MPP coursework

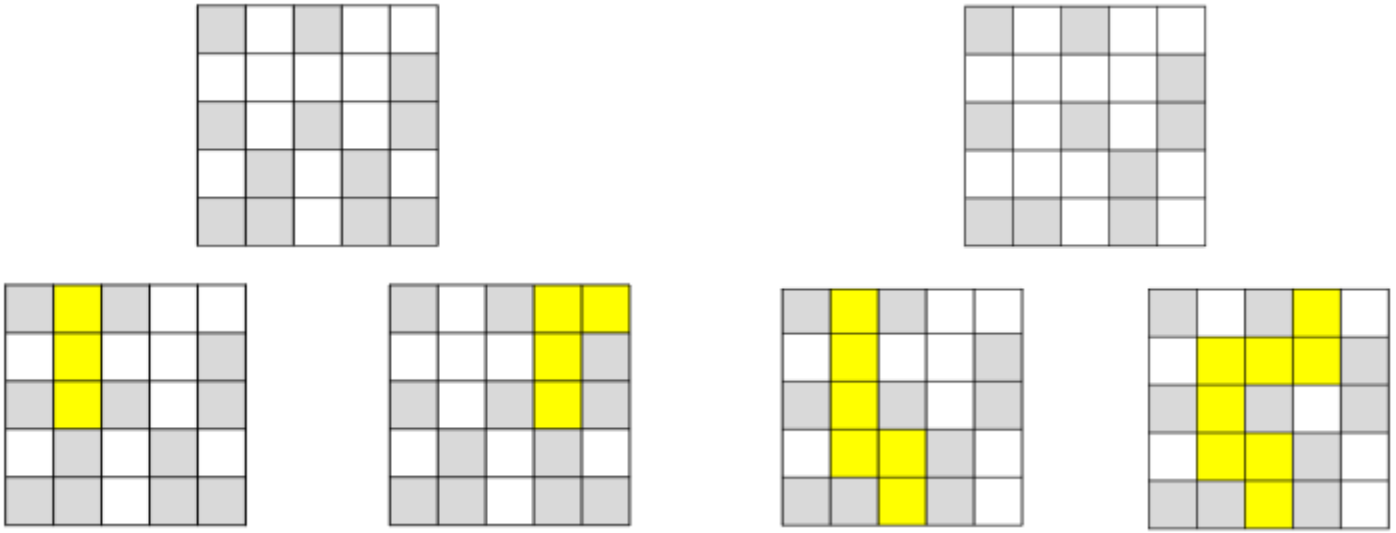
Details and answers to common questions

epcc

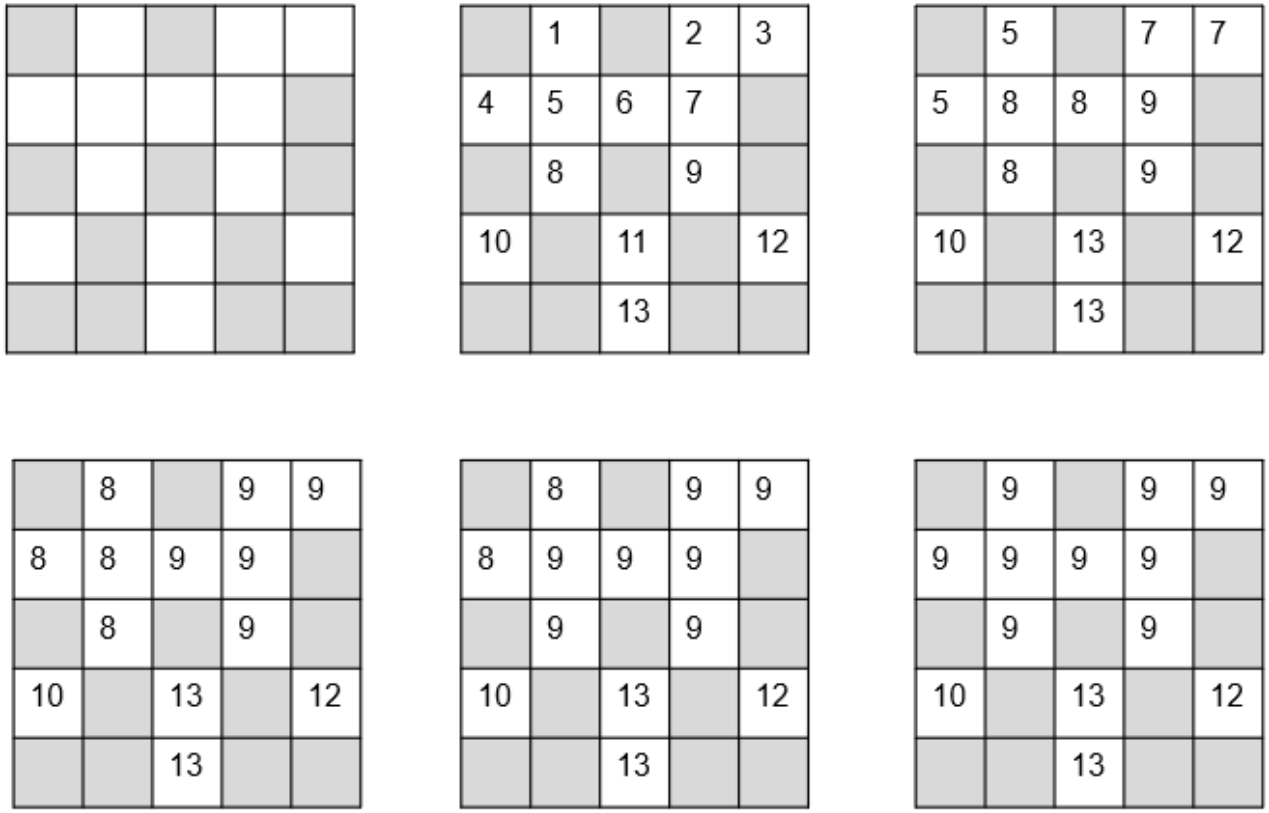
THE UNIVERSITY OF EDINBURGH

Reusing this material

Percolation



Algorithm


1 2 3
4 5 6 7
8 9
10 11 12
13
5 7 7
5 8 8 9
8 9
10 13 12
13
8 9 9
8 8 9 9
8 9
10 13 12
13
8 9 9
8 9 9 9
9 9
10 13 12
13
9 9 9
9 9 9 9
9 9
10 13 12
13

Halos

0	0	0	0	0	0	0
0	0	1	0	1	1	0
0	1	1	1	1	0	0
0	0	1	0	1	0	0
0	1	0	1	0	1	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	1	0	2	3	0
0	4	5	6	7	0	0
0	0	8	0	9	0	0
