MSc Programming Skills

Performance Experiments Assessment

Adrian Jackson, Mark Bull, David Henty, Alan Simpson, and Mike Jackson

25 August 2020

Contents

1 Introduction

In this assessment you are given a program that implements a scientific algorithm. You will design and carry out a performance experiment to assess the performance of this program.

The program is implemented in C. It is expected that you may not be familiar with C. A valuable programming skill is being able to use, read and profile code that is written in a language you have not encountered before, learning about the language as you go. A summary of the key aspects of C syntax and how to compile and run C programs is provided online at the same location as this handout.

The assessment must be done individually.

1.1 Questions about the assessment

Please e-mail the course organiser if you have any questions, especially those relating to C syntax, compilation and linking. Questions of interest to the whole class will be anonymised and the question, plus the answer, forwarded to the whole class.

2 Percolate

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 an empty (white) square on the top row to an empty square on the bottom row, following only empty squares. This is a bit like solving a maze. The connected 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.

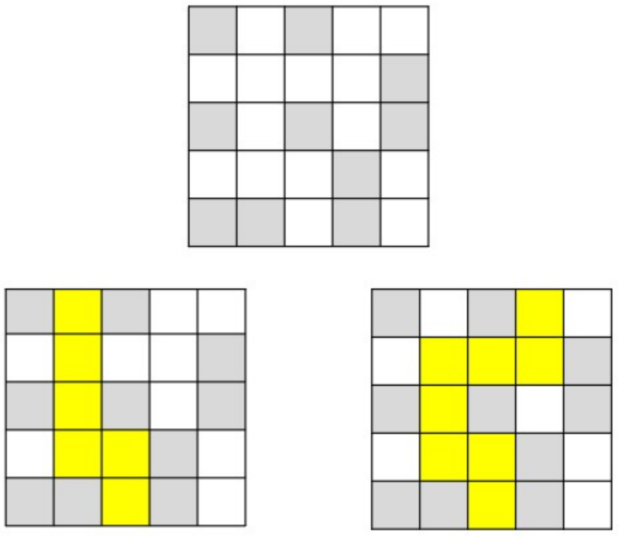


Figure 1: The grid at the top has a cluster which has paths through empty squares from the top to the bottom of the grid. Two examples of these paths are highlighted in the grids at the bottom. This cluster percolates.

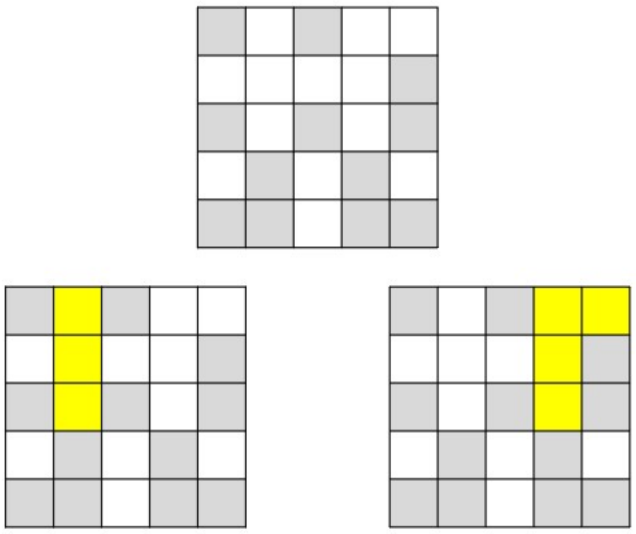


Figure 2: The grid at the top has no cluster with a path through empty squares from the top to the bottom of the grid. Two examples of the longest paths are highlighted in the grids in the bottom. This grid does not have a cluster that percolates.

To solve our problem, to see whether there is a path from an empty (white) square on the top row to an empty square on the bottom row, following only empty square, the largest clusters may be of interest.

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.

