Matrix Transpose

Nandhana Sakthivel

Exercise 6

Objective

The objective of this exercise is to compare the performance of implementing **transpose of a matrix** with global memory and shared memory using the **CUDA** language.

Result

Bandwidth is calculated in **GB/s** via the following formula:

Bandwidth = Size of the matrix * Size of the element * 2 (load and store operations) / Time taken

The size of the matrix is considered to be **2048 * 2048** and the thread count is varied for exeperimental purpose. We know that the **TeslaK20m** has a memory bandwidth of **208 GB/s**.

Comparison between the performance of Naive Transpose and Coalesced Transpose

Threads per block	Naive Transpose Time (ms)	Bandwidth (GB/s)	Coalesced Transpose Time (ms)	Bandwidth (Gb/s)
64	0.748	44.841	0.459	73.030
256	0.650	51.618	0.311	107.867
1024	0.870	38.550	0.493	67.934

From the table given above, we can see that the performance is much better in **Coalesced Transpose** when compared to the **Naive Transpose**. We can see that when the thread count is **256 per block**, the performance is higher and it is **51.8%** of the theoretical performance of the GPU. When we increase the thread count per block to **1024** the performance drastically reduces to **67.934 GB/s**

Conclusion

We can conclude that the performance of **coalesced memory access** using the shared memory is better than the global memory access and also the performance of thread block of size **256** is better than **1024** because the increase in the number of threads increases the waiting time for all the other threads to reach the **syncthreads** barrier. Hence the performance is better when the thread count is less.