

CSC2002S Parallel Programming

Assignment 3



September 9, 2019

Tawanda Muzanenhamo

MZNTAW004

Table of Contents

[1. Introduction 2](#_Toc19220001)

[2. Method 2](#_Toc19220002)

[2.1 Sequential Approach 2](#_Toc19220003)

[2.2 Parallel Approach 3](#_Toc19220004)

[2.3 Validation of Algorithm 3](#_Toc19220005)

[2.4 Timing 3](#_Toc19220006)

[2.5 Speed up 3](#_Toc19220007)

[2.6 Interesting Facts/Problems 4](#_Toc19220008)

[3. Results and Discussion 4](#_Toc19220009)

[3.1. The effect of varying Data sizes 4](#_Toc19220010)

[3.2. The effect of varying sequential cut off 6](#_Toc19220011)

[3.3. The effect of using different architectures 6](#_Toc19220012)

[4. Conclusion 9](#_Toc19220013)

[5. Git usage 10](#_Toc19220014)

Table of Figures

[Figure 1: Sequential vs Parallel Graph 5](#_Toc19220040)

[Figure 2: Dataset Vs Speed up 5](#_Toc19220041)

[Figure 3: Sequential cut-off vs Speedup 6](#_Toc19220042)

[Figure 4: Dataset Vs Run Time for 4 core machine 7](#_Toc19220043)

[Figure 5: Dataset Vs Speed up for 4 core machine 8](#_Toc19220044)

[Figure 6: Sequential Cut-off vs Speed up for 4 Core Machine 8](#_Toc19220045)

[Figure 7: Dataset vs Runtime for 2 and 4 core Machines 9](#_Toc19220046)

[Figure 8: Dataset vs Speed up for 2 Core and 4 Core 9](#_Toc19220047)

Table of Tables

[Table 1: The effect of varying data size table 4](#_Toc19220086)

[Table 2:Effect of Increasing Sequential Cut-off on speed up 6](#_Toc19220087)

[Table 3: The effect of increasing dataset on a 4 core Machine 7](#_Toc19220088)

[Table 4: The effect of varying sequential cut-off on a 4 core machine 7](#_Toc19220089)

Table of Equations

[Equation 1: Speed up formula. 3](#_Toc19220111)

[Equation 2: Amdahl's law 3](#_Toc19220112)

# Introduction

The aim of this assignment is to use the Fork/Join framework available in java to parallelise weather simulations. The results obtained from the parallelised version of the weather simulation are then compared with sequential version of the weather simulation in order to determine if parallelising programs is worth considering.

The Fork/Join framework helps the speed up parallel processing by using divide and conquer to distribute the work on all available processors. The framework first “forks” recursively breaks the task into subtasks and then ‘joins’ whereby it recursively joins the results of subtasks into a single subtask. If method is void, the program will wait until every subtask is executed. The parallel execution of the fork/join framework uses the ForkJoinPool which manages worker threads.

The Fork/Join framework is expected to achieve 2x speed up depending on the number of cores the computer architecture being used is.

# Method

The methods used to solve the weather simulation problem are discussed below

## Sequential Approach

The sequential approach method involved using the Scanner class in java to read the given input file. The sequentialCloudData.java class calculates the average of the wind by traversing through the given input file and summing values of the corresponding coordinate of the wind vector i.e. the wind vector has x and y coordinates. The sum of both x and y is then divided by the value returned by the dim method which is a product of the time step, size of x and size of y, this gives the average of wind vector which is the prevailing wind. The cloud classification is made comparing the local average and the uplift value. The local average is calculated by finding the mean of an element and 8 of its immediate neighbours. This is achieved through iterating in the range number of time steps, size of x and size of y depending on the current element index. The local average is then divided by 8 which is the number neighbours and in order to classify the clouds according to the given cloud’s codes, the magnitude of local averages is calculated and compared to uplift.

