# Assignment 1: Parallel Programming with MPI
# A Distributed Data Structure

CPDS Parallelism

Nicolás Zhao
cpds1104
nicolas.zhao@estudiantat.upc.edu

*Universitat Politècnica de Catalunya (UPC)*

December 6, 2023

# Section 1.1

## Exercise 1

Write the solution for 12 statements, identified as S1 ... S12 in the source codes, that have some missing parameters in some calls to MPI primitives. Indicate the identifier of the statement and how you have filled it in.

```
//S1
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
//S2
MPI_Comm_size(MPI_COMM_WORLD, &size);
//S3
MPI_Recv(xlocal[0], maxn , MPI_DOUBLE , rank - 1, 0, MPI_COMM_WORLD, &status);
//S4
MPI_Send( xlocal[1], maxn, MPI_DOUBLE, rank - 1, 1, MPI_COMM_WORLD);
//S5
MPI_Recv(xlocal[maxn/size+1], maxn, MPI_DOUBLE, rank + 1, 1, MPI_COMM_WORLD, &
    status);
//S6
MPI_Reduce( &errcnt, &toterr, 1 , MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
//S7
MPI_Isend(xlocal[maxn/size], maxn, MPI_DOUBLE, rank + 1, 0, MPI_COMM_WORLD, &r[
    nreq++] );
//S8
MPI_Irecv(xlocal[0], maxn, MPI_DOUBLE, rank - 1, 0, MPI_COMM_WORLD, &r[nreq++] );
//S9
MPI_Isend(xlocal[1], maxn, MPI_DOUBLE, rank - 1, 1, MPI_COMM_WORLD, &r[nreq++] );
//S10
MPI_Irecv(xlocal[maxn/size+1], maxn, MPI_DOUBLE, rank + 1, 1, MPI_COMM_WORLD, &r[
    nreq++]);
//S11
MPI_Reduce(&errcnt, &toterr, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
//S12
 MPI_Sendrecv(xlocal[1],maxn , MPI_DOUBLE  , prev_nbr, 1, xlocal[maxn/size+1],
     maxn, MPI_DOUBLE, next_nbr, 1, MPI_COMM_WORLD, &status );
```

# Section 1.2

## Exercise 1

Why do we need to use `MPI_Allreduce` instead of `MPI Reduce` in all the codes in section 1.2?

If we use `MPI_Reduce` instead we will have different values of the variable gdiffnorm between the MPI processes. One process will have computed the most recent gdiffnorm but because it didnt notify to the other processes about the most recent value of gdiffnorm , each process will have a different gdiffnorm, which will still make the program compile correctly, however, the semantics would be broken. We want gdiffnorm to be global, so we should use `MPI_Allreduce` so all processes have the same value.

## Exercise 2

Alternatively, which other MPI call would be required if we had used MPI Reduce in all the codes in section 1.2?

`MPI_Bcast` so that each process is synchronized with the most recent version of gdiffnorm, which is the value we are using for testing the convergence of the computation.

## Exercise 3

| We have said that we have a halo so that some of the values in the matrix are read but not written during the computation. Inspect the code and explain which part of the code is responsible for defining such halo. |
|---|

Since we are computing the value of each position of the matrix with the average of its left, right, top and bottom neighbours values, it would not make sense to compute the values of the first and last rows and columns. So we have the following *for* loops inside the *do while* loop

```
do{
    ...
        for (i=i_first; i<=i_last; i++)
        {
            for (j=1; j<maxn-1; j++)
            {
                xnew[i][j] = (xlocal[i][j+1] + xlocal[i][j-1] +
                              xlocal[i+1][j] + xlocal[i-1][j]) / 4.0;
                diffnorm += (xnew[i][j] - xlocal[i][j]) *
                              (xnew[i][j] - xlocal[i][j]);
            }
        }
        for (i=i_first; i<=i_last; i++)
        {
            for (j=1; j<maxn-1; j++)
            {
                xlocal[i][j] = xnew[i][j];
            }
        }
    ...
} while(...)
```

## Exercise 4

| Explain with your own words how the *load is balanced* across the different processes when the number of rows is not evenly divided by P. |
|---|

The code utilizes the `int getRowCount(int rowsTotal, int mpiRank, int mpiSize)` function to determine the number of rows allocated to each MPI process. If the division of rows is even, an equal number of rows is assigned to each process. However, if the division results in a remainder, an additional row is assigned to some processes. Let's delve into the function: Initially, it computes the row count using `rowsTotal / mpiSize`. However, in C, integer division truncates the fractional part, so 3/2 equals 1 instead of 1.5. So we add an additional row to processes with ranks less than the remainder of `rowsTotal % mpiSize`. Lastly, given that process ranks start from 0 in C, we employ the strictly greater operator (>) in `rowsTotal % mpiSize > mpiRank` to determine which processes receive the extra load.

## Exercise 5

Write the solution for the statements identified as S13 ... S24 in the source codes, which have errors or some missing parameters in some calls to MPI primitives.

```
//S13
if (next_nbr >= size) next_nbr = MPI_PROC_NULL;

//S14
MPI_Allreduce( &diffnorm, &gdiffnorm, 1,MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

//S15
MPI_Gather( xlocal[1], maxn * (maxn/size), MPI_DOUBLE, x, maxn * (maxn/size),
    MPI_DOUBLE, 0, MPI_COMM_WORLD );

//S16
MPI_Wait( &r[2], &status );

//S17
MPI_Waitall( nreq, r , statuses);

//S18
MPI_Wait( &r[3], &status );

//S19
MPI_Iallreduce(&diffnorm, &gdiffnorm, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD, &r
    [0] );

//S20
MPI_Igather(xlocal[1], maxn * (maxn/size),MPI_DOUBLE, x, maxn * (maxn/size),
    MPI_DOUBLE, 0, MPI_COMM_WORLD, &r[0] );

//S21
return (rowsTotal / mpiSize) + (rowsTotal % mpiSize > mpiRank);

//S22
nrows  = getRowCount(maxn, rank, size);

//S23
MPI_Gather(&lcnt, 1, MPI_INT, recvcnts, 1, MPI_INT, 0, MPI_COMM_WORLD);

//S24
MPI_Gatherv( xlocal[1], lcnt, MPI_DOUBLE, x, recvcnts, displs, MPI_DOUBLE,0,
    MPI_COMM_WORLD);
```