My solution follows directly the one described in the paper.

To calculate optimal pm, pn, pk I started with Lagrange Multipliers but I realized that when working with floating point numbers one need to check both floor and ceil of such a solution so I decided to switch to a for loop for simplicity.

When the number of MPI_COMMON_WORLD processes is greater than pm*pn*pk I am creating new world with processes which rank > pm*pn*pk are excluded.

The matrices A and B are generated by the first K-Task group, if A or B are not replicated then all Canon groups in the first K-Task generate them, otherwise only the first Cannon group.

Each step in the algorithm has its own MPI communicator to avoid unintuitive indexing needed when using only one communicator.

To increase performance I am storing the matrix parts as a continuous array rather than 2d one which improves locality of memory access. When we matmul A times B, we access A elements in row order and B in column order so the continuous array of A i row by row whereas B array is column by column.

Printing is not optimized but in order to compute number of elements which are greater than or equal I am using MPI_Reduce function to improve the performance.