CSC2002S Tutorial 2020

Parallel Programming with the Java Fork/Join framework: Terrain Classification

It is becoming common to undertake scans of the earth's surface for a variety of purposes, including agricultural and forest management, studies of climate change, and other scientific applications. This also includes so-called bare earth surveys, which map the height of the earth's surface, stripped of building and trees. In fact, most of the earth's surface has already been mapped in this way, albeit at a relatively course resolution. The datasets involved can be quite large and their analysis complex, which makes them good candidates for acceleration using parallel computing.

One such analysis is to determine how water flows across the landscape in order to protect against flooding and manage crops. As a precursor to this kind of analysis, in this assignment, you will take a scanned input terrain and find local minima (small-scale basins) where water might accumulate. Figure 1 shows an example of the type of terrain data that you will be analysing.

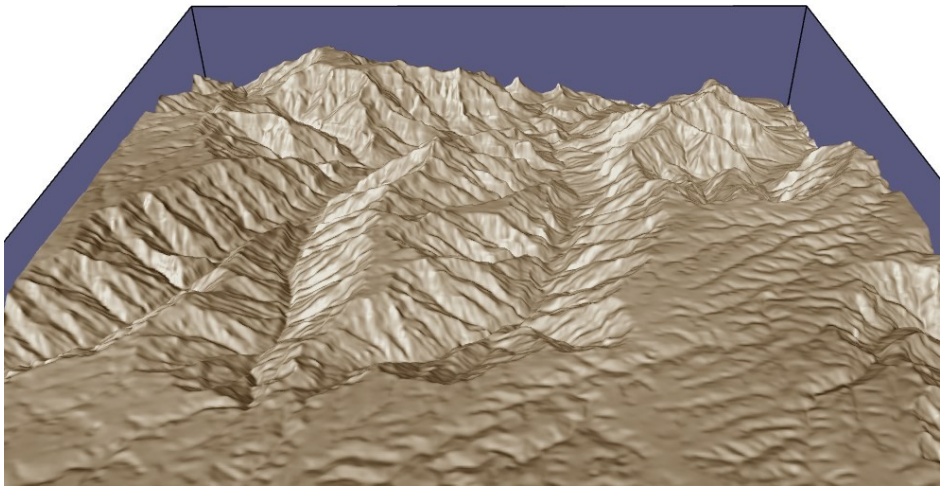


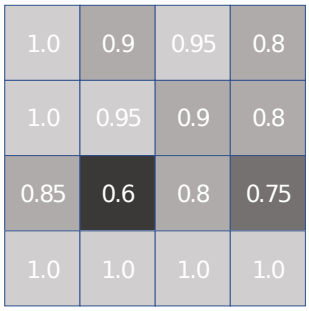
Figure 1: A rendering of a scanned terrain from the Grand Canyon area, in the United States.

You will be provided with data for a terrain represented as a regular grid of height values in meters above sea level, with each height encoded as a single floating point value.

The file input format is as follows:
<terrain num rows – INT> <terrain num cols – INT>
<height at grid pos (0,0) - FLOAT> <height at grid pos (0,1) - FLOAT> … etc.

For example, an unrealistically small 4 x 4 terrain might be encoded in an input file as:
4 4
1.0 0.9 0.95 0.8
1.0 0.95 0.9 0.8
0.85 0.6 0.8 0.75
1.0 1.0 1.0 1.0

This corresponds in a simple visualization to the following:



This trivial example terrain has a local minimum or basin at (2, 1) or row 2, column 1 with the column and row indices beginning from 0. Your task is to categorize every grid point as either a basin or non-basin. A basin is determined for a given grid position by checking the ring of neighbouring height values. If all of the neighbours are at least 0.01m higher than the grid point then it is classified as a basin. (Using an offset of 0.01m allows for some noise in the measurement data). In this case the neighbours around index (2,1), namely 1.0, 0.95, 0.9, 0.8, 1.0, 1.0, 1.0, 0.85 are all greater than or equal to 0.6+0.01=0.61 and so it is classified as a basin. There is no need to classify points on the outer edge of the grid as they are automatically considered non-basins.

To finish, a pass over the classified grid values is used to extract a list of basins, which is then output to file. Note that this final pass can be excluded from your parallelization efforts and any performance timings.

The file output format is as follows:
<number of basins – INT>
<first basin row index – INT> <first basin column index – INT>
<second basin row index – INT> <second basin column index – INT>
…

For the toy problem above (which only has a single basin) this would mean an output of:
1
2 1

In this assignment you will attempt to parallelize the problem in order to speed it up. You will be required to:

- Use the Java Fork/Join framework to parallelize the terrain classification using a divide-and-conquer algorithm.

