Parallel Programming WS15 **HOMEWORK #6**

Submit by email to: `fabregat@aices.rwth-aachen.de`
Due by: Friday, December 4th, 5pm.

MPI – Part 4

1. In this task we ask you to sketch another version of *Homework 5, exercise 1* using one-sided communication. Notice that we are asking for a sketch (algorithm) and not actual C code. To solve this task, the following instructions are available:

   - `Y = fetch(var_name, j, label);`

     `fetch` gets the content of variable `var_name`, local to processor p_j, and stores it onto the local variable `Y`. `label` is a numeric label –local to p_i– that uniquely identifies the data transfer. The instruction is *one-sided* and *asynchronous*: p_i executes the next instruction in the program without waiting for `fetch` to complete; p_j does not participate in the data exchange.
   - `wait(label);`

     p_i idles until the data transfer identified by `label` is completed.
   - `barrier();`

     p_i idles until every processor issues a `barrier()`.

2. 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[1] $\phi_{i,j}$ 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 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.

[1] Remember that the boundaries are kept constant at a temperature of either 0 or 100 degrees.

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 neighbors. 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:

1) Blocking communication.
2) Non-blocking communication, where communication and computation overlap.

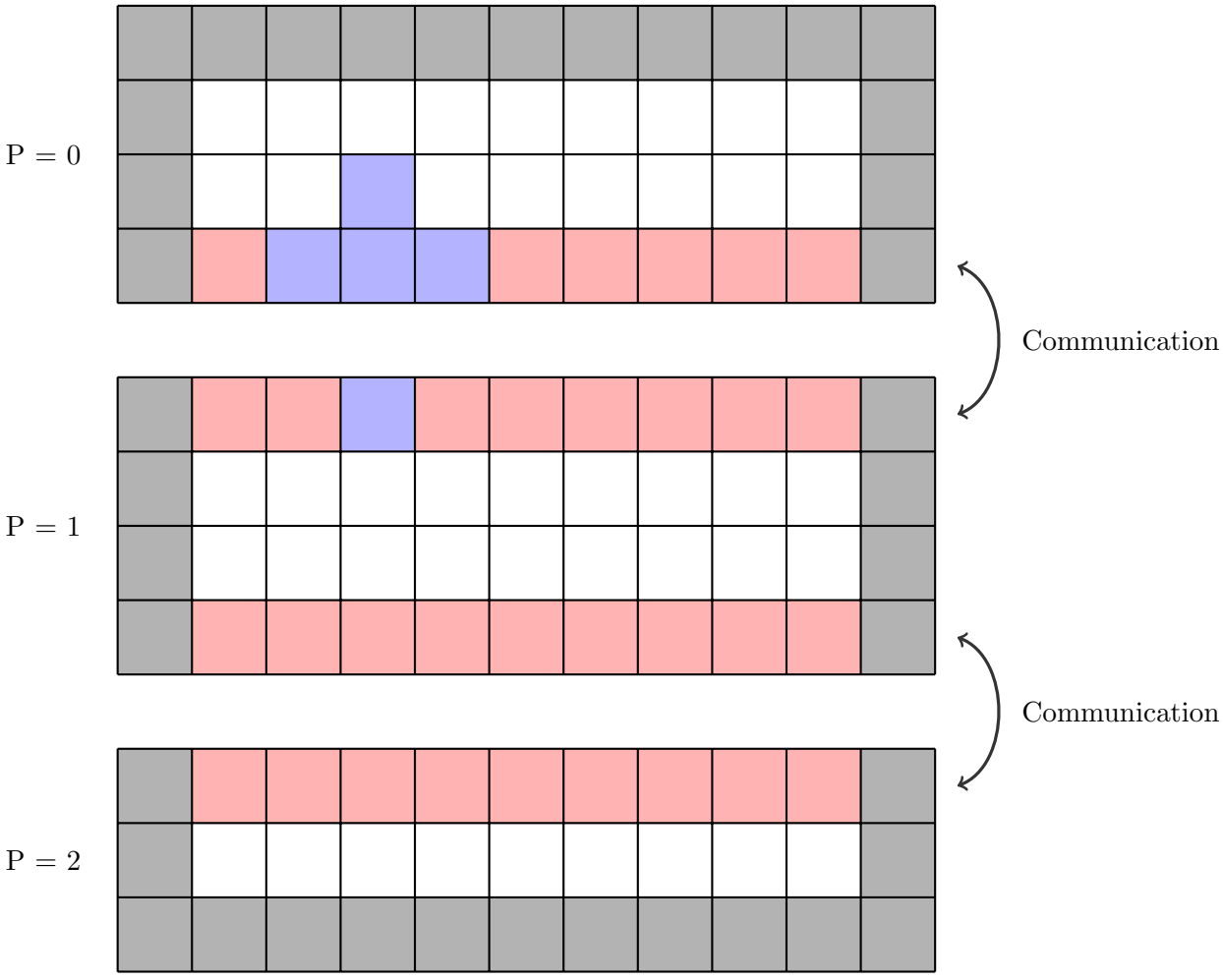

Figure 1: Data partitioning and communication for a matrix of size 11×11 and 3 processes.