MONASH University
Information Technology

ASSESSMENT COVER SHEET

Student ID number	29743338
Given Name	James
Family name	Schubach

Unit Name and Code:	Parallel Computing - FIT3143		
Campus:	Clayton		
Assignment Title:	ASSIGNMENT 1		
Name of Lecturer:	Christopher Watkins		
Name of Tutor:	Sunny Patel		
Tutorial Day and Time:	Thursday 5pm		
Phone Number:			
Email Address:	jsch0026@student.monash.edu		
Has any part of this assignment been previously submitted as part of another unit/course? ☐ Yes ☑ No			
Due Date: Friday 6th of Sep		Date Submitted: 6/09/19	

All work must be submitted by the due date. If an extension of work is granted this must be specified with the signature of the lecturer/tutor.

Extension granted until (date)___________________ Signature of lecturer/tutor ___________________________

Please note that it is your responsibility to retain copies of your assessments.

Intentional plagiarism or collusion amounts to cheating under Part 7 of the Monash University (Council) Regulations

Plagiarism: Plagiarism means taking and using another person's ideas or manner of expressing them and passing them off as one's own. For example, by failing to give appropriate acknowledgement. The material used can be from any source (staff, students or the internet, published and unpublished works).

Collusion: Collusion means unauthorised collaboration with another person on assessable written, oral or practical work and includes paying another person to complete all or part of the work.

Where there are reasonable grounds for believing that intentional plagiarism or collusion has occurred, this will be reported to the Associate Dean (Education) or delegate, who may disallow the work concerned by prohibiting assessment or refer the matter to the Faculty Discipline Panel for a hearing.

Student Statement:

- I have read the university's Student Academic Integrity Policy and Procedures.
- I understand the consequences of engaging in plagiarism and collusion as described in Part 7 of the Monash University (Council) Regulations http://adm.monash.edu/legal/legislation/statutes
- have taken proper care to safeguard this work and made all reasonable efforts to ensure it could not be copied.
- No part of this assignment has been previously submitted as part of another unit/course.
- I acknowledge and agree that the assessor of this assignment may for the purposes of assessment, reproduce the assignment and:
  - i. provide to another member of faculty and any external marker; and/or
  - ii. submit it to a text matching software; and/or
  - iii. submit it to a text matching software which may then retain a copy of the assignment on its database for the purpose of future plagiarism checking.
- I certify that I have not plagiarised the work of others or participated in unauthorised collaboration when preparing this assignment.

Signature .....JS... Date.06/09/2019....................

* delete (iii) if not applicable

The information on this form is collected for the primary purpose of assessing your assignment and ensuring the academic integrity requirements of the University are met. *Other purposes of collection include recording your plagiarism and collusion declaration, attending to course and administrative matters and statistical analyses. If you choose not to complete all the questions on this form it may not be possible for Monash University to assess your assignment. You have a right to access personal information that Monash University holds about you, subject to any exceptions in relevant legislation. If you wish to seek access to your personal information or inquire about the handling of your personal information, please contact the University Privacy Officer:* privacyofficer@adm.monash.edu.au

Design and Implementation of Parallel Mandelbrot Set Image Generation via OpenMPI

Report prepared by:

James Schubach
Melbourne, Australia
schubach.james@gmail.com

***Abstract*—The Mandelbrot set is a set of numbers in the complex plane. This set is found via iteration. With some basic quadratic polynomials and a seed, the generation of our set can start. The generation of the set has been referred to as being embarrassingly parallel. This means that it is very easy to split the problem into parallel tasks. Hence the purpose behind this report. This report explores the generation of the Mandelbrot set using a parallel scheme, which the work is split up into rows. The parallelisation of the problem is implemented using the C language and a library called OpenMPI which is appart of the C language. OpenMPI allows us to communicate data across different cores or processors. This means that every process is given a row and will iterate over that row till it reaches the max value, this will be called iXmax. The max number of rows will be called iYmax. This parallel scheme along with discussion of tile based will be analysed via Amdahl's law. It was found that the use of a row-based segmentation with a Dynamic master slave approach for distributing the tasks found the greatest improvement in time taken to complete the Mandelbrot image. After analysing the run time of the serial generation of the Mandelbrot set and the run time of the parallel version, it was found that by doubling the number of cores gave an increase of 43% on average. However, from going from serial to 2 cores gave a 89% increase. The results show a significant change in speed from a single core to multi-cores, showing the parallelisation of the Mandelbrot Set was successful.**

Keywords— Mandelbrot set, OpenMPI, MPI, C, parallel computing, distrubuted computing.