2.1 Solving Percolate

One solution to identify whether there is a cluster such that a path of empty squares from top to bottom through a cluster, that a cluster "percolates", is as follows. First, we initialise each of the empty squares with a unique positive integer (all the filled squares are set to zero) Then, we loop over all the squares in the grid many times and during each pass of the loop we replace each square with the maximum of its four neighbours. In all cases, we can ignore the filled (grey) squares. The large numbers gradually fill the gaps so that each cluster eventually contains a single, unique number. This then allows us to count and identify the clusters. If we find a part of a cluster on both the top and bottom row of the grid and it shares the same number, then we know that this is the same cluster and so we can conclude that the cluster percolates. An example is shown in Figure 3.

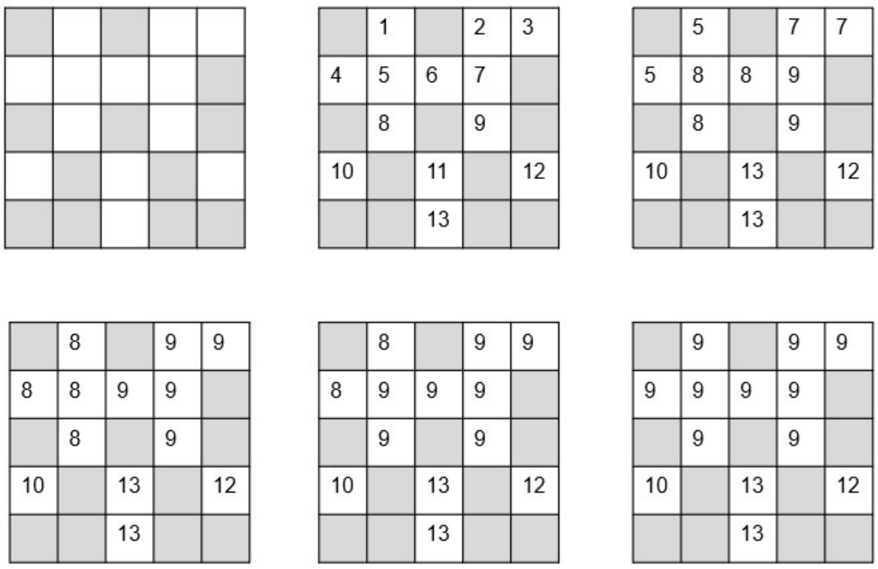


Figure 3: Solving Percolate. Top-left: our initial grid. Top-middle, grid initialised with unique numbers 1 to 13. Top-right: grid after first round of updates (8 squares are updated) Bottom-left: grid after second round of updates (5 squares are updated). Bottom-middle: grid after second round of updates (2 squares are updated). Bottom-right: grid after second round of updates (2 squares are updated). No squares would be updated on a subsequent round. 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? We do not want to write a lot of code to handle these as a special case, so a common solution to this problem is to define a "halo" around the grid. So, to solve Percolate for a L × L grid we use a (L+2) × (L+2) grid and set the halo squares to be filled (i.e. grey). We can do this and then set these values to zero as the zeroes will never propagate to other squares. See Figure 4.