0	10	0	11	0	12	0
0	0	0	13	0	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	5	0	7	7	0
0	5	8	8	9	0	0
0	0	8	0	9	0	0
0	10	0	13	0	12	0
0	0	0	13	0	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	8	0	9	9	0
0	8	8	9	9	0	0
0	0	8	0	9	0	0
0	10	0	13	0	12	0
0	0	0	13	0	0	0
0	0	0	0	0	0	0

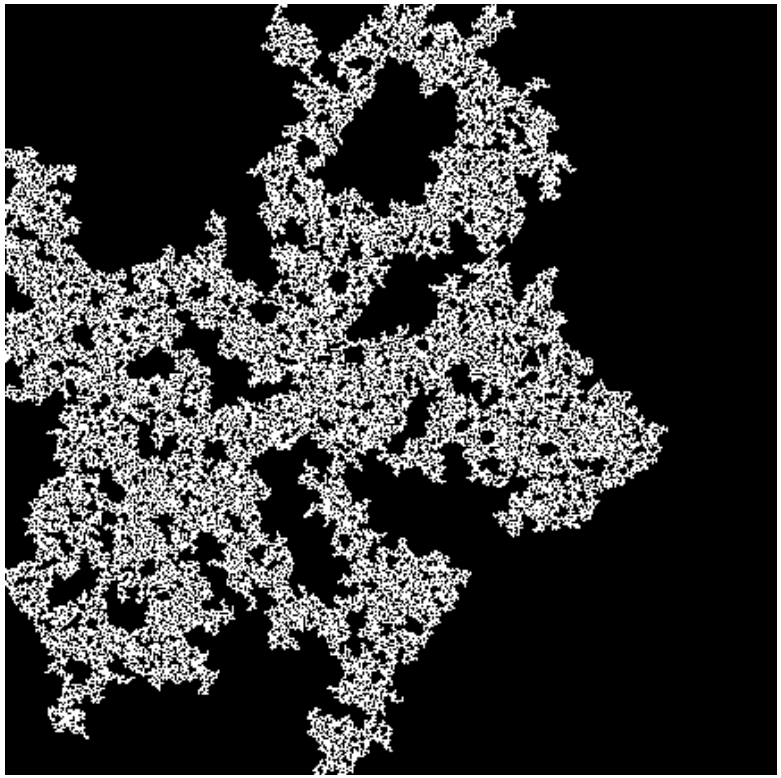
0	0	0	0	0	0	0
0	0	8	0	9	9	0
0	8	9	9	9	0	0
0	0	9	0	9	0	0
0	10	0	13	0	12	0
0	0	0	13	0	0	0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	0	9	0	9	9	0
0	9	9	9	9	0	0
0	0	9	0	9	0	0
0	10	0	13	0	12	0
0	0	0	13	0	0	0
0	0	0	0	0	0	0

