NATURAL ENVIRONMENT RESEARCH COUNCIL

EPSRC

Percolate exercise

epcc

Overview

- The purpose of this practical is to
  - gain experience of taking a real scientific problem and implementing it as a program in Fortran
  - Try out the object oriented features in F2003 functionality we will discuss in the lectures.

- Aim of this practical
  - Develop the Percolate code, to fill in missing code to complete the implementation.
  - Implement a module structure in the percolate code
  - Define derived types for use in the program
  - Implement Fortran classes
  - Compare performance between the different versions

Getting the Code

- Copy code from the ARCHER website:
  - http://www.archer.ac.uk/training/course-material/2016/01/oofortran_culham/index.php
  - Download to ARCHER
  - Unpack the code
    - tar xvf percolate.tar.gz
- Initially
  - the code will compile and run ...
  - ... but do nothing useful until you complete the missing code

```
$ ./build
$ ./percolate
```

The Output

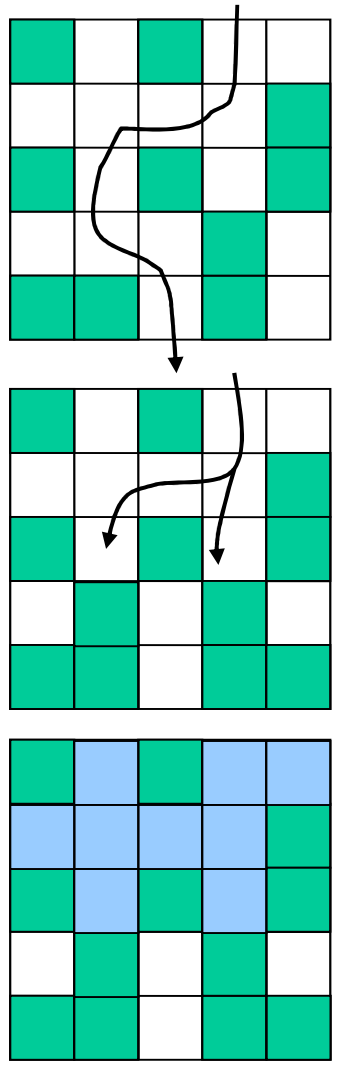
- The program produces a visual output
  - as a Portable Grey Map (PGM) file
  - which can be viewed with standard tools
- To visualise

```
$ display -resize 200x200 map.pgm
```

- Initially this will be a meaningless chequerboard
  - because you haven't written the code yet
    - (and `percwrite` will incorrectly report number and size of clusters)

The Percolation Problem

- Take a grid of squares (eg 5x5)
  - fill a certain fraction of them (green)
  - leave the rest empty (white)
- Is there a path from top to bottom?
  - like solving a maze
- Connected spaces form clusters
  - if a cluster touches both top and bottom we say it *percolates*
  - might also be interested in the sizes of the largest clusters
  - here they are 13 & 9 (second largest 2 & 2)



Mathematical Formulation

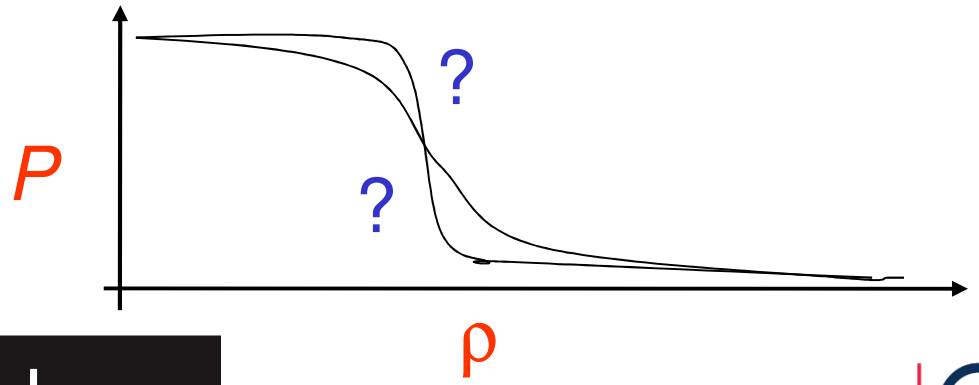
- Consider an *L*x*L* map
  - what is the probability *P* of percolation if the grid is filled with a given density ρ
  - value of ρ lies between 0.0 (empty) and 1.0 (100% filled)

- On previous slide
  - L=5, ρ = 0.40 and 0.48 (10/25 and 12/25)

- For a given ρ there are many possible maps
  - generate different random maps with the correct density
  - compute the probability that a cluster percolates

What Do We Know Already?

