|  |
| --- |
| Coventry University |
| Big Data Programming Project |
| 5011CEM |

|  |
| --- |
| Gurvinder Nagra  Student ID: 1024278 |

Table of Contents

[Introduction 2](#_Toc101182977)

[Code report 2](#_Toc101182978)

[Parallel and Sequential comparison 3](#_Toc101182979)

[Estimations of comparison 4](#_Toc101182980)

[Summary and recommendations 4](#_Toc101182981)

[Specification Items 5](#_Toc101182982)

[Gantt Chart 6](#_Toc101182983)

[Flow charts 7](#_Toc101182984)

[Code 15](#_Toc101182986)

[Log Book 24](#_Toc101182987)

[References 31](#_Toc101182988)

[GitHub Link 31](#_Toc101182989)

# Introduction

This report provides an overview on the investigation of Big Data techniques that will be used to reduce the processing time for the analysis of atmospheric science and climate data. Our client has current requirements of taking 2.5 hours to analyse 1 worth of data and they have 25 hours of data per day to analyse and they have over 9TB of data to analyse. They wish to analyse the data of a 24-hour period (25 data subsets) and wish to analyse 2 hours and when doing so we should only load 1 hour worth of data at a time.

We have been asked to show how we can investigate the use of parallel processing to show how it is faster than sequential processing. From that we are to extrapolate the times that we have taken and be able to learn about the different memory constraints we will have and what we would recommend for the client to use. But also use the extrapolated results to show how parallel is faster than sequential. Furthermore, we expect to run automate tests to check for different errors as well as let our client check results after leaving the tests running.

This report will contain explanation of the code used to run the analysis as well as showcasing the difference between parallel and sequential processing and conclusions and results that we will give the client. This will also be supported through the use of figures such as graphs and flowcharts. Furthermore, we will make sure to give recommendations to the client about how they can run this analysis and any other steps and considerations they may need to make.

# Code report

The code we have been working with has been designed to make use of reusable functions, these help use automate our testing as we can call a single script which will call all the functions. The functions themselves will take in input arguments which will be passed in through to the processing, this allows the client to change certain variables such as number of workers used. By being able to change variables like this we are able to test the code in different ways such as comparing speeds by using different number of workers, as well as changing the amount of data we process and how many hours’ worth we do it for. This allows us to only test what we need to test instead of having to test all the data and we can instead just use a segment of the data to compare to. The code itself is organised so that it will firstly run through the testing so that it checks if there are any issues with it and will either prevent the code from running or skip past corrupted data. This allows us to ensure that the analysis will be correct and not halted at any point, which would be an issue if we are deep into it, and we get an error.

The testing that the code does is making sure to check if there are any text errors before we start anything else in the code and if there are then the code will not run from this point due to such errors. As well as that we also test for NaN errors each time we load the data, if a NaN error is found, the code will simply have it, so it loads another hour of data without errors and skip past the data with errors.

The different sections for the code are divided into the main script. The main script contains the different function calls, the first 2 function calls being for Sequential and Parallel processing. These functions set worker and data parameters for both the sequential and parallel processing and pass those parameters into the analysis itself so we can control how many times the code process and for parallel how many workers it will use or iterate through. This will then output those results which we then pass back into the main script and pass into a plotting functions, this will then take those results and plot the results that we have got from both parallel and sequential processing and give us graphs we can use to show how long our analysis takes

# Parallel and Sequential comparison

The differences between parallel and sequential processing are that unlike sequential processing, parallel processing makes use of workers, processors. This allows the analysis to happen much faster as it is able to make use of more processing power as well as increased memory per worker. As you increase the number of processors the speed at which parallel processing occurs will also increase. With this the data is analysed a lot faster and we do not have to wait as long.

Sequential processing on the other hand will process data in the model as it is received making it much slower as it takes longer to process each piece of data bit by bit. From our results below we have extrapolated and shown the difference in speed for both parallel and sequential processing and as well as that as we prove how we get faster speeds as we increase the number of workers.

When doing our comparison, we had created plots using our plot function that we had created which gave us the results we have explained earlier.

In figure 1 we can see the difference in parallel and sequential and how as we increased the number of processors used the time taken for parallel decreased and how sequential was slow than parallel overall.

