CSC2002S ASSIGNMENT 1

LINDA MAFUNU

MFNLIN003

Introduction

The aim of this assignment is to compare the performance of Parallel programming versus Sequential programming. To achieve this objective, measure the amount of time it takes to find basins from the grid terrain data given using Parallel and Sequential programming then comparing the times. Below are the differences between Parallel and Sequential Programming.

Parallel Programming
Multiple things are done at once in one program by using multiple cores. It uses data decomposition, task decomposition and work pool to execute an operation. It is nondeterministic the order of execution is not in a fixed order

Sequential Programming
Execution starts at the beginning and proceeds one operation at a time in a fixed order. The order is determined by the program and its input ie.it is deterministic.

Methods and Classes

Several classes were created in order to archive the objective of the assignment namely Terrain, Sequential, Basin, Parallel and TestFileGenerator. Below are the descriptions on the functions and operations carried out of by the classes.

1. Terrain Class:

   The Terrain class is used to populate the grid terrain by taking in the text file grid terrain data used as input

   The class has instance variables row (number of rows in grid train) of type integer , column (number of columns in grid terrain)of type integer and heights (stores the grid terrain heights for each point in grid terrain as a two dimensional array) of type float.

   Terrain Constructor method named Terrain same name as the class name's purpose to initialize the object of the Terrain class by executing the java code. The method takes in the number of rows and columns size of the grid as parameters.

   The parameter values are assigned to the instance variables' parameters are used to set the two-dimensional array size.

   getHeight() method is an accessor method, it returns the float height at the requested row and column coordinate in the terrain grid. The method takes in the coordinate values (x, y)

as integer type and the point at position [y][x] in the heights two-dimensional array is returned.

setHeight() method is a mutator method, it is used to set the height values of the grid terrain. In this case this method it is used to populate the heights 2D array. It in the integer values of the grid terrain point coordinates (x, y) and the float value (h)of the height as parameters. The position [y][x] in the 2D array is assigned the value of h.

display() method is used to check if the grid terrain has be populated correctly based on the text file input data.

Grid() method is an accessor method it is used to return the grid terrain as a 2D array of type float. This is done by returning the Terrain class instance variable heights.

2. Sequential Class

This class carries out Sequential programming using a single thread of control with no concurrency to populate the graph and find the basins in the grid terrain.

A lot of java library utility classes and interfaces were imported into this class namely:

```
1java.io.FileReader;
2java.io.FileNotFoundException;
3java.io.*;
4java.util.Scanner;
5java.util.ArrayList;
6java.util.Iterator;
```

1 is used for reading the text files used as input containing the grid terrain dimensions and the float heights

2 is used for catching the FileNotFoundException in case there is an attempt to read files that do not exist or that are not in the file path passed as arguments

3 imports all java interfaces

4 is used in order to take in keyboard input

5 is used for the declaration of Array List in this class it is going to store the coordinates of the basins found on the grid terrain

6 is used for iterating through the array list

A static variable named startTime of type long is created that record the starting time of the finding basin in grid terrain operation.

A private method tick() is a mutator method used to set the start time of the find basins operation by calling the System.currentTimeMills() method.

A private static method tock() is an accessor method that returns the float value of the time elapsed between the start time of the find basin operation and the end time. The time elapsed is converted to seconds by diving by 1000 before it is returned.

A private static method basinFinder() is an accessor method that returns the number of basins found in the grid terrain as an integer. The method takes in a 2D array of type float as a parameter (`float[][] grid`). To find the basins a nested for loop is used to iterate through the array to check for basins. If a basin is found the

variable number_of_basins is incremented, then it is returned when all points are checked in the grid terrain. The condition for a point to be a basin is that the surrounding points are supposed to be greater than or equal to height of point plus 0.01meters. All outer points of the grid are non-basins so they should not be checked. So, the for loops should start at position 1 to exclude the first row and column and end at number of rows minus 1(`grid.length - 1`)and number of columns minus one (`grid[0].length – 1`)

A private static method allBasins() is an accessor method that returns an ArrayList of coordinate values of the basin points found in the grid terrain as an ArrayList of type String (basins) . The method is like the basinFinder but instead when a basin is found the coordinate (x, y) are added to ArrayList basins.