- Obviously
  - if $\rho = 0.0$, $P = 1.0$; if $\rho = 1.0$, $P = 0.0$
- What about $\rho = 0.5$?
  - need to do many computer *simulations*
  - if we generate 100 maps and 72 of them percolate, $P = 0.72$
- What does the graph of P versus ρ look like?
  - how does it depend on L?



The Programming Exercise

- You have to complete four routines

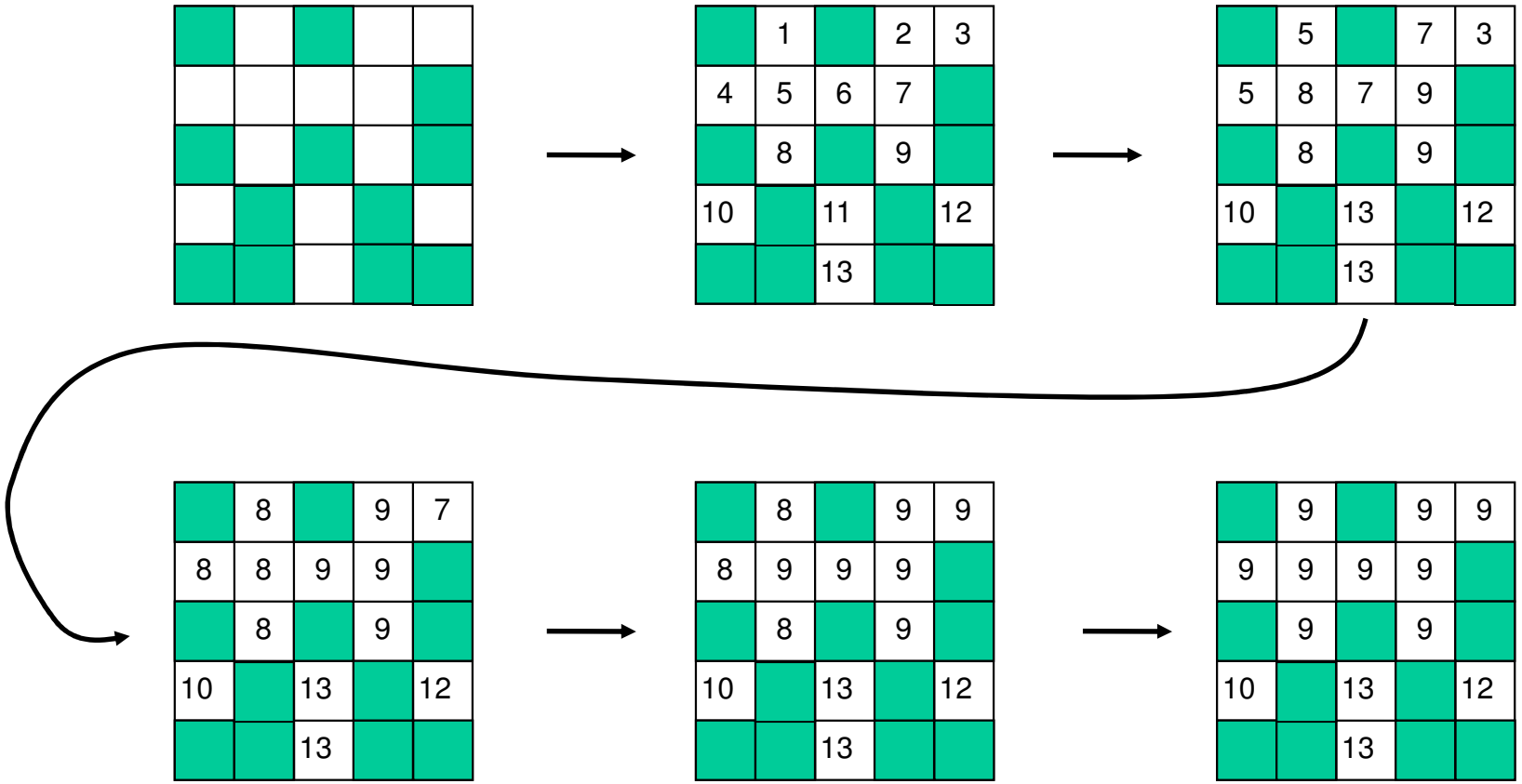
- **`fill`**
  - to fill the *L*x*L* map with ones and zeros with the correct density
- **`init`**
  - to initialise the map for the `update` routine
- **`update`**
  - to look for clusters until they are all found
- **`test`**
  - to check if percolation has occurred