A picture containing chart

Description automatically generated Figure 1

# Estimations of comparison

From our results we have made estimations on how many workers would be required to complete this within under 2 hours. We have done this by calculating the amount of time that it would take us for only 1000 data and from that we used the results to figure out how long it would take for all the data which was 58 hours, we had used 4 workers in our estimation and we need to finish this in under 2 hours so we would need this to be about 30 times faster meaning we would need 30 times as many workers. This means we would need 120 workers to process the entire data in under 2 hours. As well as that using resource monitors, we discovered that each worker used about 275mb meaning each worker would need that much memory dedicated to it, for 4 workers the total amount of memory needed 1100mb (1.1 GB). From this we can estimate we need about 33000mb (33 GB) of memory to be able to analyse the data within that much time.

Below are the calculations that were made

4 workers= 58 hours

58 rounded up is 60 hours

60 hours/2 hours= 30 times speed difference

4\*30=120 workers

Also, in the figures below for the amount of data we had processed we had extrapolated our results on the graph and how for 5 hours’ worth and 1000 data we would need 12 processors for the fastest time, though this isn’t the amount of data the client wants us to analyse for, we do learn that there is a trend in the amount of processors increased and the time taken being lower, backing up the analysis.

Chart, line chart

Description automatically generatedTable

Description automatically generated

# Summary and recommendations

Overall, from our results we have proven that parallel process will be much quicker then sequential is when analysing the data and as we increase the number of workers the time for the analysis will become much quicker. We have also used the results produced from the time taken to show estimate how many workers we will need for the entire dataset and how much memory we would need for all of the workers. Recommendations for the client would be to ensure they have a processor that has over 120 cores or get a processor with over 60 that has hyper threading so they can simulate 120 cores, allowing them to get the correct number of workers. It will be advisable that the biggest processors used in typically HPC, come with 64 cores, 128 cores or 256 cores. So, taking 64 cores or 128 cores will mean that the device used for the processing will not be able to do much else since the processor will be heavily used meaning it will be advisable to just leave the analysis running for 2 hours and not use the device for anything else. Another recommendation will be to ensure there is plenty of memory, there will be needed at least 33 GB, but it will also be good to have plenty of buffer memory for the system. As well as that the client will need to make sure the speed of the memory is adequate, for example the memory used for this analysis ran at 2667mhz, the client could use faster memory which may slightly accelerate results, but since it is mainly dependant on workers a significant increase is not to be expected.

Other recommendations we can make for the client can be if there are other types of errors within the data such as -9999 as an error code then they will need to implement a way to check for such an error as the code only checks for text and NaN errors.

# Specification Items

When working on this project, we came up with a list of specification items that would be needed to help us plan what we are doing throughout the module.

MATLAB: We will need to ensure the code runs on MATLAB version r2020b and runs across windows and since it uses backslashes for file paths it won’t be compatible with Linux, it will require the parallel processing toolbox to run properly. As well as that we will need to make sure it is set to the correct folder from which we pull and push to Git from to ensure we have the right files, throughout we will have a single script which calls all of ours functions which will run our code.

The code will run in under 2 hours: We will be processing a segment of 1000 data from a single file and not be measuring it against anything else, the hardware used for this will be an i5-1135G7 (4 cores, 8 threads) and using 8GB of DDR4 RAM (2667mhz) and running from an SSD. It will be measured over a few runs to get appropriate results that we will extrapolate to find out how we can do it in under 2 hours. We will not use the computer since this could skew the results as processing power may directed elsewhere if used.

The code will check for errors: We will be checking for 2 types of errors, NaN and text errors. For text errors we will be checking it at the very beginning before any processing is run, as for NaN we will check it for each hour we process in the data so we can keep processing and just skip hours. If there are errors in text the code will not run.

The code will plot results: Our code will also plot results; it will take a result from sequential processing time and results from parallel processing. For sequential it will plot the total time taken for the processing. For parallel it will plot time taken for each worker, in this case 4 workers so there will be 4 plots for parallel processing. The plots will be measured in the number of seconds that were taken for the processing. It will output these results using a standard plot graph, there will also be lines of best fit added to see the trend in parallel processing. We cannot automatically save the plot and we must make sure to manually save it into our directory.

