COL380 Assignment 4 Report

- Aim :
    - We need to calculate the answer of multiplication of N sparse matrices
- Parallelization Strategy :
    - Distributed Processing with MPI :
        - Matrix Distribution: Matrices are divided across MPI ranks using cyclic distribution. Each rank processes `floor(N/size) + (rank < N%size)` matrices
        - Communication Pattern: Broadcasting the information present in file size to all processes from rank 0. Three-phase gathering of metadata, matrix data, and index differences read from files by each MPI process to rank 0 process. Master rank (0) reconstructs full matrix graph for path finding
    - Path construction :
        - We create an adjacency list based on the height and width of the matrices
        - We perform a dfs to identify one valid path of matrix multiplication
    - Matrix multiplication heuristic :
        - Cost-based Pair Selection: Parallel OpenMP loop calculates min/max costs for adjacent matrix pairs. The formula used is (A.numNonZero * B.numNonZero) / A.width
        - Intersection Handling: GPU priority is given when min/max pairs intersect we execute CUDA kernel for max-cost pair and then we update matrix graph in-place
        - We then reduce the global count of number of matrices by 1

        - For the non-intersection of min/max pairs, we perform concurrent CUDA and OpenMP matrix multiplication for max/min cost respectively.
        - We update the result of these multiplication in place so as to preserve the path order and then reduce the global count by 2

    - CUDA Matrix Multiplication :
        - launchCudaKernel Workflow: Only MPI rank 0 executes CUDA operations.
        - Pair Identification: Find matching block indices (A.col == B.row)
        - Device Allocation: This is done using cudaMalloc
        - Data Transfer: Host-to-device copies via cudaMemcpy
        - blockMultKernel Design : Each CUDA block processes one matrix pair (pairIdx = blockIdx.x). 2D thread layout (threadIdx.x/y) matches block dimensions
        - Timing is recorded using cudaEventRecord

    - OpenMP Matrix Multiplication :
        - Block Pair Identification is done similar to what was done during CUDA multiplication
        - Parallel Computation: 3D loop structure does arithmetic operations and Modulo operation prevents integer overflow

    - We will be repeating this process until the global number of matrices becomes 1

- Performance analysis of OpenMP, CUDA and MPI
    - The medium test case (provided) analysis (2 nodes, 4 cpu, 1 gpu) :
        - N = 5,
        - K = 4
        - MPI : Rank 0 (master process) time = ~3 seconds
        - CUDA Matrix multiplication time = ~0.006 seconds
        - OpenMP Matrix Multiplication Time: ~0.12 seconds

The MPI time includes reading the files, performing matrix multiplication and then writing the files i.e the complete duration of the code execution for one process