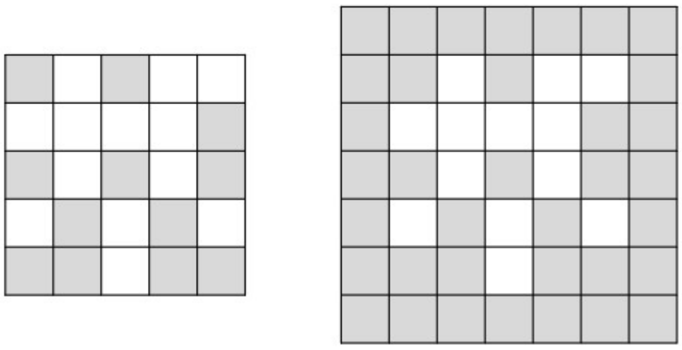
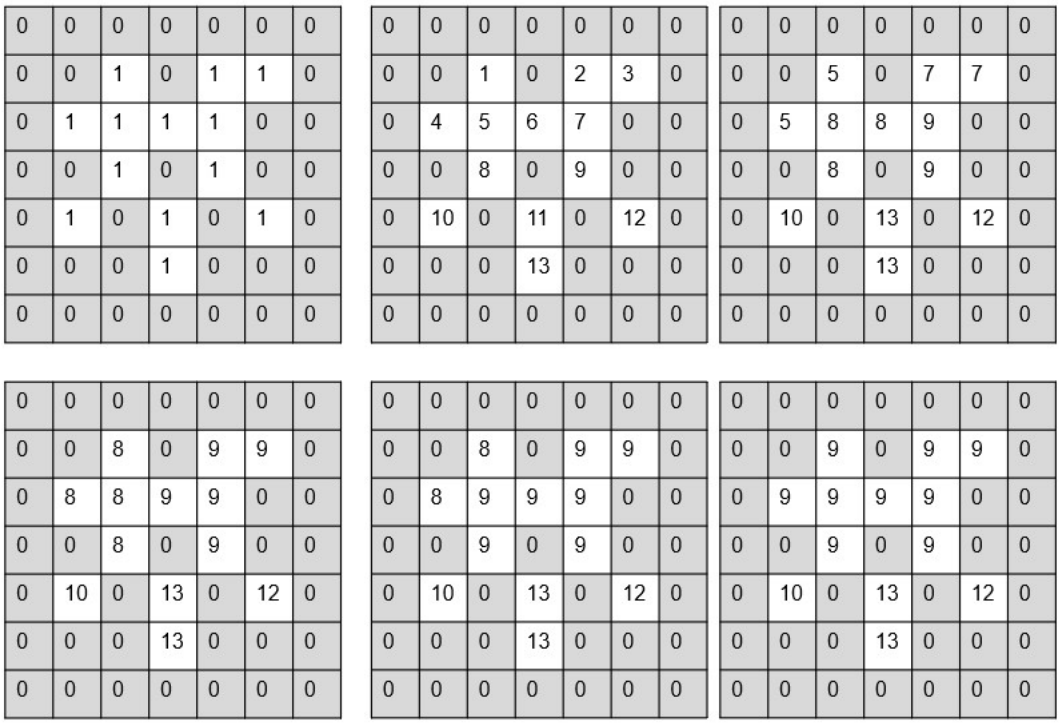


Figure 4: To process a 5x5 grid, we add a halo, with empty squares round the border of the grid, to produce a 7x7 grid.

See Figure 5 for the example of Figure 3 but with the use of a halo.



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

Figure 5: Solving Percolate using a halo

3 A C program to Percolate

The program is available online at the same location as this handout. The program implements the above approach to solving the Percolate problem.

The program is in files called `coursework-percolate-performance.zip` and `coursework-percolate-performance.tar.gz` (both files contain the same contents).

Download one of these files and unpack it.

Unpacking the bundle will produce a folder called `coursework-percolate-performance`.

3.1 Compiling and running the C program

A `README.md` file in the above folder explains how to compile and run the program.

3.2 What the program does

The program creates a grid. Each square is randomly assigned to be empty (1) or filled (0) according to the designed density distribution of filled squares. Each empty square is then assigned a unique number. The program iteratively updates the grid, updating each square is updated with the maximum of its neighbours. 循环结束标志 When an iteration results in no squares being updated, the iteration exits. The program then checks whether there is a cluster on the top row of the grid that has the same number as one on the bottom row i.e. it checks whether percolation has occurred.

The program writes out two files. The first is a data file which contains the values of each empty square when each square can be no longer updated i.e. the final state of the grid.

The second file is a type of image file, a Portable Grey Map (PGM) image file. The largest cluster will be coloured black. Other clusters will be shades of grey, the smaller the cluster the lighter the shade. The filled squares will be coloured white.

These files do not include the halo as the use of a halo is an implementation detail.

3.2.1 Converting grids to PGM files

The plain-text file holds the final state of the grid. For example, consider the following file with the final state of the grid from our example:

```
   0    9    0    9    9

   9    9    9    9    0

   0    9    0    9    0

  10    0   13    0   12

   0    0   13    0    0
```

This grid has 4 clusters. The cluster with value 9 is the largest cluster, and the size of this cluster is, coincidentally, 9 squares. The second largest cluster is the cluster with value 13, and the size of this cluster is 2 squares.