The code will log information: As the code is running it will also log information, this is done when we are NaN testing, the log file will record the name of the file it is testing and every time it is checking through the file making sure to record date and time. The file generated will be stored in our main directory where we are working for easy access. From this we can see what point an error occurs if there is an error, so we know what part of the data is corrupt. The log file itself will be saved in a .txt file format and it will be writable too from the code as it will clear and write on the file each time it is run.

# Gantt Chart

Chart, waterfall chart

Description automatically generated

# Flow charts

Diagram

Description automatically generated Parallel Processing flow chart

Diagram

Description automatically generated Sequential Processing flow chart

Diagram

Description automatically generatedMemory saving.m flow chart

A picture containing text, indoor

Description automatically generated

Parallel for loop flow chart

A picture containing text, indoor

Description automatically generated Sequential processing for loop flow chart

Diagram

Description automatically generated Plots flow chart

Diagram

Description automatically generated

Text tester flow chart

# Diagram Description automatically generated

NaN tester flow chart

# Code

MemorySaving.M

%% This script allows you to open and explore the data in a \*.nc file

clear all % clear all variables

close all % close all windows

FileName = 'C:/Users/Gurvinder Nagra/Documents/GitHub/5011CEM2022NagraG4/Model/o3\_surface\_20180701000000.nc'; % define the name of the file to be used, the path is included

Contents = ncinfo(FileName); % Store the file content information in a variable.

[TestTextChecker]=TestText(FileName);

if TestTextChecker==false

[ResultsSeq] = SequentialForLoop(FileName);

%disp(ResultsSeq)

[ResultsPar] = ParallelForLoop(FileName);

Plots(ResultsPar,ResultsSeq)

end

TestText.m

%% Script to examine NetCDF data formats and check for non-numeric values (chars only)

function [TestTextChecker] = TestText(FileName)

close all

%% Define plain text variable types

DataTypes = {'NC\_Byte', 'NC\_Char', 'NC\_Short', 'NC\_Int', 'NC\_Float', 'NC\_Double'};

TestTextChecker=false;

Contents = ncinfo(FileName); % Store the file content information in a variable.

FileID = netcdf.open(FileName,'NC\_NOWRITE'); % open file read only and create handle

for idx = 0:size(Contents.Variables,2)-1 % loop through each variable

% read data type for each variable and store

[~, datatype(idx+1), ~, ~] = netcdf.inqVar(FileID,idx);

end

%% display data types

DataInFile = DataTypes(datatype)'

%% find character data types

FindText = strcmp('NC\_Char', DataInFile);

%% print results

fprintf('Testing file: %s\n', FileName)

if any(FindText)

fprintf('Error, text variables present:\n')

TestTextChecker=true;

else

fprintf('All data is numeric, continue analysis.\n')

TestTextChecker=false;

end

TestNaNErrors.m

function [TestNan] = TestNanErrors(FileName)

%% Script to examine NetCDF data formats and check for NaN

% Note, you would carry out this test each time you load data.

% You should NOT test the whole file at the start

close all

%% Test File with Errors

NaNErrors = 0;

%% Set file to test

%%DataFileName = 'C:\Users\Gurvinder Nagra\Downloads\4c6ede5b8c139ad36720fb4bb4a98e5dmodel/Model/TestFileNaN214.nc'; % define our test file

%%FileName = 'C:\Users\Gurvinder Nagra\Downloads\4c6ede5b8c139ad36720fb4bb4a98e5dmodel/Model/o3\_surface\_20180701000000.nc'; % un rem this line to see what happens with good data

Contents = ncinfo(FileName); % Store the file content information in a variable.

%% Create and open log file

LogFileName = 'C:\Users\Gurvinder Nagra\AnalysisFile.txt';

% create new log file, 'w' replaces the file if present. To continually

% append, use 'a'

LogID = fopen('AnalysisFile.txt', 'a');

if LogID < 0

fprintf('Failed to open myfile');

end

%%LogID= fopen('AnalysisLog.txt', 'a');

fprintf(LogID, '%s: Starting analysis of %s\n', datestr(now, 0), FileName);

StartLat = 1;

StartLon = 1;

fprintf('Testing files: %s\n', FileName)

for idxHour = 1:25

for idxModel = 1:8

Data(idxModel,:,:) = ncread(FileName, Contents.Variables(idxModel).Name,...