Method

- Initialise each of the empty cells with a unique positive integer
  - loop over all the cells in the map many times
  - replace each cell with the maximum of its four neighbours
    - in all cases, ignore the filled cells

- The large numbers gradually fill the gaps
  - each cluster eventually contains a single, unique number
  - this allows us to count and identify the clusters
  - and look for percolation
    - if the same number appears at top and bottom of the map

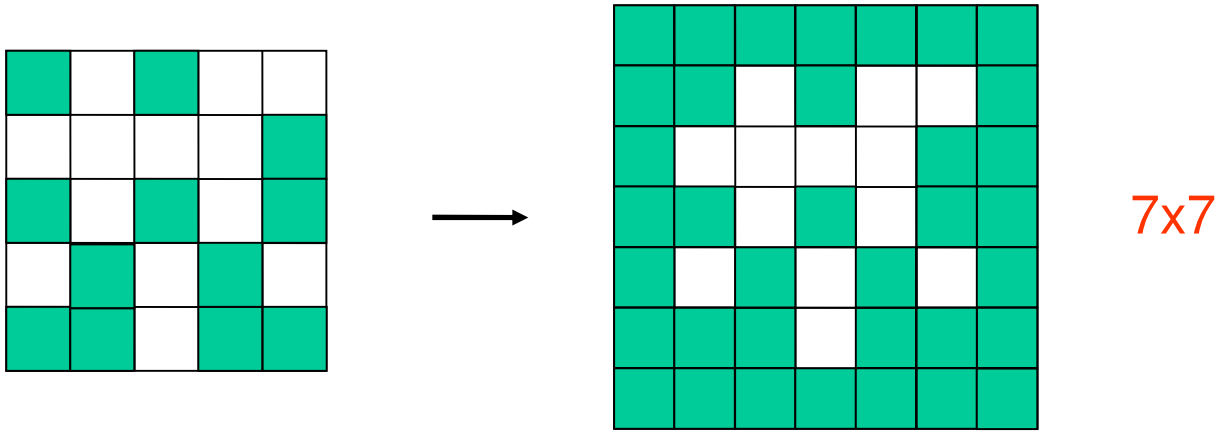
Solving Test Problem



Boundaries

- Must ensure we don't look outside the map
  - easiest to achieve by having a halo all round the problem
  - use a map of size (L+2)x(L+2) and fill in the halo

5x5



7x7

- Must ignore filled cells when updating map
  - set filled (green) cells to zero
  - zeroes will NEVER propagate to other cells

