Day 5- CUDA Transpose of a Matrix

Nicola Meneghini

Aim of the exercise was to write a CUDA program to transpose a matrix.

The first approach consists in creating one thread for each element of the matrix and then generate one index for both of dimensions of the matrix; clearly I will have a number of blocks `N_block = N*N/n_threads`.

Secondly I wrote a function that exploits the shared memory of the threads, which basically means that I take a block, transpose it in cache and then read it in the new matrix - this is done for each block in the grid. However, before every read, I have to synchronize threads in order to coordinate their access to the fast memory. Also, note that `temp` has dimensions `[BLOCK_X][BLOCK_Y+1]` to avoid bank conflicts, as suggested at the end of the article cited in the README.md.

The plot below shows the performaces of these two functions with a square matrix of size $N = 8192$. We can see that in both cases we need at least 128 threads to exploit all the GPU resources. Once reached this point, the time required for the shared memomry approach is around $10ms$, which is far better than the $30ms$ spent with the "naive" function.