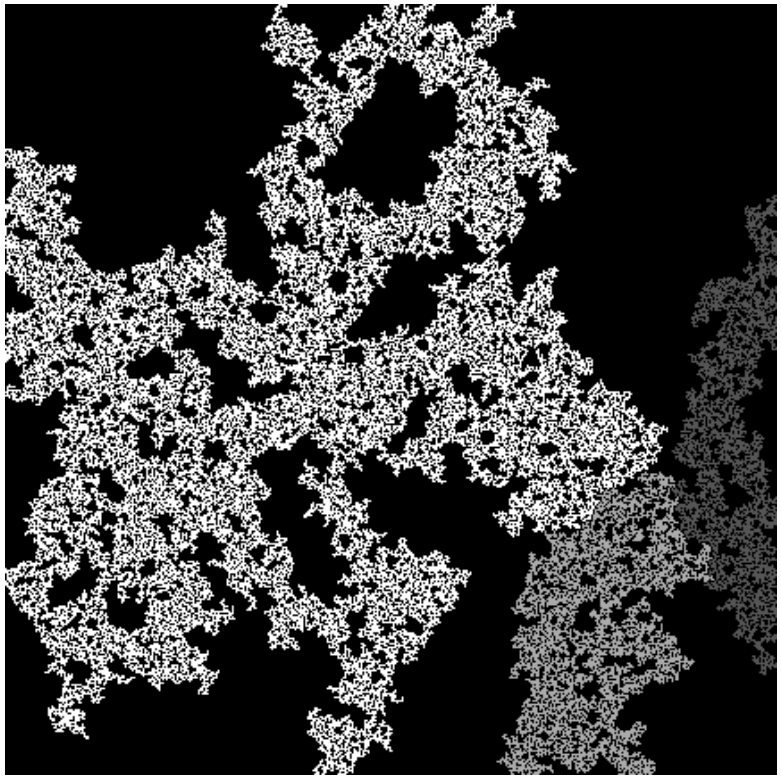
epcc

THE UNIVERSITY OF EDINBURGH

$L = 432$, $\rho = 0.411$, *seed* = 6543



Largest cluster (just fails to percolate)



Largest three clusters

Case study: serial code

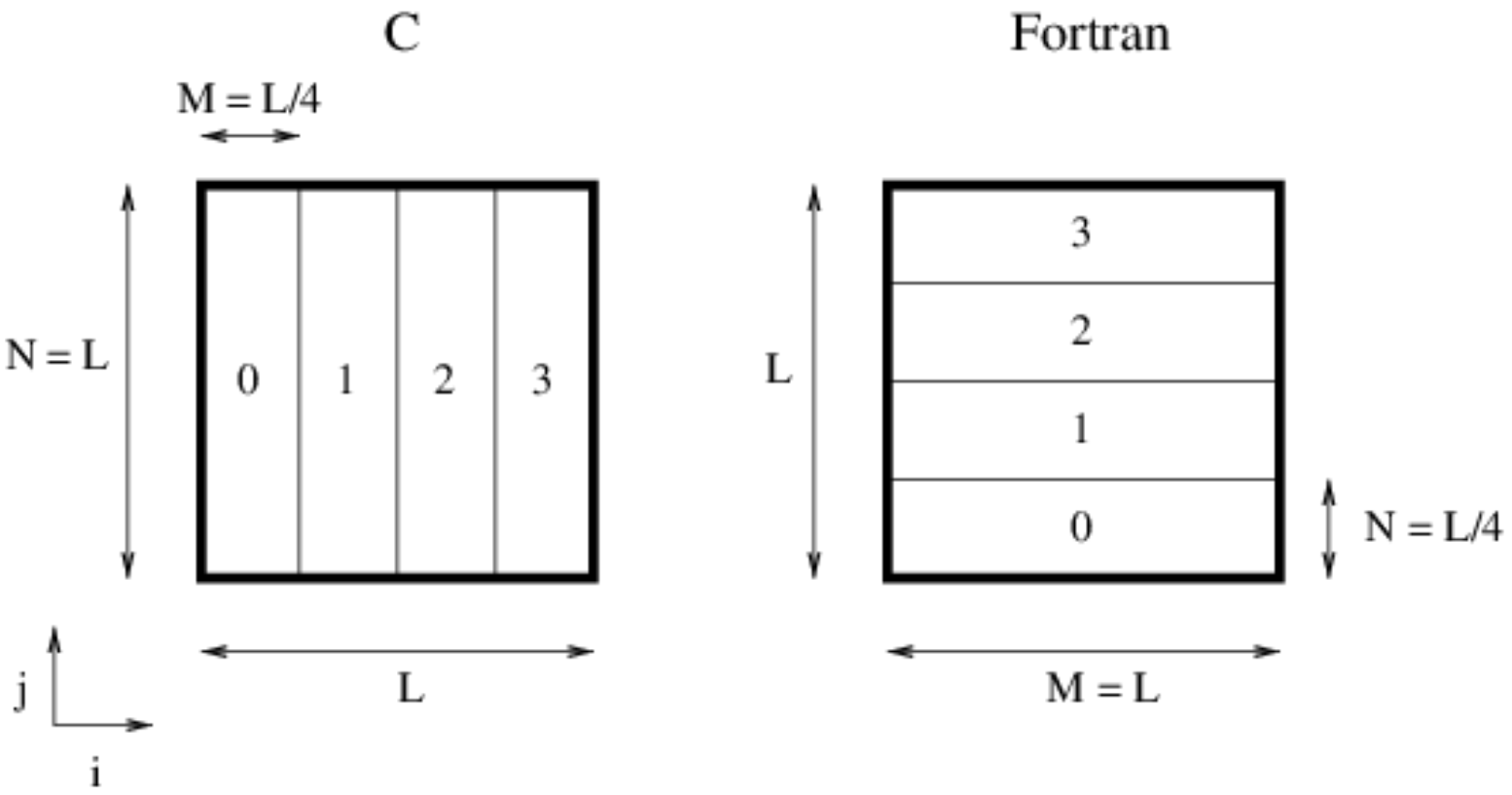
- Uses explicit halos for boundary conditions
  - distinguish between *L*, *M* and *N* although all the same values in serial
  - declares arrays to have bounds 0, 1, ..., *M+1* and 0, 1, ... *N+1*
  - fills boundaries (all pixels with *i*=0, *i*=*M*+1, *j*=0 or *j*=N+1) with 0
  - initializes random number generator with seed (a runtime argument)
  - set rock to zero with appropriate density ρ
  - fills empty spaces with unique positive values
  - applies the update equation for many steps
  - copies *old* back to *map* excluding the halos
  - writes *map* to an output file “map.pgm”
- Simplifications
  - map dimension *L*, density ρ and file name specified at compile time

