, we observe that the performance improvement reaches a plateau after a certain point for any block size. This is because the extent of performance improvement depends on how much parallelism can be effectively implemented and executed in the code.

6. Why is the BLOCKSIZE of 8 so much worse than the others?
   ➔ When the block size is smaller, such as 8, there is limited parallelism. This means that only a small number of threads, in this case, 8, can be executed at the same time. On the other hand, when the block size is 128, there are 128 threads that can run concurrently. Having a smaller block size with fewer threads reduces the potential for parallel execution on the GPU. As a result, the GPU's resources are not fully utilized, leading to lower overall performance.
   With a block size of 8, there are fewer threads available to efficiently utilize the GPU's resources. This creates competition among the threads for resources, which negatively impacts performance. However, when the block size is large enough, at least 64, the GPU's resources are utilized more effectively, resulting in performance improvement.

7. How do these performance results compare with what you got in Project #1? Why?
   ➔ Using CUDA provides a significant improvement in performance compared to using OpenMP, as observed in Project #1 and Project #5. CUDA is designed specifically for handling massive parallelism, which is when many tasks can be executed simultaneously. GPUs are well-suited for executing highly parallel tasks. On the other hand, OpenMP supports parallelism using shared memory but relies heavily on the CPU cores for implementing this parallelism.

In simpler terms, CUDA is better for tasks that require a high level of parallelism, while OpenMP relies more on the CPU and is not as efficient for highly parallel tasks.

8. What does this mean for what you can do with GPU parallel computing?
   ➔ Using a GPU with massive parallelism allows you to efficiently handle complex tasks by dividing them into smaller parts that can be processed simultaneously. This leads to faster and more efficient computations, especially when dealing with large datasets or complex programs that can be parallelized. By utilizing larger block sizes, you can further speed up and optimize these computationally intensive processes.