Matrix Transpose

By: Rabindra Khadka

Overview:

The objective of this lab was to optimize the transpose code of a matrix by adopting out of place arrays operation in memory. For simplicity, only square matrices were taken into account for the purpose of this exercise. The time (ms) for each transpose operation has been recorded relative to the number of threads per block and the effective bandwidth has been calculated in GB/s as a performance metric .

Bandwidth is given by :
[matrix size * size of element * 2 (for read and write operation)] divided by **[execution time]**.

The size of the matrix was chosen to be **8192 * 8192 a**nd different amount of threads per block were chosen to launch the transpose kernel both in coalesced fashion using shared memory and uncoalesced fashion using global memory. For coalesce access , the matrix was divided into tiles with the tile size k* k which is the size of a thread block. Threads in a thread block will copy the tile into shared memory and perform the transpose in the shared memory which will be finally copied to global memory. The kernels map threads to our given matrix element with the use of thread index in x and y direction; **threadIdx.x** and **threadIdy.y** respectively. Blocks per grid is provided by (N/k,N/k) assuming N is the multiple of k . With the purpose of bench-marking, the kernels were launched repeatedly for 100 times and the average elapsed time were calculated.

With the help of deviceQuery, we noted that the Tesla K20 m GPU has **memory buswidth** of **320 bits** and **memory clock rate** of **2600 MHz,** therefore the **theoretical peak bandwidth** of our GPU stands at **104 GB/s**

Outcome :

The elapsed run time for the transpose of the given matrix using different number of threads for both naive matrix transpose and coalesced matrix transpose have been tabulated below.

Tesla k20m				
# of threads per block	**Naive Transpose Time(ms)**	**Bandwidth (GB/s)**	**Coalesced Transpose Time (ms)**	**Bandwidth (GB/s)**
64	11.029	48.6779	8.52912	62.9456
256	9.3156	57.6311	5.20995	103.047
1024	9.837	54.5735	7.53979	71.2051

Table 1: Record of elapsed time on cuda for Naive Transpose and Coalesced Transpose.

As we can observe in the table above, the bandwidth recorded with 64 threads per block stands at 46.8% and 60.5 % of peak bandwidth for naive transpose and coalesced transpose respectively. This percentage is greatly improved to **99.11 % of peak bandwidth** when we launch the kernels with 256 threads per block. As we increased the threads to 1024 per block, the bandwidth drastically dropped for the coalesced transpose kernel to 71.2051 GB/s.

Conclusion:

Having observed the elapsed time and the bandwidth, we can conclude that by performing the coalesced global memory access for read and write; and using shared memory for the transpose yields the best performance bandwidth than the uncoalesced read and write operation. We also observed that by shrinking the tile size to 16 * 16 i.e. using 256 threads per block the bandwidth achieved was close to the peak bandwidth. This is due to the fact that having higher number of threads will lead some threads to spend large chunk of their life time waiting for other threads to finish the operation at synchthreads barrier. So, by reducing the number of threads per block upto a certain point, will reduce the time between memory transactions and hence results in higher bandwidth.