MPI: Practical session 6

1 Laplace's equation

In this example, we will be solving Laplace's equation,

$$\nabla^2 x = 0,$$

on a square grid, with x fixed on the top and bottom.

We will be using the numerical scheme

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

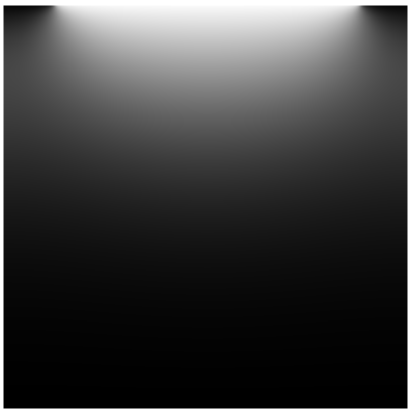
The serial code is found in file `laplace.cpp`. (The boundary condition at the bottom is $x = 0$, and at the top, it is 1 in the central three quarters of the domain, and 0 near the edges.) The code defines two grids, `g1` and `g2`, which hold the previous and current cell values, respectively, and iterates a fixed number of times. (Note that this is not good practice, in general! We should have checked for convergence before stopping).

The top and bottom rows of the domain remain unchanged, and special treatment is needed for the left and right boundaries; the code updates them using an average of the three remaining neighbouring values.

The result is output to a `.pam` file, as in the Mandelbrot example, which can be opened using

```
$ open <file>
```

Running the code, we obtain the following image



1.1 Blocking point-to-point communication

As a first attempt to parallelise the Laplace's solver, divide the grid horizontally into bands, one for each process.

1.1.1 `MPI_Send` and `MPI_Recv`

In the first instance, use calls to the blocking point-to-point communication functions `MPI_Send` and `Recv` to obtain the values at the boundary of each process' (ghost cells), from the correct neighbour.

At the end of the simulation, send the results back to rank 0 in order, so that it writes the output file.

1.1.2 MPI_SendRecv

When exchanging boundary conditions between processes, replace the multiple calls to `MPI_Send` and `MPI_Recv` with just two calls to `MPI_Sendrecv`.

1.1.3 Buffered sends

The issue with the previous approach is that it is not entirely parallel. Because of the use of blocking communication, the boundary exchange procedure happens in order. For example, process `nproc - 1` will only receive its top boundary row when all other processes have, irrespective of when `nproc - 2` had the updated values ready.

Modify your code from Section 1.1.1, so that it now uses buffered sends, rather than standard sends. Remember to use `MPI_Buffer_Attach` to define the buffer space.

1.2 Non-blocking point-to-point communication

The extra buffer required in the buffered sending mode uses time and space, which may come to be a problem when sending large amounts of data. Modify your code to use the non-blocking point-to-point sending routine, `MPI_Isend`. Make sure to use completion testing routines, where needed.

Compare the performance of the different versions of your code for different numbers of processes. Try also varying the size of the domain and repeating this exercise.