- Evaluate your program experimentally:
    - Using high-precision timing to evaluate the run times of your serial and your parallel method, across a range of input sizes, and
    - Experimenting with different parameters to establish the limits at which sequential processing should begin.

- Write a report that lists and explains your findings.

Note that parallel programs need to be both **correct** and **faster** than the serial versions. Therefore, you need to demonstrate both correctness and speedup for your assignment. If

speedup is not achieved you need to explain why.

Assignment details and requirements

Your program will find local basins in an input terrain if they exist.

Input and Output

Your program must take the following command-line parameters:
<data file name> <output file name>
The format of these files must follow the specification provided above.

We will provide you with sample input files (on Vula), though you may also create your own using random data.

1.2 Benchmarking

You must time the execution of your parallel classification across a range of data **sizes** and **number of threads** (sequential cutoffs) and report the speedup relative to a serial implementation. The code should be tested on a single **multi-core machine architecture.**

Timing should be done at least 20 times. Use the System.currentTimeMillis method to do the measurements. Timing must be restricted to the terrain classification (that is the process in which every grid point is classified as TRUE (non-basin) or FALSE (a basin)). Timing should exclude the time to read in the file beforehand, and extract the list of basins afterwards, as well as other unnecessary operations, as discussed in class. Call System.gc()to minimize the likelihood that the garbage collector will run during your execution (but do not call this within your timing block!).

1.3 Report

You must submit an assignment report **in pdf format**. Your clear and **concise** report should contain the following:

- An *Introduction*, comprising a short description of the aim of the project, the parallel algorithms and their expected speedup/performance.
- A *Methods* section, giving a description of your approach to the solution, with details on the parallelization. This section must explain how you validated your algorithm (showed that it was correct), as well as how you timed your algorithms with different input, how you measured speedup, the machine architecture you tested the code on and interesting problems/difficulties you encountered.
- A *Results and Discussion* section, demonstrating the effect of data sizes and numbers of threads and different architectures on parallel speedup. This section should **include speedup graphs** and **a discussion**. Graphs should be clear and labelled (title and axes). In the discussion, we expect you to address the following questions:
    - Is it worth using parallelization (multithreading) to tackle this problem in

Java?
    - For what range of data set sizes does your parallel program perform well?
    - What is the maximum speedup obtainable with your parallel approach? How close is this speedup to the ideal expected?
    - What is an optimal sequential cutoff for this problem? (Note that the optimal sequential cutoff can vary based on dataset size.)
    - What is the optimal number of threads on your architecture?
- A *Conclusions* (note the plural) section listing the conclusions that you have drawn from this project. What do your results tell you and how significant or reliable are they?

Please do NOT ask the lecturer for the recommended numbers of pages for this report. Say what you need to say: no more, no less.

1.4 Assignment submission requirements

- You will need to submit a GIT usage log as a .txt file (use `git log –all` to display all commits and save this as a separate file).
- Your submission archive must consist of BOTH a technical report and your solution code and a **Makefile** for compilation.
- **Label** your assignment archive with your **student number** and the **assignment number e.g. KTTMIC004_CSC2002S_Assignment1.**
- Upload the file and **then check that it is uploaded.** It is your responsibility to check that the uploaded file is correct, as mistakes cannot be corrected after the due date.

Target Deadline:

10am on 24th August 2020

- Submissions will be accepted until the **hard deadline of 10am 27th August 2020** without penalty. Submissions after the hard deadline will not be accepted.
- The deadline for marking **queries** on your assignment is **one week after the return of your mark.** After this time, you may not query your mark.

1.5 Extensions to the assignment

If you finish your assignment with time to spare, you can attempt one of the following for extra credit:

- Compare the performance of the Fork/Join library with standard threads.
- Expand the basin check so that it covers more than a single layer of surrounding grid points.
- Include a search for peaks, as well as basins, and report on how this effects performance.
- Parallelise the final extraction of the list of basins.

1.6 Assignment marking

Your mark will be based primarily on your analysis and investigation, as detailed in the report.

Rough/General Rubric for Marking of Assignment 1	
Item	Marks
Code – conforms to specification, code correctness, timing, style	15

and comments, produces correct outputs	
Documentation - all aspects required in the assignment brief are covered, e.g. Introduction, Methods, Results (with graphs and analysis), Conclusions.	25
Going the extra mile - excellence/thoroughness/extra work - e.g. additional investigations, depth of analysis, etc.	5
GIT usage log	2
Makefile – compile, docs, clean targets, readme	3
Total	50

Note: submitted code that does not run or does not pass standard test cases will result in a mark of zero. **Any plagiarism, academic dishonesty, or falsifying of results reported will get a mark of 0 and be submitted to the university court**.