## Parallel Approach

The parallel approach implements the Fork/Join Framework which uses the divide and conquer approach. The approach differs from the sequential method in that had to traverse through the array of different matrixes calculating the sums of each matrix and in the end the sums of the matrix’s are then added together to get the overall sum. The sums of the matrix’s are computed by different threads and each thread writes to its own sum of the corresponding matrix.

## Validation of Algorithm

Due to the non-deterministic nature of threads, it’s impossible to check on the correctness of the algorithm until the program execution is complete hence multiple runs of the same program were run with same dataset and the same values for averages and cloud type were obtained and these values conformed to the sample output file that was given. In order to validate the correctness of, the files, the automaker.java class was used to compare the output files.

## Timing

In order to time the program execution, the java system’s library was used. The currentTimeMillis () function was called at the beginning of the execution and recorded as start time and then it was called again at the end of execution and recorded as the end time. The difference between the start time and stop time was then recorded as the time of execution.

## Speed up

Speed up was calculated using formula 1 below.

Equation 1: Speed up formula.

Two different machine architectures were used to test the execution of the program.

The predicted speed up depending on the machine architecture is calculated using Amdahl’s law.

Equation : Amdahl's law

Where p = the fraction of the algorithm that can be made parallel, n = number of processors hence for a 2-core machine the expected speed up when the sequential cut-off is 5000 is

And for a 4-core machine the expected speed up when the sequential cut-off is 5000 is

1. Intel Core i5-7200 CPU @2.50GHz with 2 cores
2. Intel core i5-4690 CPU @3.50GHz with 4 cores

## Interesting Facts/Problems

Whilst executing the programs I learnt that running other programmes whilst executing the parallel program resulted in the program executing much slower than when not running anything else this is because when no other programs are running, there is running then the entire program is fully dedicated to running the program. When there are many programs executing the run time increases because of sharing resources meaning that a scheduled thread will spend more time waiting in line to be allocated resources.

# Results and Discussion

## The effect of varying Data sizes

The experiment was conducted by varying different data sets for the sequential and parallelized program. In the parallelized program, the sequential cut-off was kept constant. The programs are run 3 times and the average is calculated. The following results were obtained.

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | Sequential Time | Parallel Time | Speed Up |
| 20x512x512 | 0.186 | 0.165 |  |
| 50x512x512 | 0.686 | 0.384 |  |
| 80x512x512 | 1.172 | 0.567 |  |
| 120x512x512 | 2.037 | 0.865 |  |
| 150x512x512 | 2.749 | 1.234 |  |
| 180x512x512 | 3.143 | 1.672 |  |
| 20x256x512 | 0.1185 | 0.0585 |  |

Table : The effect of varying data size table

A screenshot of a cell phone

Description automatically generated

Figure : Sequential vs Parallel Graph

A screenshot of a cell phone

Description automatically generated

Figure : Dataset Vs Speed up

From the table above, it is evident the bigger the dataset, the greater the amount of time taken for the sequential and parallel program to finish executing. The dataset that achieved the greatest speedup of 2.355 is of size 120x512x512. Data sets greater than that had less speedup, and this can be attributed to the fact that there are too many threads hence more there is an overhead for managing threads. The speed up obtained in larger than 2 which is the expected speed up this is because a 2-core machine had very minimal background processes running whilst executing the algorithm.

