Distributed Memory Programming with MPI
Programming Assignment – 3

Exercise – 1 [4 points]

In this task you will play with different modes of point-to-point communication. The computational problem to be solved is the following:

Matrix $A \in R^{n \times n}$ is partitioned and distributed over the four processors p_0, p_1, p_2, p_3 as follows:

$$A = \begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} p_0 & p_1 \\ p_2 & p_3 \end{pmatrix}, \quad \text{with} \quad A_{TL} \in \Re^{\frac{n}{2} \times \frac{n}{2}}$$

Matrices $B \in R^{n \times n}$ and $C \in R^{n \times n}$ are defined as

$$B = \begin{pmatrix} B_{TL} & B_{TR} \\ B_{BL} & B_{BR} \end{pmatrix} := \begin{pmatrix} A_{TL} + A_{TR} & A_{TL} - A_{TR} \\ A_{BL} + A_{BR} & A_{BL} - A_{BR} \end{pmatrix}$$

and

$$C = \begin{pmatrix} C_{TL} & C_{TR} \\ C_{BL} & C_{BR} \end{pmatrix} := \begin{pmatrix} B_{TL} + B_{BL} & B_{TR} + B_{BR} \\ B_{TL} - B_{BL} & B_{TR} - B_{BR} \end{pmatrix}$$

Each processor has enough local memory to store explicitly two matrices of size $\frac{n}{2} \times \frac{n}{2}$.

Your task is to write a program that will compute the distributed matrix C. The program should take one command-line argument n, i.e. the size of the matrix (you can assume it is a multiple of 2). Each of the 4 processes initializes its own $\frac{n}{2} \times \frac{n}{2}$ chunk of the matrix A, and performs the necessary communication and computation. You may need to explicitly synchronize the processes with `MPI_Barrier`. If you do, minimize the number of barriers used. You can initialize the matrix in any way you want, however, we suggest to do it in a way that you can easily check the result.

Implement multiple versions of the program, each time using one of the following communication codes :

a) 2-sided communication based on `MPI_Send` and `MPI_Recv`;

b) 2-sided communication based on `MPI_Isend` and `MPI_Recv`;

c) 2-sided communication based on `MPI_Send` and `MPI_Irecv`;

Exercise – 2 [3 points]

The goal of this task is to optimize for memory usage. We ask you to solve thr same problem as in the above task, this time with restriction in the amount of memory that is available per process. Now, each process can only store one submatrix of size $\frac{n}{2} \times \frac{n}{2}$ plus one vector of size $\frac{n}{2}$. Implement one single version using the communication mode of your choice.

Exercise – 3 [3 points] [MPI implementation]

In this task you will parallelize a given sequential code that solves a steady-state heat conduction problem over a thin square plate. Concretely, the code simulates the diffusion of heat on a plate to determine at which temperature it stabilizes. The initial temperature of the surface is 50 degrees, and a constant temperature is applied at the boundaries. Specifically, the left, top and right boundaries are kept constant at 100 degrees while the bottom boundary is kept at 0 degrees.

The code relies on the iterative computation of the Poisson equation $\Delta\phi = f$ using finite differences. The surface is discretized in a grid of $n \times n$ grid points. At each iteration, the interior grid points $\phi_{i,j}$ are updated following the rule:

$$\phi(i,j) = \frac{\phi(i-1,j) + \phi(i,j-1) + \phi(i+1,j) + \phi(i,j+1)}{4}$$

The sequential program provided (*poisson_seq.c*) represents the discretized grid as a matrix of size $n \times n$. The matrix is initialized according to the conditions described above, and then the program performs an indefinite number of timesteps (iterations). At each iteration, the program estimates the new temperature at timestep t from that of the previous (old) iteration $t-1$. When the difference between every pair $(\phi_{old_{i,j}}, \phi_{new_{i,j}})$ from two consecutive iterations is below a given threshold, the program has found the steady state and it terminates.

You have to parallelize the given code using the MPI library. Your program will partition the matrices ϕ_{old} and ϕ_{new} by blocks of rows, where each of p processes will work on $\frac{n}{p}$ contiguous rows. Figure 1 illustrates an example with $n = 11$ and $p = 3$. In the example, the blue pattern (known as stencil) represents the update rule for each point in the grid: to compute the new value of each point, we need the old value of the corresponding four neighbours. Notice that the stencil is not applied to the gray points, since their values are constant. Also, when applying the stencil to the red points, each process needs data "belonging" to its neighbor(s).