[StartLat, StartLon, idxHour], [inf, inf, 1]); % 'inf' reads all the data

end

% check for NaNs

if any(isnan(Data), 'All')

NaNErrors = 1;

%% display warning

fprintf('NaNs present\n')

ErrorModel = find(isnan(Data), 1, 'first');

TestNan=true;

%% find first error:

fprintf('Analysis for hour %i is invalid, NaN errors recorded in model %s\n',...

idxHour, Contents.Variables(ErrorModel).Name)

% Write to log file

fprintf(LogID, '%s: %s processing data hour %i\n', datestr(now, 0), 'NaN Error', idxHour);

else

% write to log file

TestNan=false;

fprintf(LogID, '%s: %s processing data hour %i\n', datestr(now, 0), 'Success', idxHour);

end

end

fclose(LogID);

if ~NaNErrors

fprintf('No errors!\n')

end

SequentialForLoop.m

function [Results] = SequentialForLoop(FileName)

Options = (1);

for idx = 1:length(Options)

LoopParameter = Options(idx);

[TestNan]=TestNanErrors(FileName);

if TestNan==false

[RunTimeSeq]=SequntialProcessing(LoopParameter,FileName);

Results= [Options(idx), RunTimeSeq];

else

end

end

SequentialProcessing.m

function [RunTimeSeq]=SequntialProcessing(LoopParameter,FileName)

%% This script allows you to open and explore the data in a \*.nc file

close all

%%FileName = 'C:/Users/Gurvinder Nagra/Documents/GitHub/5011CEM2022NagraG4/Model/o3\_surface\_20180701000000.nc';

Contents = ncinfo(FileName);

TestNanErrors(FileName);

Lat = ncread(FileName, 'lat'); % load the latitude locations

Lon = ncread(FileName, 'lon'); % loadthe longitude locations

Segments= 1000;

IncrementsProcessed = 1;

%% Processing parameters provided by customer

RadLat = 30.2016; % cluster radius value for latitude

RadLon = 24.8032; % cluster radius value for longitude

RadO3 = 4.2653986e-08; % cluster radius value for the ozone data

%% Cycle through the hours and load all the models for each hour and record memory use

% We use an index named 'NumHour' in our loop

% The section 'sequential processing' will process the data location one

% after the other, reporting on the time involved.

StartLat = 1; % latitude location to start laoding

NumLat = 400; % number of latitude locations ot load

StartLon = 1; % longitude location to start loading

NumLon = 700; % number of longitude locations ot load

tic

for NumHour = 1: IncrementsProcessed % loop through each hour

fprintf('Processing hour %i\n', NumHour)

DataLayer = 1; % which 'layer' of the array to load the model data into

for idx = [1, 2, 4, 5, 6, 7, 8] % model data to load

% load the model data

HourlyData(DataLayer,:,:) = ncread(FileName, Contents.Variables(idx).Name,...

[StartLon, StartLat, NumHour], [NumLon, NumLat, 1]);

DataLayer = DataLayer + 1; % step to the next 'layer'

end

% We need to prepare our data for processing. This method is defined by

% our customer. You are not required to understand this method, but you

% can ask your module leader for more information if you wish.

[Data2Process, LatLon] = PrepareData(HourlyData, Lat, Lon);

%% Sequential analysis

t1 = toc;

t2 = t1;

for idx = LoopParameter: Segments %size(Data2Process,1) % step through each data location to process the data

% The analysis of the data creates an 'ensemble value' for each

% location. This method is defined by

% our customer. You are not required to understand this method, but you

% can ask your module leader for more information if you wish.