The main method is used to populate the grid terrain, to time the basin finder operation and find the number of basins and their associated coordinates in the grid terrain. A try catch operation is initiated to catch an exception in the file reading attempt. To populate the grid terrain a text file is passed in as an argument (`FileReader file = new FileReader(args[0]);`) to read in the dimensions of the grid terrain and the associated heights of the points. A Terrain object is created and initialised (`Terrain t = null;`) a BufferReader is used to read in the text file contents (`BufferedReader reader = new BufferedReader(file);`). The text file contains two lines, the first line contains the dimensions of the grid and the second is for the height in the grid

.So by reading in the first line and storing the values in an array where there is the space character the row and column dimension is accessed (`String[] row_column = line.split(" ");`). The terrain object size is set (`t = new Terrain(r, c);`) by using the dimensions accessed. A while loop is used to read in the second line that has heights and to set the terrain object t with the height values from the second text file contents. This is done by calling the Terrain class method setHeight() and passing the height accessed as a parameter to it (`t.setHeight(x_axis, y_axis, height);`). The next operation is to time the find basin operation and store the number of basins found and in the grid terrain and write the results to a text file. The operation is to be done at least 20 times and the average is stored to the created text file. The last operation is to call the allBasins() method and the ArrayList contents are written to a text file for comparisons cases .The contents are compared the provided expected output the current terrain grid data given from vula.

3. Parallel Class:

The class is used to populate the grid terrain, find the number of basins using parallel programming aand the time it takes to find the number of basins in the grid terrain.

This class carries out the Parallel programming with the use of multiple threads of control (i.e. With advent of multiple processors processes gained internal concurrency through lightweight sub-processes sharing access to other memory and each has its own stack) and fork/join parallelism. All threads share one collection of

objects and static fields and threads communicate through shared objects. Fork/join framework is for making divide and conquer algorithms easier to parallelise making it easier, simpler and less error prone for threading.

The Parallel class carries out similar operations as the Sequence class there are a few differences, additional methods and library utility classes imported to it (`4 import java.util.concurrent.ForkJoinPool;`) to it.

The tick() and tock() methods are the same and they carry out the same function but this time they are timing the find basin operation through fork/join parallelism.

A final static object of type ForkJoinPool is created and initialised it is unchanging.

A static accessor method is created named basins() it is used to find the basins in the grid terrain using fork/join parallelism.

In the main method the way the grid terrain is populated is the same as in the Sequential class. A Scanner utility class is used to accept keyboard input of the file name which is going to be created and used to store the number of basins found when the basins() method is called and the time elapsed between the start of the finding basin operation and end. When the basins() method the ForkJoinPool method is invoked to start the algorithm and then fork or compute the recursive calls.

4. Basin Class:

The basin class is used to find the basins in the grid terrain and return an ArrayList containing the coordinates of the basin points found in the grid terrain.

A RecursiveTask fork/join specialisation was is used because most divide and conquer algorithms return a value from a computation over a data case. In our case we want to return number of basins found basins found in the grid terrain. So that is why in our Basin class we imported the utility class concurrent (`3 import java.util.concurrent.RecursiveTask;`)

The Basin class extends Recursive task whose compute method is going to be overrode in order to carry out its required purpose.

The class has five instance variables namely:

```
8           float[][]grid;
9     int   row_low;
10          int   row_high;
11          int   column_low;
12          int   column_high
13     static final int ROW_CUTOFF=1500;
14     static final int COLUMN_CUTOFF=1500;
```

8 is a 2D array of grid terrain heights
13 and 14 are the sequential cut-off that determine where we are going to start using divide and conquer algorithm by use of threads.

Basins constructor method named Basin same name as the class name's purpose to initialize the object of the Basin class by executing the java code. It takes in the 2D array and 4 integer values as parameters. The instance variables are initialised.

The compute() method returns number of basins found in the terrain grid. If the condition (high-low) is less than the sequential cut-off then switches

to sequential algorithm checking the grid points one by one. This means that we should stop using the forking and joining. The method to find the basins like the one used in the Sequential code .findBasin() method.