## The effect of varying sequential cut off

The sequential cut-off refers to the smallest size of a dataset that runs faster sequentially than in parallel. The effect of different sequential cut-off values was investigated in order to find the optimal value where the parallel version is most efficient i.e. where there are enough threads to fully harness the power if parallel programming. The dataset used to investigate different sequential cut offs is 50x512x512.

|  |  |  |  |
| --- | --- | --- | --- |
| Sequential Cut-off | Sequential Time | Parallel Time | Speedup |
| 1000 | 0.686 | 0.384 | 1.78 |
| 5000 | 0.686 | 0.276 | 2.48 |
| 10000 | 0.686 | 0.274 | 2.51 |
| 25000 | 0.686 | 0.298 | 2.302 |
| 50000 | 0.686 | 0.294 | 2,33 |
| 75000 | 0.686 | 0.3505 | 1.957 |
| 100000 | 0.686 | 0.362 | 1.90 |
| 150000 | 0.686 | 0.387 | 1.77 |
| 300000 | 0.686 | 0.4355 | 1.575 |
| 500000 | 0.686 | 0.447 | 1.536 |
| 800000 | 0.686 | 0.434 | 1.58 |
| 1000000 | 0.686 | 0.459 | 1.494 |
| 2000000 | 0.686 | 0.589 | 1.16 |

Table :Effect of Increasing Sequential Cut-off on speed up

A screenshot of a cell phone

Description automatically generated

Figure : Sequential cut-off vs Speedup

From the results obtained, it can be seen that the best speed up was obtained when the sequential cut-off was between 5000 and 10000. After the 10000 sequential cut-off there was a decrease in the speed up meaning that too many threads were created and there was much cost in creating and destroying threads that the power of parallelization was not fully harnessed.

## The effect of using different architectures

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | Sequential Time | Parallel Time | Speed-up |
| 20x512x512 | 0.093 | 0.051 | 1.82 |
| 50x512x512 | 0.313 | 0.115 | 2.72 |
| 80x512x512 | 0.475 | 0.168 | 2.83 |
| 120x512x512 | 0.752 | 0.252 | 2.98 |
| 150x512x512 | 0.920 | 0.34 | 2.70 |
| 180x512x512 | 1.07 | 0.452 | 2.36 |

Table : The effect of increasing dataset on a 4 core Machine

|  |  |  |  |
| --- | --- | --- | --- |
| Sequential Cut-off | Sequential Time | Parallel Time | Speedup |
| 1000 | 0.313 | 0.142 | 2.21 |
| 5000 | 0.313 | 0.117 | 2.67 |
| 10000 | 0.313 | 0.115 | 2.72 |
| 25000 | 0.313 | 0.119 | 2.64 |
| 50000 | 0.313 | 0.127 | 2.57 |
| 75000 | 0.313 | 0.132 | 2.38 |
| 100000 | 0.313 | 0.139 | 2.25 |
| 150000 | 0.313 | 0.148 | 2.1 |
| 300000 | 0.313 | 0.154 | 2.03 |
| 500000 | 0.313 | 0.166 | 1.88 |
| 800000 | 0.313 | 0.179 | 1.75 |
| 1000000 | 0.313 | 0.181 | 1.73 |
| 2000000 | 0.313 | 0.2 | 1.56 |

Table : The effect of varying sequential cut-off on a 4-core machine

Methods 3.1 and 3.2 were repeated on different machine with processor specifications listed in section 2.5. The results obtained are shown below.