Case study: parallel algorithm

- Split map into slices amongst all processes
  - divide across the first index *i* in C, second index *j* in Fortran
  - local array sizes are now *M* and *N* (plus halos)
    - C: *M* = *L*/*P*, *N* = *L*; Fortran: *M* = *L*, *N* = *L*/*P*
- Rank 0 initialises *L x L map* array (with no halos)
  - scatter to local *smallmap* array (also with no halos)
- Implement exactly the same calculation as before except:
  - **before start** of each step, swap boundary values as appropriate
- Gather *smallmap* array back to *map*
  - Rank 0 writes to file

Domain decomposition

- Different choices in C and Fortran



Coursework

- Differences from Case Study
  - use a 2D decomposition, i.e. try to split into many rectangular regions, not restricted to slices
    - requires more halo swaps than before (and they must be non-blocking!)
    - you cannot do the distribution and collection using scatter and gather
  - different boundary conditions
    - periodic in *second* dimension: $map_{i,0} = map_{i,N}$; $map_{i,N+1} = map_{i,1}$
  - compute the average value of map at regular intervals
  - stop when *nchange* = 0

Timeline

- Start working on Case Study example as soon as you have completed the other exercises
  - I will hand out simple solutions to the exercises next week
- First coursework due 16:00 GMT on Fri 6th Nov
  - *plan* for how your final 2D coursework solution will be designed, implemented and tested (to improve on 1D Case Study)
  - performance evaluation of 1D Case Study
    - sample 1D solution will be provided
- Second coursework due 16:00 GMT on Wed 2nd Dec
  - source code for your 2D implementation
  - report on the *actual* implementation, testing and performance

Questions

- What if the map does not decompose exactly?
  - you might want to deal with this case (see tutorial problems)
  - however, perfectly acceptable to quit and say “can’t do this!”
    - can still run on many different process counts
- What about a prime number of processes?
  - on 3 processes, for example, your code should just use a 1x3 or 3x1 decomposition (i.e. slices as for the case study)
    - 1D decomp is a subset of 2D: your code should run without problems
- Should I get the same answer?
  - Yes! The exercise has been designed so that (rather unusually for a parallel program) I would expect you to get exactly the same answer regardless of how many processes you run on

Hints (i)

- Do no worry about doing elegant IO
  - just implement something simple that works
  - e.g. broadcast the whole map to all processes
    - copy relevant section to your local smallmap array
- Test section by section
  - run on a single process
    - checks for serious bugs!
  - run for zero steps
    - checks the initialisation, distribution, collection and IO are correct
  - run with no halo swaps
    - checks the calculation is correct
  - run with only horizontal or vertical halo swaps
  - ...

Hints (ii)

- You don’t always need to run for thousands of steps
  - can set a maximum count for performance testing
- You are welcome to experiment with larger or smaller maps
- Bring your problems to the tutorial and drop-in sessions!