Else we split the program into another 2-thread operation by processing the first half of the grid terrain and the 2 second half separately but concurrently by creating the 2 new Basin objects consisting of 1st half and the 2nd has the second half. The .fork() method recursively breaks the task into smaller pieces independent subtasks until they are simple enough to be executed. This method is invoked on the 1st half. Call the. compute() method to called on the 2nd half. Return the sum of number of basins found basins found in the 1st half and second half to the Parallel class. The row and column sequential cut-offs are constantly changed to find the ideal cut off depending on the data size (i.e. grid terrain dimension size) used as input. In this case the ideal cut off is 1500 for a maximum grid terrain of dimension 1500 by 1500.

5. TestFileGenerator Class:

This class is used to generate addition test data for variation of data sizes. 10 additional text files were used to be used as input to further get the time process operation for sequential and parallel programming. Rectangular and square grid terrain text files were created for testing the finding basin operation.

Results:

There is a file in the "src" file named "results" inside there is a "sequential" and "parallel" file. The files contain the text files results for the assignment. There are two text files for each grid terrain data input text file in the sequential file .The 1st text file contains for example text file named "small" contains the times elapsed for the 25 operation repetitions and the average time for the operations. The second corresponding text file named" small_out" contains the number of basins found in the grid terrain and the basin coordinates for each basin point. All 2nd files contents corresponded with the expected output files given on Vula in the Data file for Sequential Programming. In the parallel file contains text files for each grid terrain data text input file. The text files contain the for example the file named "small" contains the times elapsed for the 25 operation repetitions, the average time for the operation and the number of basins found in the grid terrain. The number of basins found during parallel programming corresponded to the ones found in sequential programming and the expected output from the data given on Vula. This is how it was verified that the code carries out its purpose.

Git log is stored in a text file named a.txt if not accessed below shows the git log:

```
commit ecd941c593eb2e8869bfb233707a998cb6b7603a (HEAD -> master, origin/master)
Author: Liz-cloud <mafunulinda@gmail.com>
Date:   Wed Aug 26 21:55:27 2020 +0200

:
commit ecd941c593eb2e8869bfb233707a998cb6b7603a (HEAD -> master, origin/master)
Author: Liz-cloud <mafunulinda@gmail.com>
Date:   Wed Aug 26 21:55:27 2020 +0200

    yesssss
:...skipping...
commit ecd941c593eb2e8869bfb233707a998cb6b7603a (HEAD -> master, origin/master)
Author: Liz-cloud <mafunulinda@gmail.com>
Date:   Wed Aug 26 21:55:27 2020 +0200

    yesssss

commit 038e676dbb8fba4efe48caf38a800d1ba6a93580
Author: Liz-cloud <mafunulinda@gmail.com>
Date:   Sun Aug 23 18:37:38 2020 +0200
:...skipping...
commit ecd941c593eb2e8869bfb233707a998cb6b7603a (HEAD -> master, origin/master)
Author: Liz-cloud <mafunulinda@gmail.com>
Date:   Wed Aug 26 21:55:27 2020 +0200

    yesssss

commit 038e676dbb8fba4efe48caf38a800d1ba6a93580
Author: Liz-cloud <mafunulinda@gmail.com>


    yessss
:...skipping...
commit ecd941c593eb2e8869bfb233707a998cb6b7603a (HEAD -> master, origin/master)
Author: Liz-cloud <mafunulinda@gmail.com>
Date:   Wed Aug 26 21:55:27 2020 +0200

    yesssss

commit 038e676dbb8fba4efe48caf38a800d1ba6a93580
Author: Liz-cloud <mafunulinda@gmail.com>
Date:   Sun Aug 23 18:37:38 2020 +0200

    yessss

commit fe4b662340aabc976e687db7789e555b3cdcead3
Author: Liz-cloud <mafunulinda@gmail.com>
Date:   Sun Aug 23 16:38:24 2020 +0200
:...skipping...
commit ecd941c593eb2e8869bfb233707a998cb6b7603a (HEAD -> master, origin/master)
Author: Liz-cloud <mafunulinda@gmail.com>
Date:   Wed Aug 26 21:55:27 2020 +0200

    yesssss

commit 038e676dbb8fba4efe48caf38a800d1ba6a93580
Author: Liz-cloud <mafunulinda@gmail.com>
Date:   Sun Aug 23 18:37:38 2020 +0200

    yessss

commit fe4b662340aabc976e687db7789e555b3cdcead3
Author: Liz-cloud <mafunulinda@gmail.com>
Date:   Sun Aug 23 16:38:24 2020 +0200

    finallly

commit 77456a631934ac9b4588385ba849d12c956ef3dd
Author: Liz-cloud <mafunulinda@gmail.com>
```

