```
/*****************************************************************************
        Task of Master to manage the MPI parallelization
 *****************************************************************************/
multiReturn matrixMultiMPI(matrix A, matrix B, matrix D){
  printf("\nStart parallel multiplication with MPI ... \n");
  double startMPI = MPI_Wtime();

  row = B.rows;
  col = B.cols;
  // Prepare dimensions of B and A -> store in array -> so all could be send in 1 step
  int *AB_dim_1D = malloc(3*sizeof(int *));
        AB_dim_1D[0] = row;
        AB_dim_1D[1] = col;
  // copy 2D-Array into 1D-Array for B -> needed in every process
  int *B_matrix_1D = malloc(row*col*sizeof(int *));
          for(i=0;i<row;i++){
                        for(j=0;j<col;j++){
                                B_matrix_1D[i*row+j] = B.matrix[i][j];
                        }
          }

  int slave;
  int numberOfSlaves = size-1; // Master is no slave
  int average_rows = A.rows/numberOfSlaves; // e.g. 10/4 = 2
  int extra_rows = A.rows%numberOfSlaves; // rest = 2

  startrow = 0;
  for(slave=1; slave <= numberOfSlaves; slave++){

        row = average_rows;
        if(slave <= extra_rows){row = average_rows+1;}
        AB_dim_1D[2] = row; // will also be send

        int *A_matrix_1D = malloc(row*A.cols*sizeof(int *)); // Prepare matrixparts of A for each slave
                int c=0;
                for(i=startrow;i<startrow+row;i++){
                                for(j=0;j<A.cols;j++){
                                        // copy part of 2D array into 1D array
                                        A_matrix_1D[c] = A.matrix[i][j];
                                        c++;
                                }
                }
                startrow += row; // remember position for next slave

        // Send all Data: 1. Dimensions  2. Matrix B  3. special part of A for each slave
        MPI_Send(AB_dim_1D, 3, MPI_INT, slave, MTYPE_FROM_MASTER, MPI_COMM_WORLD);
        MPI_Send(B_matrix_1D, B.rows*B.cols, MPI_INT, slave, MTYPE_FROM_MASTER, MPI_COMM_WORLD);
        MPI_Send(A_matrix_1D, row*A.cols, MPI_INT, slave, MTYPE_FROM_MASTER, MPI_COMM_WORLD);

        free(A_matrix_1D);
  }

  free(AB_dim_1D);
  free(B_matrix_1D);


  // Return from slaves
  startrow = 0;
  for(slave=1; slave <= numberOfSlaves; slave++){

        row = average_rows;
        if(slave <= extra_rows){row = average_rows+1;}

        int *D_matrix_1D = malloc(row*B.cols*sizeof(int *));
        // Receive part of result matrix D from every slave
        MPI_Recv(D_matrix_1D, row*B.cols, MPI_INT, slave, MTYPE_FROM_SLAVE, MPI_COMM_WORLD, &status);

                int c=0;
                for(i=startrow;i<startrow+row;i++){
                                for(j=0;j<A.cols;j++){
                                        // copy 1D array to the right pos. in the 2D array D
                                        D.matrix[i][j] = D_matrix_1D[c];
```

```
                                        c++;
                                }

                }
                startrow += row; // remember the position

                free(D_matrix_1D);

  }
  double endMPI = MPI_Wtime();
  printf("finished\n\n");

  multiReturn mr = {"MPI",endMPI-startMPI};

  return mr;
}

/**************************************************************************************
        TASK OF SLAVES
 **************************************************************************************/
void beMySlave(){

        int* AB_dim_1D = malloc(3*sizeof(int *));
        // 1. receive the dimensions of the matrices
        MPI_Recv(AB_dim_1D, 3, MPI_INT, MASTER, MTYPE_FROM_MASTER, MPI_COMM_WORLD, &status);

        int B_rows = AB_dim_1D[0];
        int B_cols = AB_dim_1D[1];
        int A_rows = AB_dim_1D[2];

        // allocate memory depending on first recv-step
        int *B_matrix_1D = malloc(B_rows*B_cols*sizeof(int *));
        // 2. receive the whole B matrix
        MPI_Recv(B_matrix_1D, B_rows*B_cols, MPI_INT, MASTER, MTYPE_FROM_MASTER, MPI_COMM_WORLD,
&status);

        int *A_matrix_1D = malloc(A_rows*B_rows*sizeof(int *));
        // 3. receive a special part of A matrix for calculations
        MPI_Recv(A_matrix_1D, A_rows*B_rows, MPI_INT, MASTER, MTYPE_FROM_MASTER, MPI_COMM_WORLD,
&status);

        // prepare the D matrix, which contains the result data
        int *D_matrix_1D = malloc(A_rows*B_cols*sizeof(int *));
        int i,j,k;
        for(i=0;i<A_rows;i++){
                for(j=0;j<B_cols;j++){
                        int tempSum = 0;
                        for(k=0;k<B_rows;k++){
                                // simulate 2D matrix multiplication with 2 1D matrices
                                tempSum += A_matrix_1D[i*B_rows+k] * B_matrix_1D[k*B_cols+j];

                        }
                        D_matrix_1D[i*B_cols+j] = tempSum;
                }
        }
        // after finishing work, slaves send back their results to master
        MPI_Send(D_matrix_1D, A_rows*B_cols, MPI_INT, MASTER, MTYPE_FROM_SLAVE, MPI_COMM_WORLD);

        free(AB_dim_1D);
        free(A_matrix_1D);
        free(B_matrix_1D);
        free(D_matrix_1D);
}
```