COL380 A4 Report

Ananya Mathur - 2020CS50416

April 2023

1 Algorithm

Every thread computes one element of output matrix. Only non zero blocks in both input matrices are passed as input to kernel. The starting and ending indices in the non zero blocks list corresponding to non zero blocks each row is a part of are also precomputed and passed as kernel input. Now, each thread of any block computes its row and column number using block and thread ids and traverses the non zero blocks in the list containing that row and column. This set of non zero blocks would be clearly the same for all threads in one thread block. Only those non zero blocks from both the matrices can be mutiplied which have the other than row/column indices equal. For these feasible blocks, the required row and column is multiplied by each thread in a given thread block corresponding to its output element.

There are $(\frac{n}{m})^2$ blocks in each grid and each block contains m^2 threads.

2 Optimisation

I used 2 pointers to iterate through the non zero blocks of both matrices and check feasibility of block multiplication. Also, sending only non zero blocks and using short int for 2 byte element input matrices reduced the memory footprint. Instead of checking all blocks of a given row and column for multiplication feasibility I used row and column indices to check for feasibility only among non zero blocks of that row and column exploting the sparseness of the matrices.

3 Observations

For n = 2^{15}, m = 8, k = 8800000, my algorithm takes 420 secs. This is the worst case as per assignment pdf.

For n = 2^{13}, m = 4, k = 2000000, time taken is 34 secs.