[EnsembleVector(idx, NumHour)] = EnsembleValue(Data2Process(idx,:,:,:), LatLon, RadLat, RadLon, RadO3);

% To monitor the progress we will print out the status after every

% 50 processes.

if idx/50 == ceil( idx/50)

tt = toc-t2;

fprintf('Total %i of %i, last 50 in %.2f s predicted time for all data %.1f s\n',...

idx, size(Data2Process,1), tt, size(Data2Process,1)/50\*25\*tt)

t2 = toc;

end

end

T2(NumHour) = toc - t1; % record the total processing time for this hour

fprintf('Processing hour %i - %.2f s\n\n', NumHour, sum(T2));

end

tSeq = toc;

RunTimeSeq=tSeq;

fprintf('Total time for sequential processing = %.2f s\n\n', tSeq)

ParallelForLoop.m

function [ResultsPar] = ParallelForLoop(FileName)

DataOptions = (1);

WorkerOptions = [1,2,3,4];

ResultsPar = [];

for idx1 = 1:length(DataOptions)

DataParameter=DataOptions(idx1);

for idx2=1:length(WorkerOptions)

WorkerParameter=WorkerOptions(idx2);

[TestNan]=TestNanErrors(FileName);

if TestNan==false

[RunTime]=ParallelProcessing(FileName, DataParameter,WorkerParameter);

ResultsPar=[ResultsPar;RunTime];

else

end

end

end

ParallelProcessing.M

function [RunTime]=ParallelProcessing(FileName,DataParameter,WorkerParameter)

%% 1: Load Data

close all

%%FileName = 'C:/Users/Gurvinder Nagra/Documents/GitHub/5011CEM2022NagraG4/Model/o3\_surface\_20180701000000.nc';

Contents = ncinfo(FileName);

%TestNanErrors(FileName);

Lat = ncread(FileName, 'lat');

Lon = ncread(FileName, 'lon');

NumHours = 1;

%% 2: Processing parameters

% ## provided by customer ##

RadLat = 30.2016;

RadLon = 24.8032;

RadO3 = 4.2653986e-08;

StartLat = 1;

NumLat = 400;

StartLon = 1;

NumLon = 700;

%% 3: Pre-allocate output array memory

% the '-4' value is due to the analysis method resulting in fewer output

% values than the input array.

NumLocations = (NumLon - 4) \* (NumLat - 4);

EnsembleVectorPar = zeros(NumLocations, NumHours); % pre-allocate memory

%% 4: Cycle through the hours and load all the models for each hour and record memory use

% We use an index named 'NumHour' in our loop

% The section 'parallel processing' will process the data location one

% after the other, reporting on the time involved.

Num2Process = 1000;

Steps = 50;

tic

for idxTime = 1:NumHours

%% 5: Load the data for each hour

% Each hour we read the data from the required models, defined by the

% index variable. Each model data are placed on a 'layer' of the 3D

% array resulting in a 7 x 700 x 400 array.

% We do this by indexing through the model names, then defining the

% start position as the beginnning of the Lat, beginning of the Lon and

% beginning of the new hour. We then define the number of elements

% along each data dimension, so the total number of Lat, the total

% number of Lon, but only 1 hour.

% You can use these values to select a smaller sub-set of the data if

% required to speed up testing o fthe functionality.

DataLayer = 1;

for idx = [1, 2, 4, 5, 6, 7, 8]

HourlyData(DataLayer,:,:) = ncread(FileName, Contents.Variables(idx).Name,...

[StartLon, StartLat, idxTime], [NumLon, NumLat, 1]);

DataLayer = DataLayer + 1;

end

%% 6: Pre-process the data for parallel processing

% This takes the 3D array of data [model, lat, lon] and generates the

% data required to be processed at each location.

% ## This process is defined by the customer ##

% If you want to know the details, please ask, but this is not required

% for the module or assessment.

[Data2Process, LatLon] = PrepareData(HourlyData, Lat, Lon);

%% Parallel Analysis

%% 7: Create the parallel pool and attache files for use

PoolSize = WorkerParameter ; % define the number of processors to use in parallel

if isempty(gcp('nocreate'))

parpool('local',PoolSize);

end

poolobj = gcp;

% attaching a file allows it to be available at each processor without

% passing the file each time. This speeds up the process. For more

% information, ask your tutor.

addAttachedFiles(poolobj,{'EnsembleValue'});

% %% 8: Parallel processing is difficult to monitor progress so we define a

% % special function to create a wait bar which is updated after each

% % process completes an analysis. The update function is defined at the

% % end of this script. Each time a parallel process competes it runs the

% % function to update the waitbar.

