1 Introduction

The aim of this exercise is to write a complete MPI parallel program that does a simple simulation of water percolating through porous rock. We will start with a serial code that performs the required calculation but is also designed to make subsequent parallelisation as straightforward as possible. The rock is represented by a large two-dimensional array, so the natural parallel approach is to use regular domain decomposition. As the algorithm we will use involves nearest-neighbour interactions between grid points, this will require boundary swapping between neighbouring processes and adding halos to the arrays. The exercise also utilises parallel scatter and gather operations. For simplicity, we will use a one-dimensional process grid (i.e. decompose the problem into slices).

The solution can easily be coded using either C or Fortran. The only subtlety is that the natural direction for the parallel decomposition of the arrays, whether to slice up the 2D grid over the first or second dimension, is different for the two languages. A sample serial solution, including a function to write the result to an image file, can be found in `MPP-percolate.tar` on the MPP course web pages.

If you are following the *Programming Skills* (PS) course then you will be familiar with the problem and the algorithm for solving it. If you want, you can skip to Section 4 to read about the provided serial code. Note that this serial code gives *exactly the same answer* as that supplied in the PS course but implements the algorithm in a slightly different way and is written differently, both to make parallelisation simpler.

2 The Percolation Problem

The program solves the following problem. Suppose we have a grid of squares (e.g. 5×5) in which a certain number of squares are "filled" (grey) and the rest are "empty" (white). The problem to be solved is whether there is a path from any empty (white) square on the top row to any empty square on the bottom row, following only empty squares – this is similar to solving a maze. The connected empty squares form clusters: if a cluster touches both the top and bottom of the grid, i.e. there is a path of empty squares from top to bottom through the cluster, then we say that the cluster *percolates*.

See Figure 1 for an example of a grid that has a cluster which percolates. See Figure 2 for an example of a grid with no cluster that percolates.

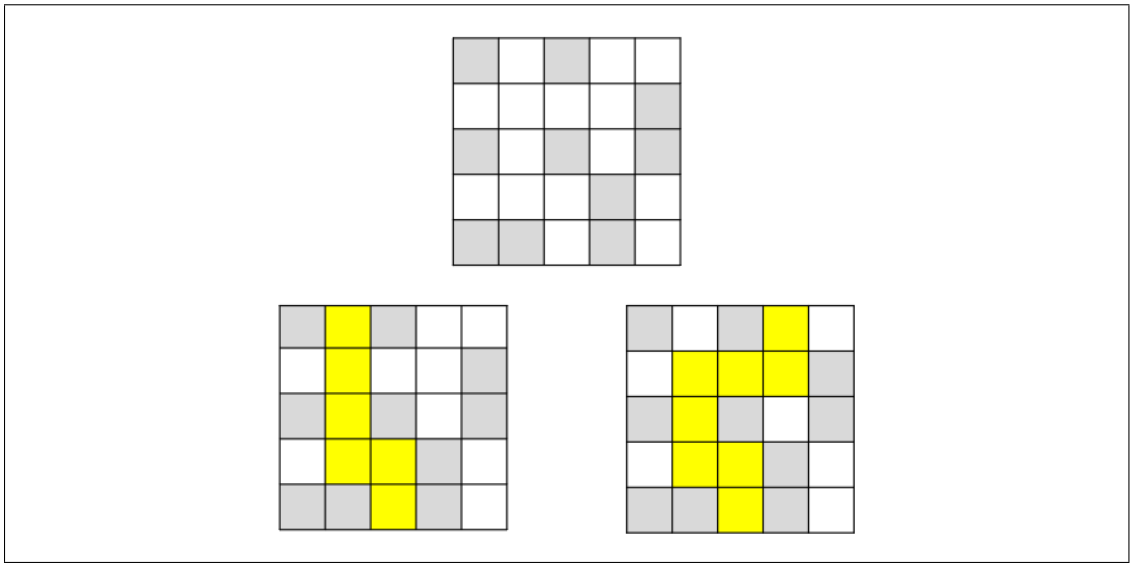

Figure 1: The grid at the top has a cluster which has paths through empty squares from top to bottom. Two example paths are highlighted in yellow. This grid *has a cluster that percolates*.