Below is a table showing the different grid dimensions and the sequential and parallel programming time it took to carry out the process. And the speed up and maximum speedup.

$$Speedup_n = T1 / Tn$$

T1 is the execution time for one core Tn is the execution time for n cores (in our case its 2 cores)

$Speedup_n$ should be >1

$$max\ Speedup = \lim_{n \to \infty} Speedup = 1 / Fsequential = 1 / (1 - Fparallel)$$

Grid	Sequential Time	Parallel Time	$F_{n\ parallel}$	$Speedup_n$	Maximum Speedup
0	0	0	0	0	1
4x4	0	0	0		1
5x2	0	0	0	0	1
10x10	0	0	0	0	1
15x20	0	0	0	0	1
100x100	0.00024	0.00016	0.4	1.5	1.00016
150x110	0.00112	0.00044	0.282051282	2.545455	1.00044
200x300	0.00192	0.00156	0.448275862	1.230769	1.001562
256x256	0.00144	0.00124	0.462686567	1.16129	1.001242
300x300	0.00224	0.00224	0.5	1	1.002245
512x512	0.004	0.00352	0.468085106	1.136364	1.003532
600x900	0.01264	0.01232	0.493589744	1.025974	1.012474
800x800	0.01524	0.01392	0.477366255	1.094828	1.014117
1024x1024	0.01336	0.0128	0.489296636	1.04375	1.012966
1500x1500	0.04816	0.04968	0.507767784	0.969404	1.052277

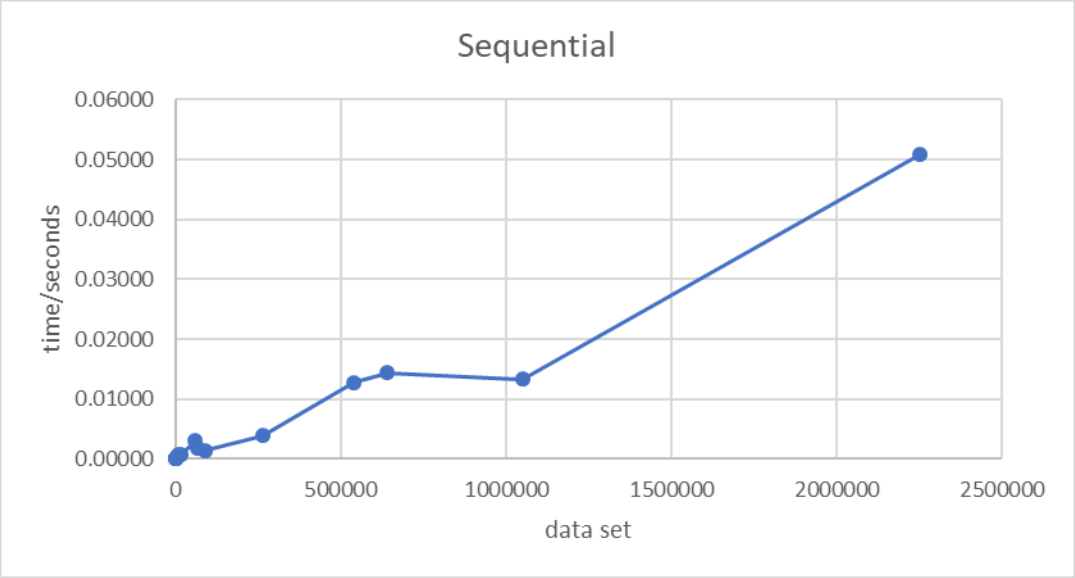
- From the above table of results, it clearly shows that Parallel programming takes less processing time in most cases to carry out the find grid basins operation.
- The speedup should be less than or equal to the number of cores which is 2.
- Maximum speedup attainable is 1.052277 according to the table of result above. The Gustafson-Barsis law states that speedup tends to increase with problem size (since the fraction of time spent executing serial code goes down).
- The optimum sequential cut off is 1500 for both rows cut off and column cut off any value lower will result in a stack overflow.
- Parallel programming code performs well for the grid range of 100x100(10000 dataset) to 1024x1024 (1048575 dataset)