In your program, each process will initialize its own portion of the grid. Then, at each iteration, the processes will compute the new values of their subgrid, performing the necessary communication. When the computation converges, the processes will terminate and the program will finish.

We ask you to generate two versions of the parallel program using:

a) Blocking communications

b) Non-blocking communication, where communication and computation overlap.

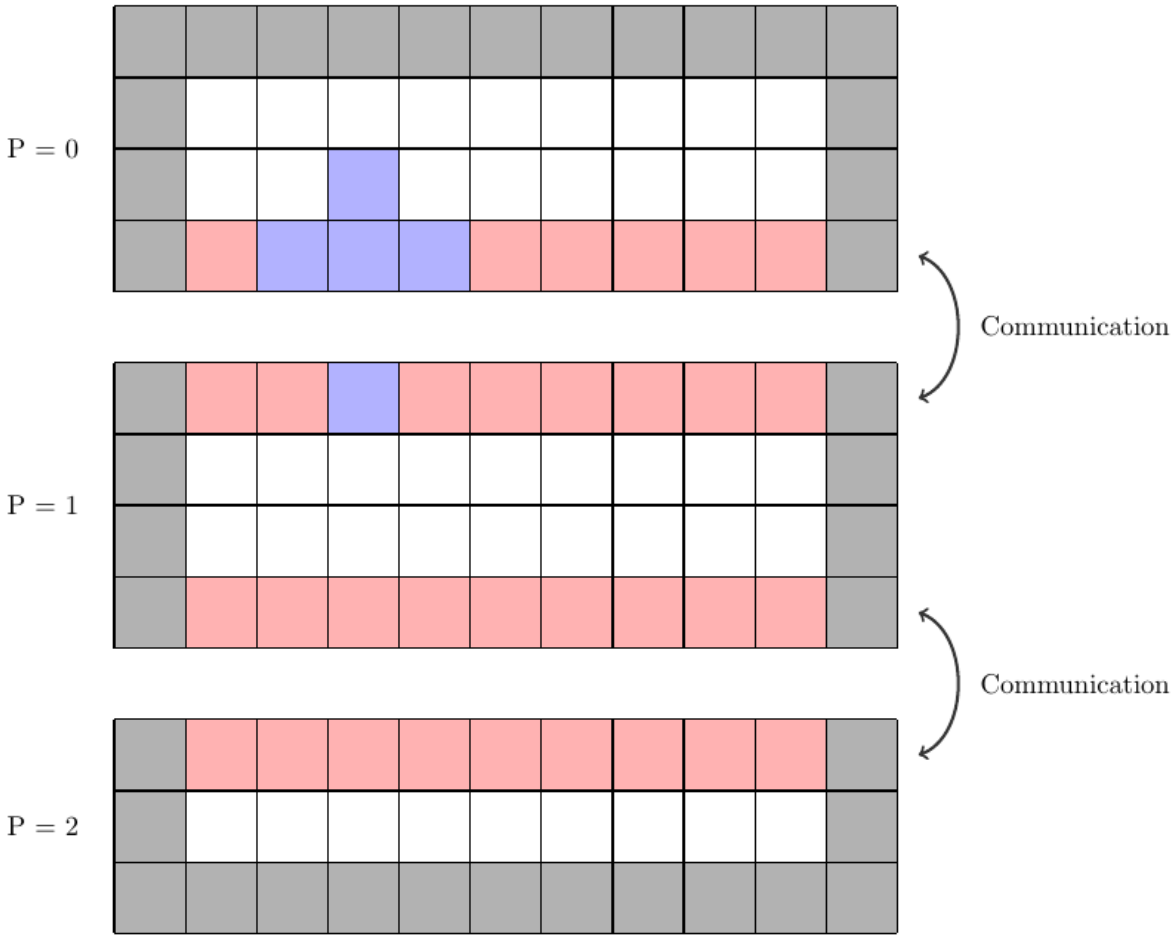

Figure 1: Data partitioning and communication for a matrix of size 11×11 and 3 processes.