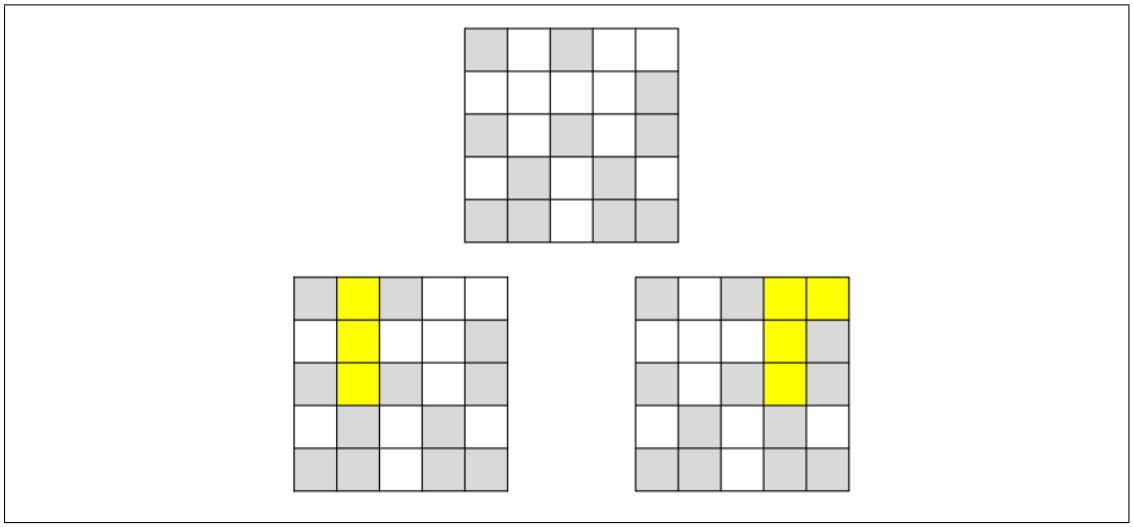

Figure 2: The grid at the top has no cluster with a path through empty squares from top to bottom. Two examples of the longest paths are highlighted in yellow. This grid *does not have a cluster that percolates*.

3 Cluster Identification Algorithm

The algorithm identifies all the connected clusters – the largest clusters are the ones of interest for percolation. In the grid of Figure 1 the largest cluster is 13 squares, and the second largest cluster is 2 squares. In the grid of Figure 2 the largest cluster is 9 squares and the second largest cluster is 2 squares.

Now, consider an $L \times L$ grid where we vary the number of filled squares. Will we get percolation if the grid is filled with a given density ρ (Greek letter, pronounced "rho") where $0.0 \leq \rho \leq 1.0$, i.e. where the density ranges from empty to completely solid rock? In the grid of Figure 1, $L = 5$ and $\rho = 0.40$ (10/25, as 10 squares are filled). In the grid of Figure 2, $L = 5$ and $\rho = 0.48$ (12/25, as 12 squares are filled). For a given ρ there are many possible grids so we generate a random grid and see if a cluster percolates. To check the reliability of our answer we would re-run many times with different grids, i.e. use different random number seeds.

The simplest cases are $\rho = 0.0$ (all squares are empty) when we always get percolation and $\rho = 1.0$ (solid rock where all squares are filled) when we never get percolation. But what about $\rho = 0.5$? Or $\rho = 0.4$ or 0.6? What is the critical value of ρ which separates percolating rocks from non-percolating rocks? Does the answer depend on the size of the simulation, i.e. the value of L?

We can answer all these questions using computer simulation.

First, we initialise the grid:

- declare an integer array of size $L \times L$;
- each of the white squares (which represent empty space) are set to a *unique* positive integer;
- all of the filled squares, representing solid rock, are set to zero.

