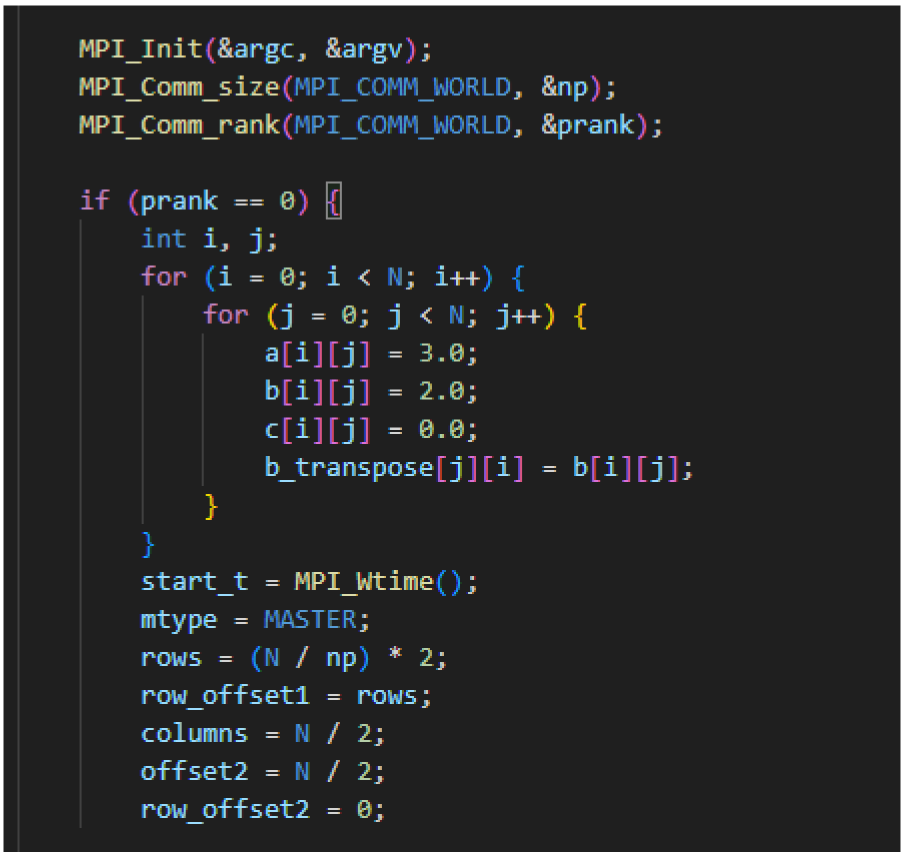
OPENMPI IMPLEMENTATION OF MATRIX MULTIPLICATION

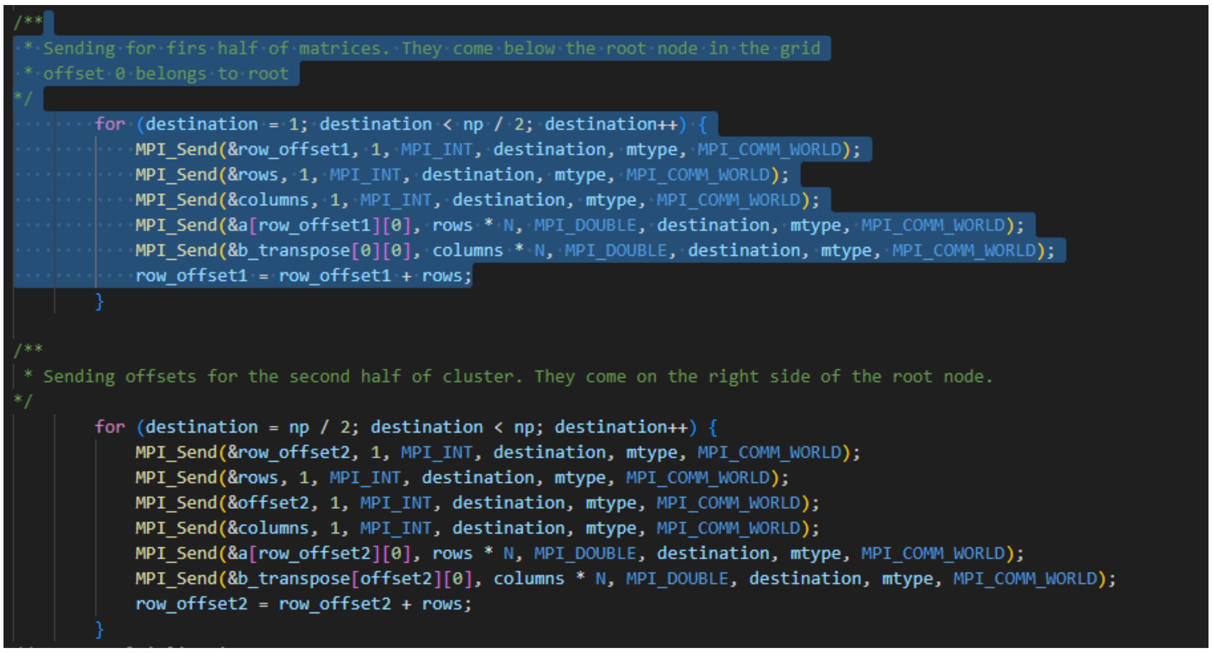
- The following code explains how to matrix multiplication using openmpi and 2d matrix partitioning.