DataQ = parallel.pool.DataQueue; % Create a variable in the parallel pool

%

% % Create a waitbar and handle top it:

%hWaitBar = waitbar(0, sprintf('Time period %i, Please wait ...', idxTime));

% % Define the function to call when new data is received in the data queue

% % 'DataQ'. See end of script for the function definition.

%afterEach(DataQ, @nUpdateWaitbar);

%N = Num2Process/Steps; % the total number of data to process

%p = 1; % offset so the waitbar shows some colour quickly.

%% 9: The actual parallel processing!

% Ensemble value is a function defined by the customer to calculate the

% ensemble value at each location. Understanding this function is not

% required for the module or the assessment, but it is the reason for

% this being a 'big data' project due to the processing time (not the

% pure volume of raw data alone).

T4 = toc;

%PARFOR=250;

parfor idx = DataParameter: Num2Process % size(Data2Process,1)

[EnsembleVectorPar(idx, idxTime)] = EnsembleValue(Data2Process(idx,:,:,:), LatLon, RadLat, RadLon, RadO3);

if idx/Steps == ceil(idx/Steps)

send(DataQ, idx/Steps);

end

end

%close(hWaitBar); % close the wait bar

T3(idxTime) = toc - T4; % record the parallel processing time for this hour of data

fprintf('Parallel processing time for hour %i : %.1f s\n', idxTime, T3(idxTime))

end % end time loop

T2 = toc;

RunTime=sum(T3);

delete(gcp);

%% 10: Reshape ensemble values to Lat, lon, hour format

EnsembleVectorPar = reshape(EnsembleVectorPar, 696, 396, []);

fprintf('Total processing time for %i workers = %.2f s\n', PoolSize, sum(T3));

%% 11: ### PROCESSING COMPLETE DATA NEEDS TO BE SAVED ###

function nUpdateWaitbar(~) % nested function

waitbar(p/N, hWaitBar, sprintf('Hour %i, %.3f complete, %i out of %i', idxTime, p/N, p, N));

p = p + 1;

end

workspace;

end % end function

Plots.m

function Plots(RunTime,RunTimeSeq)

%% Plotting graphs in Matlab

close all

%% Show two plots on different y-axes

%% 250 data processed

y1Vals = (RunTime);

figure(1)

yyaxis left

plot(y1Vals, '-bd')

xlabel('Number of Processors')

ylabel('Processing time (s)')

title('Processing time vs number of processors')

%% 5,000 data processed

disp(RunTimeSeq)

a=RunTimeSeq(2);

y2Vals = a;

figure(2)

yyaxis left

plot(y2Vals, '-rx')

xlabel('Number of Processors')

ylabel('Processing time (s)')

title('Processing time vs number of processors')

%% Show two plots on same y-axis

%% Mean processing time

y1MeanVals = y1Vals;

y2MeanVals = y2Vals;

figure(3)

plot(y1MeanVals, '-bd')

hold on

plot(y2MeanVals, '-rx')

xlabel('Number of Processors')

ylabel('Processing time (s)')

title('Processing time vs number of processors')

legend('Parallel', 'Sequential')

# Log Book

Logbook

Week 1

Tuesday

11am: Turned up to first class

11:10am-12:10am: Went through introduction PowerPoint and introduction to the module itself and what we will be doing through the coming weeks

12:10am-1pm: Began work on checking through the task and any PowerPoints that were on Aula

Wednesday

2:30pm: Got home downloaded software need to create a Gantt chart and began checking through each different week on aula

2:45pm-3:30pm: Began creating Gantt chart for the first 4 weeks and identifying times needed

3:30pm-4:17pm: Began installing MATLAB and anything else to depend on

Week 2

Tuesday

9am: Turned up to class and got set up on laptop, began checking through tasks.

9:10am: Listened to what the work was about

9:23am: Continued looking through the task and begin understanding how the work was supposed to be done

9:34am-10:40am: Completed work on All the tasks for this week, collaborated with course mate when I was stuck on anything throughout making sure to take advice but not to copy any work.

10:40am-11am: Began more work on the Gantt chart and continued through it too week 7.