Then, we loop over all the non-zero squares in the grid for many steps and successively *replace each square with the maximum of its four neighbours*. The largest positive numbers gradually fill the gaps so that each cluster eventually contains a single, unique number (the filled squares are automatically ignored as they are set to zero). If we monitor how many values change at each step then we can tell when we have identified all the clusters as nothing will change between successive steps.

We can then count and identify the clusters. If we find a part of the same cluster on both the top and bottom row of the grid then we can conclude that the cluster percolates – see Figure 3.

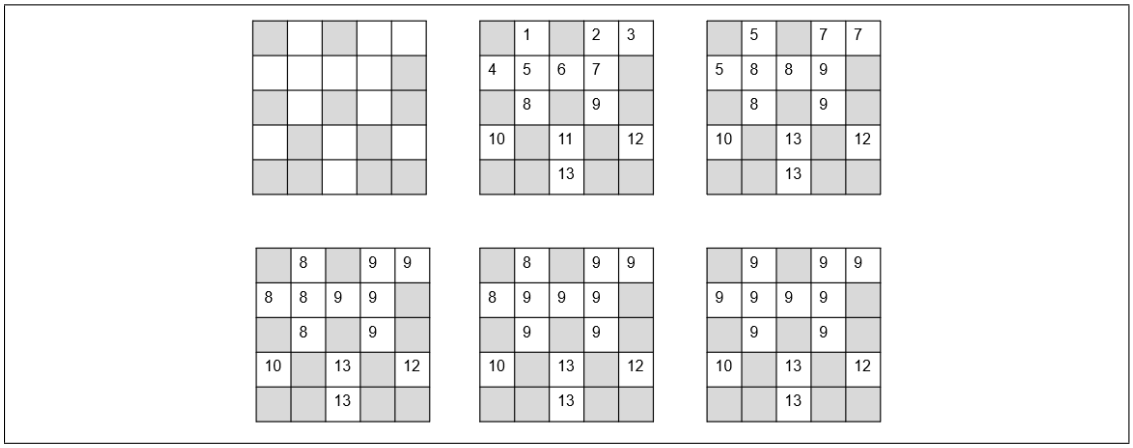

Figure 3: Solving percolate in five steps after initialisation. No squares would be updated on a subsequent step. All the filled (grey) squares can be assumed to be zero.

Our solution involves replacing each square with the maximum of its four neighbours, but what about squares at the edges of the grid which have no neigbours? We do not want to write extra code to handle these as a special case, so a common solution to this problem is to define a "halo" around the grid: we surround our grid with a layer of solid rock (to stop the water leaking out!).

For an $L \times L$ percolation problem we therefore use an $(L+2) \times (L+2)$ grid and set the halos to be filled (i.e. zero). Zeros block cluster growth and they will never propagate to other squares – see Figure 4.

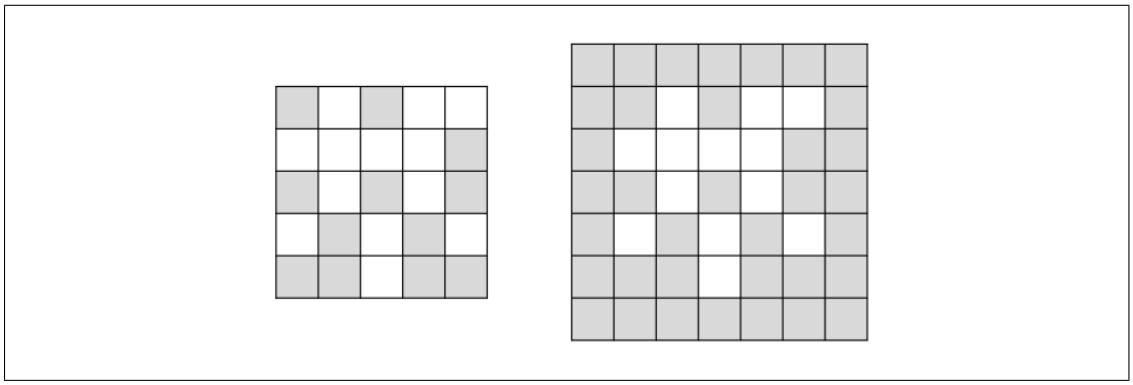

Figure 4: To process a 5×5 grid we add a halo of filled squares round the border of the grid to produce a 7×7 grid. See Figure 5 for the example of Figure 3 but with the use of a halo.