Filling the Map

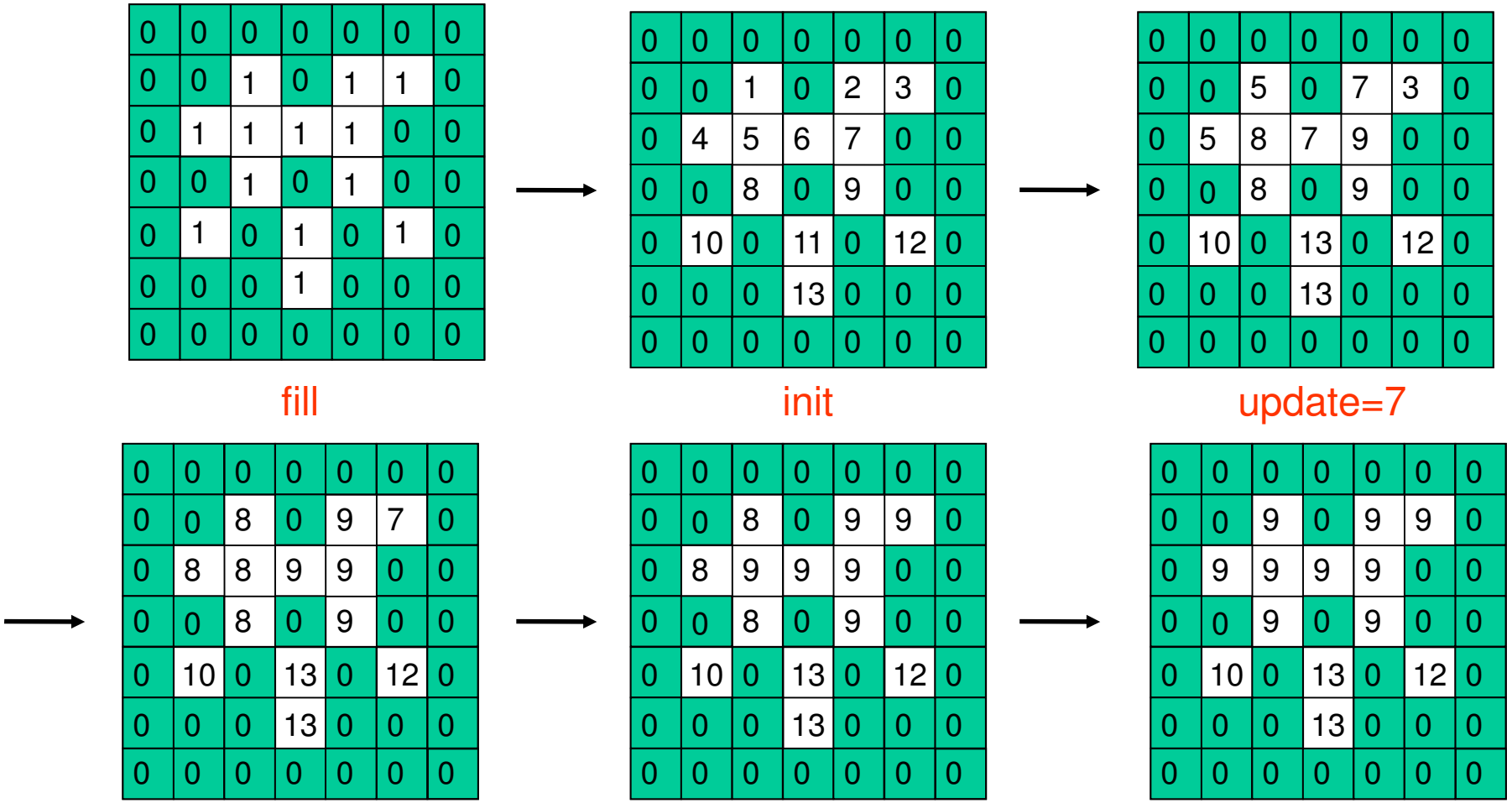
- `fill` should fill the map with probability ρ
  - eg if ρ = 0.6, 60% of the interior cells are zero, the rest are one
  - check against the value of a random number 0.0 <= *r* < 1.0
    - eg for ρ = 0.6, decide based on whether *r* is less / greater than 0.6
    - see random number generator in `uni.f90`
    - compute actual density = number of zeroes / (L*L), compare to ρ
  - always set the boundaries to zero

- Implement the routine
  - run the code and visualise the output for different values of ρ
  - empty cells (the clusters) will appear black
  - only the interior is displayed

Looking for Clusters

- `init` sets non-zero cells to unique integers
- `update` looks at every non-zero interior cell
  - replaces it with the maximum of its four nearest neighbours
    - and returns the number of cells that have changed
  - main program stops updating when there are no more changes
- `test` checks if any clusters have percolated
  - returns true if percolated; false otherwise
- `write` also checks some statistics
  - prints the number of clusters and the size of the largest
  - you can also control how many appear in output
    - display only the largest two or three clusters for nicer pictures!

The Whole Program



Summary

- Work on the percolation example
  - Initially just developing an “old school” Fortran program
- Aim is:
  - to obtain a program that produces useful output
  - to obtain graph of P vs ρ for various L
- Come back to this example for the language features we are looking at in the lectures
  - Add modules
  - Add derived types
  - Add classes
  - etc…

Measuring performance

- Plot performance (eg iterations/second) vs *L*
  - `dtime()` or `etime()` in Fortran
  - Compare the different implementations to see if there are performance penalties to the different approaches