The program converts the grid into a PGM file. An example corresponding to the above data is below. The first line "P2" states that this is a PGM file. The second line specifies the height and width of the grid. The third line has the maximum colour value. This is equal to the number of clusters. The colour values range from 0 to the number of clusters inclusive. Clusters are converted to colours as follows. If there are C clusters then the largest cluster is assigned value 0 (black), the 2nd largest cluster value 1 (a deep grey) and so on up to the Cth cluster, the smallest, which is assigned the colour value C-1. The colour value of C itself is used for the filled squares in the grid.

```
P2

5            5

4

   4   0   4   0   0

   0   0   0   0   4

   4   0   4   0   4

   2   4   1   4   3

   4   4   1   4   4
```

For our example there are 4 clusters, so the maximum colour value is 4. The largest cluster (the cluster of values of 9 above) is assigned the colour value 0 (look at how the positions of the 9 values in the data file are the same as the positions of the 0 values in the PGM file). The second largest cluster (the cluster of values of 13 above) is assigned the colour value 1. The filled squares (with value 0 above) are assigned the maximum colour value of 4.

A variable allows this behaviour to change so that only a specific number of the largest clusters are shown.

4 Assessment

You must design and carry out a performance experiment and write a report describing your experiment and your results. It is up to you what experiment you conduct, but some ideas include:

- Investigate the effects on performance of different compiler optimisation flags.
- Identify hotspots in the program which could be optimised in future.
- Investigate the behaviour or performance of the program when configured with different sized maps.
- Investigate the behaviour or performance of the program when configured with different proportions of empty and filled squares.
- Any other performance-related experiment you think are of interest,

Do **not** carry out all the experiments above – you only need to design, execute, and write up **a single experiment**. Quality is more important than quantity. Credit will be given for intelligent and insightful analysis of your results, in preference to the quantity of data you present.

Your performance report should be structured as follows:

- **Introduction**: Short introduction to the rest of the report.
- **Method**: Description of how you carried out your experiment, what your set up was, how you collected data.
- **Results:** Your results, including a clear graphical presentation of any performance data.
- **Discussion**: Analysis and discussion of your results.
- **Conclusion**: Some brief conclusions and suggestions for future work.

Your report must a **maximum of 8 pages**. Any submission of longer than 8 pages will result in any pages after the first 8 pages not being read or marked.

4.1 Submitting your assessment

Submissions are done via the Turnitin submission tool from within LEARN. There will be a submission area set up for this assessment.

You are required to submit your performance report as a PDF document.

The filename of your submission must include:

- Your **exam number** which is the "B" number which appears on your matriculation card (this is **not** the same as your student number that begins with "s").
- The course identifier, `PS`, for Programming Skills.
- The assessment identifier, `Assessment2-Performance`.

For example, if your exam number is `B123456`, then you would name your submission file `B123456-PS-Assessment2-Performance.pdf`.

4.2 Deadline

The deadline for submission for your assessment is **16:00 Wednesday 25th November 2020** for all students.

Submission are allowed up to 7 calendar days after the deadline, with a 5% deduction per day or part-day late.

If you submit your assessment by the deadline then it will be marked, and you cannot submit an amended version to be marked later.

If you do not wish the submitted version of your assessment to be marked, and intend submitting an amended version after the deadline, then it is your responsibility to let the course organiser know, *before the deadline*, that the submitted version should not be marked.

4.3 Marking

Your submission will be marked out of 100 as follows:

- Introduction: 10
- Experimental method: 30
- Results: 20
- Discussion: 30
- Conclusions: 10
- **Total: 100**

The mark for this assessment forms 50% of your overall mark for the Programming Skills course.

4.4 Provisional marks

It is intended that your feedback and provisional marks will be released to you on 16:00 Wednesday 16th December 2020. All marks are provisional until confirmed by the MSc course board in January 2021.