Below are graphs and tables for both Parallel and Sequential Programming:

Sequential

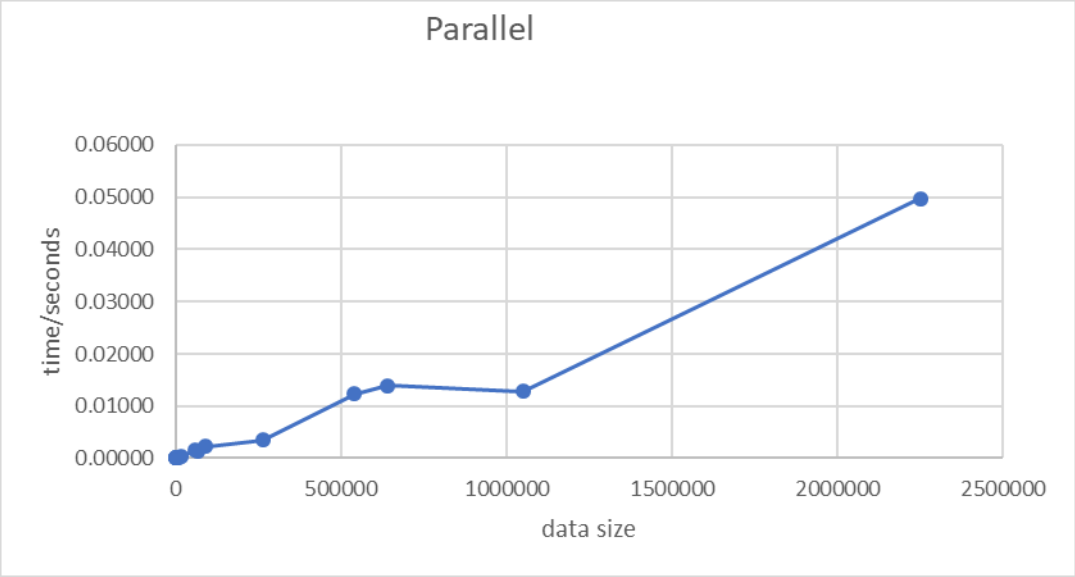
size	time(seconds)
0	0.00000
7	0.00000
16	0.00000
100	0.00000
300	0.00000
10000	0.00072
16500	0.00080
60000	0.00300
65536	0.00176
90000	0.00148
262144	0.00392

540000	0.01280
640000	0.01444
1048576	0.013360
2250000	0.05080



Parallel:

size	time(seconds)
0	0.00000
7	0.00000
16	0.00000
100	0.00000
300	0.00000
10000	0.00016
16500	0.00044
60000	0.00156
65536	0.00124
90000	0.00224
262144	0.00352
540000	0.01232
640000	0.01392
1048576	0.01280
2250000	0.04968



- The above graphs show that the larger the data size is the more time it takes the process to operation.

- It is worth multi-threading because since it faster takes less time even though it is a small difference sometimes, but it is worth it.

Conclusion:

From the project it seems that parallel programming is faster than sequential programming. It takes less time to process operations. The results drawn might have a few errors, so they are not as reliable since there are times like on the 1500x1500 grid sequential programming seemed to be taking less time than parallel programming which is not exactly plausible of this assignment's objective. This is because simply because the sequential version has no coordination overhead (breaking down work and building the total again). But on overall the results do show a clear trend on various data sizes that parallel programming indeed takes less time in processing operations.