I. Introduction

The purpose of this report was to explore the parallelisability of the Mandelbrot set, understand the intricoes of sending data between process and understand how parallel schemes play a part in the speed of computation.

The Mandelbrot set is a set of complex numbers referred to as C, in which to calculate z via the function $Z = Z^2 + C$ [1]. In this case C is representing a constant number in which will not change throughout the iterations. The value of Z itself has no real significance, however the magnitude of the number is the important part of the number. To calculate the magnitude of a complex number is quite hard, so to do this, get the square of the numbers distance to the x-axis and the square of the numbers distance to the y-axis, then add them together and square the result. As the equation is iterated the magnitude of Z changes. The magnitude of Z can do two things as its iterated; it will be equal to or less then two or it will be greater than two forever. If said number is greater then two then it is not a part of the Mandelbrot set. To make the previously said set, you can graph the values on the complex number plane, after plotting 100 to thousands of points a familiar image of the Mandelbrot set starts to generate. To get colour to the said image, the values of the points not in the Mandelbrot set are given colour. [2]

Due to the independent nature of the Mandelbrot Set generation, it was hypothesised that paralleling the serial version of the Mandelbrot Set code via master/slave communication and row-based segmentation of tasks would give a linear increase in speed of the generation.

II. Design and anaysis of design

A. Parallelisability of Mandelbrot Set

For a calculation to be parallelised, it is needed to see if they meet Bernstein's conditions. The three conditions are:

$$I_0 \cap O_1 = \emptyset$$
$$I_1 \cap O_0 = \emptyset$$
$$O_0 \cap O_1 = \emptyset$$

I_o represents the input of the first calculation while O_o represents the output of the first calculation. Then I_1 and O_1 are the input and output for the second calculation. As the parallel scheme is based on rows our constant value will change depedent on row. The z iterations is represented by j and thus for rows i. This give the equation:

$$\text{Row I} = Z_{ij} = Z_{ij-1}^2 + i$$
$$\text{Row I+1} = Z_{i+1j} = Zi_{j-1}^2 + i+1$$
$$\text{Thus } I_o = Z_{ij-1}^2 + i$$
$$I_1 = Zi_{j-1}^2 + i+1,$$
$$O_0 = Z_{ij}$$
$$O_1 = Z_{i+1j}$$

The Z values are independent to its respected rows. Thus Bernsteins conditions have been met. As,

$$I_o \cap O_1 = \emptyset$$

$$I_1 \cap O_0 = \emptyset$$

$$O_0 \cap O_1 = \emptyset$$

Thus, the mandelbrot set has met the conditions it can now be parallelised.

B. *Therotical Speed Up of Mandelbrot Set*

Amdahl's law is an equation that given the right values give a value of such is the theoretical speed up of a given problem. The serial algorithm has 2 parts. (1) Printing the header to the file which does not make sense to do parallel. (2) Calculation of the Z values which from Bernstein's conditions are known to be parallelisable. The printing to the file takes like then a thousandth of a second. Compare this to the bulk execution of the algorithm, the calculation gets a 99.998% of the time taken to execute.

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}} \tag{1}$$

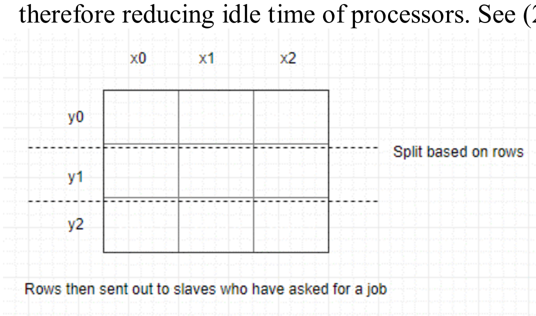
$S_{latency}$ = Speed up
p = Paraellel ratio
s = number of processors

By using Amdahl's law as denoted by (1), the given table 1 is calculated.