11am: Headed to the library to get more work done

11:15am-11:40am: Resumed and completed the Gantt chart

11:40am: Double checking my code that I had done earlier, and a course mate pointed out a mistake and helped me correct it

12am: Finished work and went on lunch break before next class

Week 3

Tuesday

9am: Arrived at class

9:10am: Went through anything live lecture material

9:30am: Began checking through any non-live lecture material

9:43am: Began checking through the tasks and got started on them

9:56am: Started the sequential processing task

10:11am: Consulted with course mate on whether what I was thinking was appropriate for the task and how else it may be possible to do the task

10:23am: Finished changes for Task 2 of the sequential processing task

10:27am: Started going through task 3 and got stuck on some points

10:38: Reading through the sequential processing code and making sense of how it would work with the task and what areas I needed to modify

11am: Left Class and headed to the library to continue on work

11:12am: Continued working on task 3

11:52am: consulted with course mate and confirmed I finished the sequential processing correctly.

11:57am: Began checking through the parallel processing task and started task 1

12:07am: Went for Lunch break

Wednesday

2:19pm: Started sequential processing and left it to run as I went to go do a daily workout as I left it running

4:31pm: Returned to see sequential processing had completed and took results

Week 4

Tuesday

9am: Arrived at class and went through anything live lecture

9:21am: Checked through the flow chart tasks and non-live lecture material given

9:34am: Began work on flow charts

10:13am: Finished work on flow charts

10:17am: Resumed work on parallel processing task and completed the modifications

10:30am: Checked the next task on what do to do for the week 3 tasks

10:34am: Took a rest for the rest of the class

11am: Headed to library and began the parallel processing on my machine

11:39am: Finished the processing after doing it multiple times and having to double check my code when comparing to what my course mate did and guiding each other on thing’s we thought we could change.

11:45am Extrapolated results and data we got from processing

12am: Went to take a lunch break

Week 5

Tuesday

9am: Arrived too class

9:14am: Began going through materials and tasks

9:31am: Began SMART specifications tasks

9:45am: Began discussing with course mate how to go about doing the task and what we should and shouldn’t include

9:54am: Continued doing the task

10:27am: Had colleague check through task and I checked through theirs

10:43am: Asked lecturer what could be done to further improve the task and began on those improvements

10:55am Finished improvements and left class for a break and to get on with work for other modules.

Week 6

Tuesday

9am: Arrived at class

9:11am: Began work on checking through lectures

9:21am: Began break testing task making sure to add missing code files into the correct areas

9:34am: Completed task 1 and made multiple code files some of which had to be deleted due to having too many and struggling to make them again

9:45am Got onto next part of the task

9:54am: Began googling how some MATLAB code in the task worked and learning what it does as well as how to do a few other things.

10:03am: Began work on task 2

10:16am: Finished Task 2 got onto task 3

10:18: Confirmed with course mate I was right that task 3 was just reading the task

10:20am: Finished Task 3 and moved onto task 4

10:44am: Finished task 4 and moved onto advanced task 3

10:52am: Finished advanced task 3 and left for the library

11:03am: Arrived at the library and moved onto the next task

11:33am: Decided to take a break from the task after struggling to understand how it worked

Week 7

Tuesday

7am: Arrived at library

7:10am: Began work on tasks for week 6 again

7:23am: After attempting again started re reading tasks to grasp a better understanding of what to do for the tasks

7:45am: After making some progress decided to get some breakfast before class

9am: Arrived at class and began going through assessment guidelines and making notes and planning what I need to do

9:34am: Started going through the sample report and checking what I would need to do for mine

9:45am: Continued on with the testing tasks

10:13am: Began checking with course mates what we were doing wrong with the task and began troubleshooting it and re trying what we were doing

11am: Moved to the library

11:10am: Continued on with the task, still struggling and making little progress

12am: Decided to take a break from the task

Wednesday

7am: Arrived at library

7:15am: Continued with task and consulting with colleague

7:45am: completed and started testing out 1 implementation

7:50am: saw it wasn’t working correctly and tried thinking of another way to implement it

8:45am: Finished another implementation and it seemed to be working fine

6pm: Got a message from colleague that the implementation was wrong after they went to a programming support sessions

Week 8

Monday

9am: Taking advice from colleague re did the implementation so that it was correct

9:45am: Decided to begin re organising the code so it was set into proper re-usable functions

10:47am: Finished organising the main code and began planning on how to add the testing functions in