4 Serial Code

Download the serial code `MPP-percolate.tar` from the MPP course web pages. It should compile and run as supplied.

The default parameters are:

- system size $L = 288$;
- rock density $\rho = 0.411$.

It takes a single command-line parameter – the random number seed. To reproduce the results shown here, and to get the same answer as the PS code, use $seed = 1564$.

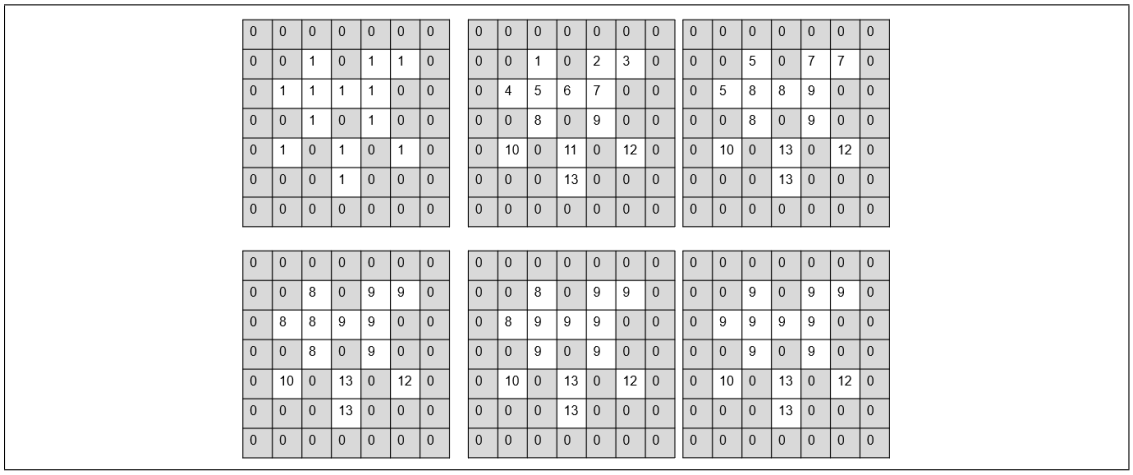

Figure 5: Solving percolate using a halo

On Cirrus:

```
[user@cirrus]$ tar -xvf MPP-percolate.tar
...
[user@cirrus]$ cd percolate/C-SER

[user@cirrus]$ make
icc -O3 -c percolate.c
icc -O3 -c percwrite.c
icc -O3 -c uni.c
icc -O3 -o percolate percolate.o percwrite.o uni.o

[user@cirrus]$ ./percolate 1564
percolate: params are L = 288, rho = 0.411000, seed = 1564
percolate: rho = 0.411000, actual density = 0.408119
percolate: number of changes on step 100 is 13128
percolate: number of changes on step 200 is 11127
percolate: number of changes on step 300 is 9173
percolate: number of changes on step 400 is 5439
percolate: number of changes on step 500 is 3497
percolate: number of changes on step 600 is 1645
percolate: number of changes on step 700 is 723
percolate: number of changes on step 800 is 133
percolate: number of changes on step 900 is 1
percolate: number of changes on step 1000 is 0
percolate: number of changes on step 1100 is 0
...
percolate: number of changes on step 4600 is 0
percolate: cluster DOES NOT percolate
percwrite: only visualising the largest cluster
percwrite: maximum cluster size is 19661
percwrite: opening file <map.pgm>
percwrite: writing data ...
percwrite: ... done
percwrite: file closed

[user@cirrus]$ display map.pgm
```