Code highlights (**Constraint - number of processors should be a factor of matrix size**)



```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &np);
MPI_Comm_rank(MPI_COMM_WORLD, &prank);

if (prank == 0) {
    int i, j;
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            a[i][j] = 3.0;
            b[i][j] = 2.0;
            c[i][j] = 0.0;
            b_transpose[j][i] = b[i][j];
        }
    }
    start_t = MPI_Wtime();
    mtype = MASTER;
    rows = (N / np) * 2;
    row_offset1 = rows;
    columns = N / 2;
    offset2 = N / 2;
    row_offset2 = 0;
```



```
/**
 * Sending for firs half of matrices. They come below the root node in the grid
 * offset 0 belongs to root
*/
        for (destination = 1; destination < np / 2; destination++) {
            MPI_Send(&row_offset1, 1, MPI_INT, destination, mtype, MPI_COMM_WORLD);
            MPI_Send(&rows, 1, MPI_INT, destination, mtype, MPI_COMM_WORLD);
            MPI_Send(&columns, 1, MPI_INT, destination, mtype, MPI_COMM_WORLD);
            MPI_Send(&a[row_offset1][0], rows * N, MPI_DOUBLE, destination, mtype, MPI_COMM_WORLD);
            MPI_Send(&b_transpose[0][0], columns * N, MPI_DOUBLE, destination, mtype, MPI_COMM_WORLD);
            row_offset1 = row_offset1 + rows;
        }

/**
 * Sending offsets for the second half of cluster. They come on the right side of the root node.
*/
        for (destination = np / 2; destination < np; destination++) {
            MPI_Send(&row_offset2, 1, MPI_INT, destination, mtype, MPI_COMM_WORLD);
            MPI_Send(&rows, 1, MPI_INT, destination, mtype, MPI_COMM_WORLD);
            MPI_Send(&offset2, 1, MPI_INT, destination, mtype, MPI_COMM_WORLD);
            MPI_Send(&columns, 1, MPI_INT, destination, mtype, MPI_COMM_WORLD);
            MPI_Send(&a[row_offset2][0], rows * N, MPI_DOUBLE, destination, mtype, MPI_COMM_WORLD);
            MPI_Send(&b_transpose[offset2][0], columns * N, MPI_DOUBLE, destination, mtype, MPI_COMM_WORLD);
            row_offset2 = row_offset2 + rows;
        }
```

- The matrices are initialized within the master/root node.
- We set offsets for columns and rows. Column size is fixed as half the matrix size. This means the matrices are split into two equal halves vertically.
- The rows are further divided based on formula **(matrix size/num processors)*2 which becomes submatrix length.**
- There are two sets of identical for loops which are used to send the relevant offsets for rows and starting position within the matrices as well to the worker nodes.
- Matrix B is transposed for convenience in streamlining a similar mode of sending/receiving for both matrices.

The same logic is repeated in slaves as well. With slaves receving the initial offsets and contents of A,B n C. At the end, each slave sends back the offsets and contents of the submtarix C using MPI_Send.

OBSERVATION
—-------------------

A general bottleneck on the cluster was observed with overall normal actions and tasks being delayed (With added network delays). Due to this metrics for large matrices could not be captured.

No more than 8 processors could be used. Kindly refer to the excel for all readings.

1. **N = 600**
   Normal execution time ~ 202 sec

Distributed multi processing peak execution time ~ 28 seconds

This is again extremely fast compared to the naive implementation. Almost 8-9 times better performance. There wasnt much performance gain even as it went from 4-8 processors.

2. **N = 900**

   Normal execution time. ~ 1020 sec
   Observed time for OpenMPI ~ 78 sec

   Speedup could be observed with more processors added to the grid. From 132 seconds with 4 processors it was reduced to 78 seconds using 8 processors. **When compared to the single threaded straight forward version the speedup is more than 12 times.**

3. **N = 1024**

   Straightforward execution time > 1800 seconds
   OpenMPI fastest time ~ 231 sec

   Using more processors, there was an increase in performance, with almost 50% increase in performance when going from 4 to 8 nodes. On comparing to its straightforward implementation it is clearly visible that it is **atleast** 7-8 times faster

4. **N = 1600**

   Straightforward implementation >2000 sec
   OpenMPI fastest time ~ 700 sec

   IBased on the trend of increasing efficiency with increase in matrix size the matrix multiplication of matrices with size = 1600 would be assured to be atleast along the lines of N= 1024. (was not able to get a proper reading for straightforward naive implementation because of cluster issues)

CONCLUSION
—-------------------------

OpenMPI provides fast means of parallel computation. Abstracting away the complexities of networking and synchronisation. There is a significant increase in performance going from 4-8 worker processors. Most effective way would be to purely load the required submatrices in the worker node, without using the master to send to each one immediately.

The impact of distributed memory multiprocessing can be observed more easily at larger matrix sizes. For example, a matrix with size around 600 seconds shows a 8-9 times better performance compared to single threaded straightforward implementation, However just as we move to a 900 sized matrix the difference becomes 12-13 times.

OpenMPI offers the best parallelism and performance compared to all the techniques we have come across till now in these assignments