11:03am: Began work on adding the testing functions into the code

11:45: Completed the testing functions that needed to be added to the code

12am: Took a lunch break

1:18pm: Returned to change the amount of processing being done in the code and began running analysis

1:48pm: After running multiple analysis confirmed code so far was correct

Tuesday

7am: Arrived at library and began helping course mates guiding them on how to do the testing and create the functions so they are properly reusable throughout the code

7:45: Finished guiding colleagues and began planning on how to implement graphing code

8am: Began testing the graphing code we already have and thinking of different ways we could do it

8:30am: Went to get some food and drink before class

9am: Arrived at class

9:15am: Worked on implementation of the graphing function

10:03am: Tested the code

10:45: Asked lecturer on what to do for graphing and trying to understand what to do after many tests and learned the changes that had to be made for the code

11am: Left for the library and began creating a function for graphing

11:30am: Completed the function and tested it with small values and got the appropriate results

1140am: Began work on the PowerPoint for the Viva and checking all the requirements and creating the appropriate template

12am: Took a lunch break

Friday

8am: Began writing up presentation for the Viva

8:30am: Began reading through and comparing to the brief on what else I needed

8:45am: Double checked the brief as I was writing

9:26am: Finished written content on the PowerPoint slides

9:30am: Took a break

10am: Began writing some general notes and planning on what I would be saying throughout the PowerPoint and did some mock runs

11am: Left for driving Lesson and Lunch

Week 9

Tuesday

9am: Arrived at class

9:10am: Continued work on viva and making sure double checking the Viva

9:33am: Began checking the report guidelines

9:45am: Began writing the report

10:23am: Did a comparison with colleagues on the general outline of what we were doing for the report and Viva and gave some advice to each other

10:36am: Checked guidelines again as writing the report

11am: Left class for a break

Week 10

Tuesday

9am: Arrived at class

9:10am: Started checking the viva and adding some improvements

9:19am: Began running analysis to get supported evidence needed in the PowerPoint

9:47am: Began adding the evidence and extrapolation of results into the PowerPoint

10:23am: Checked the general outline of what we were doing for the Viva, making sure it was finished and returned to the report

10:39am: Checked report guidelines once more as writing and used Viva to aid what I was writing

11:03am: Left class for a break

Week 11

Monday

8am: Got to work on double checking the viva and having made sure I knew what to say

8:17am: Began setting up OBS for recording

8:28am: Began recording

8:34am: Finished recording

8:41am: Checked recording was not taking in audio properly

8:43am: Tweaked OBS settings and re did the recording

8:50am: Tried to submit the recording and found out file size was too large

8:57am: Used a special tool to reduce the file size but made sure the recording was still fine

9:06am: Submitted the recording

Tuesday

9am: Arrived at class

9:07am: Continued working on report

9:30am: Talked to colleague about how they were going about it

10:07am: took a short breaking from writing the report

10:19am: Resumed writing

10:45: Asked lecturer some questions about the report

11am: Left for a break

Week 12:

Tuesday

9am: Arrived at class

9:06am: Continued on with report

10:42am: Asked lecturer some questions on what I was doing with some issues I had and the report and continued on with it

11am: Left for a break

Friday

8am: Continued work on report

9am: Ensured all flow chart and figures were placed into the table

9:31am: Ensured GitHub was correct

9:43am: Pasted Logbook into report

# References

MATLAB. (n.d.). *parfor.* Retrieved from MATLAB Documentation: https://uk.mathworks.com/help/parallel-computing/parfor.html

Rajpurohit, A. (2017, April 17). Must-Know: When can parallelism make your algorithms run faster? When could it make your algorithms run slower? Retrieved from kdnuggets: https://www.kdnuggets.com/2017/04/must-know-parallelism-algorithms.html

MATLAB. (n.d.). *Length.* Retrieved from MATLAB: https://www.mathworks.com/help/matlab/ref/length.html?searchHighlight=length&s\_tid=srchtitle\_length\_1

MATLAB. (n.d.). *Disp.* Retrieved from MATLAB Documentation: <https://www.mathworks.com/help/matlab/ref/disp.html>

(European Centre for Medium-Range Weather Forecasts, n.d.)

# GitHub Link

<https://github.coventry.ac.uk/5011CEM-2122JanMay/5011CEM2122_nagrag>