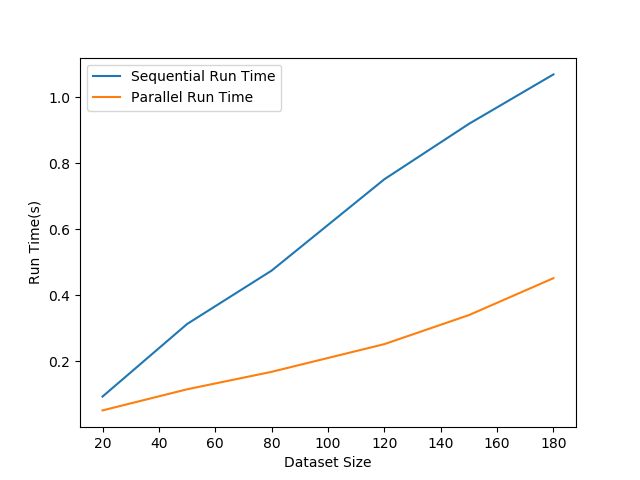


Figure : Dataset Vs Run Time for 4 core machines

A screenshot of a cell phone

Description automatically generated

Figure : Dataset Vs Speed up for 4 core machines

Using a different architecture, 4 core computer produced the results and graphs showed in table 3, figure 4 and figure 5 respectively. The results show that as the size of the dataset increases, the amount of time taken by the program to finish executing is in conformance with what was observed on a core i3. However a 4 core takes less time with it’s run time for a 120x512x512 data set being 0.752 and 0.252s for the sequential and parallel executions respectively whereas those for a 2 Core are 2.037s and 0.865 for the sequential and parallel executions thus a 4 core machine is almost twice as fast as the 2 core machine. The results are in conformance with the theory that a 4-core computer has more logical processors hence it executes faster than a 3-core machine. However, the fastest speed up achieved for a 120x512x512 dataset using the 4-core machine is 2.98 which is less than the speed up of 4 which is predicted by Amdahl’s law. This can be because more time was made creating and destroying threads for the dataset hence achieving a less speed up.

A screenshot of a cell phone

Description automatically generated

Figure : Sequential Cut-off vs Speed up for 4 Core Machine

The effect of varying as sequential cut off shows that the more the sequential cut off increases the more the time spent executing the program due to the decrease in the number of threads.

A screenshot of a cell phone

Description automatically generated

Figure : Dataset vs Runtime for 2 and 4 core Machines

A close up of a map

Description automatically generated

Figure : Dataset vs Speed up for 2 Core and 4 Core

The graphs above show the comparison of a 2 core and 4 core machine in terms of run time and speed up and the 4 core machine is seen to be much faster than a 2 core machine because of the availability of more logical processors in a 4 core machine that handles the threads.

# Conclusion

From the results it can be seen that using a 2 core machine achieves the predicted speed up of 2.355, and a 4 core machine achieve a speed up of 2.98 proving that 4 core machines are faster than 2 core machines thus conforming to the theory discussed in class and this proves that multi-threading can be very beneficial if applied to large dataset and using a higher core machines. Parallel programs are useful because it fully harnesses the power of computers by distributing the processes amongst the computers processors and using all the available processor resources concurrently.

# Git usage

$ git log

commit ce4da33dc9ec87bdc04f86f4d2a6eec538d618ef (HEAD -> master)

Author: Tawanda <tawandamuzanenhamo14@gmail.com>

Date: Thu Sep 5 10:26:41 2019 +0200

Done with some tests

commit cf71ae2e264f36053833f4d817cda75fac89619d

Author: Tawanda <tawandamuzanenhamo14@gmail.com>

Date: Tue Sep 3 12:29:47 2019 +0200

Wind generation file added

commit 21ecd947e95470d0e68b8ad174d3be1821895007

Author: Tawanda <tawandamuzanenhamo14@gmail.com>

Date: Tue Sep 3 01:59:26 2019 +0200

Calling it a night

commit 02d8e0f29005a1378336704be79b60ed14e01da3

Author: Tawanda <tawandamuzanenhamo14@gmail.com>

Date: Tue Sep 3 01:35:15 2019 +0200

added text files

commit 514d14cca7ddc464df015f75542ab8249ca321d8

Author: Tawanda <tawandamuzanenhamo14@gmail.com>

Date: Tue Sep 3 01:34:18 2019 +0200

13 hours later

commit b553683c574750603c525f6fab0db9b3ac707bcb

Author: Tawanda <tawandamuzanenhamo14@gmail.com>

Date: Tue Sep 3 00:50:15 2019 +0200

Almost there

commit 26cb5da1c53cd7a8132af5d0d5173cebff51bfd3

Author: Tawanda <tawandamuzanenhamo14@gmail.com>

Date: Mon Sep 2 22:59:52 2019 +0200

I'm about to CRY