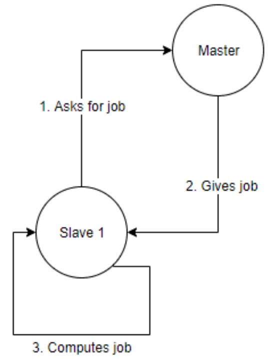
Number of processors (s)	1	2	4
Speed Up ($S_{latency}$)	1.0	2.0	4.0

C. *Partioning Method*

The design of the parallel portioning method was developed by using a Dynamic Load Balancing method along with a row-based segmentation approach. DLB is implemented via a Master/Slave communication. The master sends out jobs to the slaves who have asked for said job see (3). This means that the tasks are broken up into small enough tasks that no one process takes to long to complete said task. Once completed it simply asks for other tasks and the process repeats. By utilizing this method, the rows are not being statically divided to each process. This could potentially give all the hard tasks to a single process, then other process would be idle while a single process finishes its computation. Thus, by sending out jobs when asked for the rows are dynamically allocated to the processors therefore reducing idle time of processors. See (2)

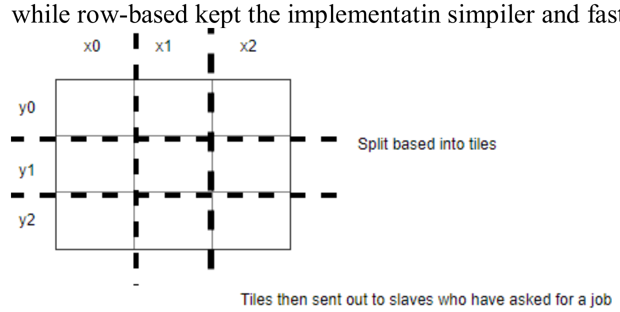


(2)



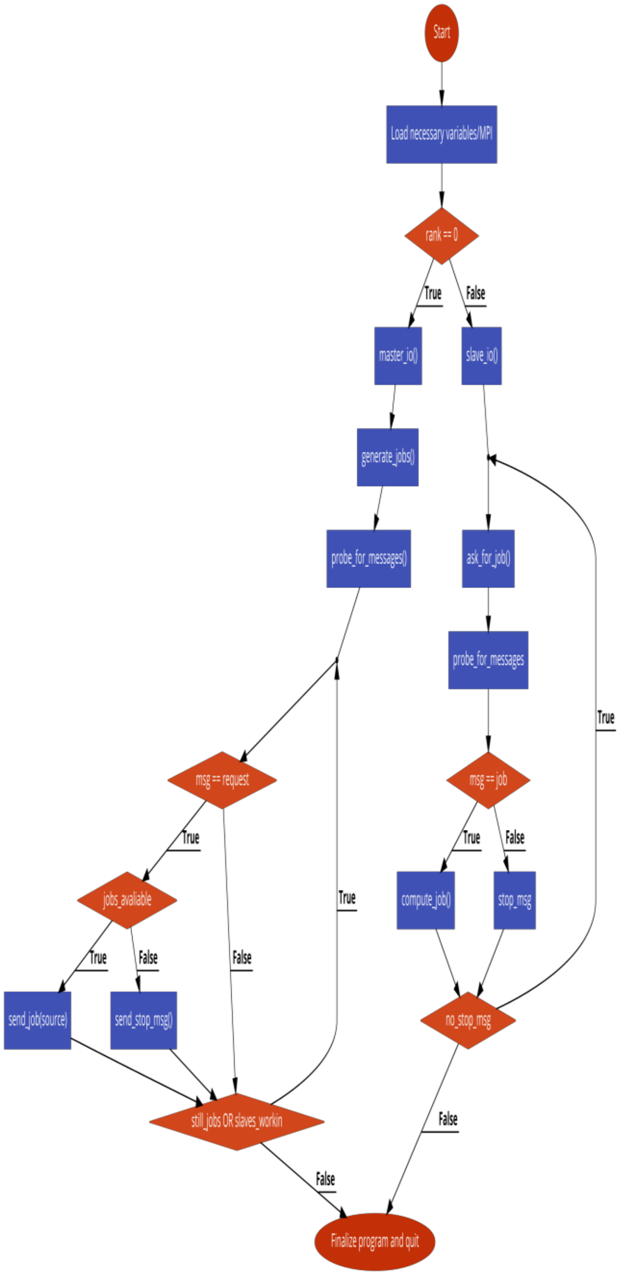
(3)

Another approach at parralising the task is to segment it into tiles see (4) However, this method makes the jobs to small such that the master has trouble keeping up with the requests of jobs. When testing both implementations, with a 8 core system, I found that the time increased from 6 seconds to 8 seconds. This time could be better increased via making the tiles larger, however this method also increased the complexity of the code and the algorithim, while row-based kept the implementatin simpiler and faster.



(4)

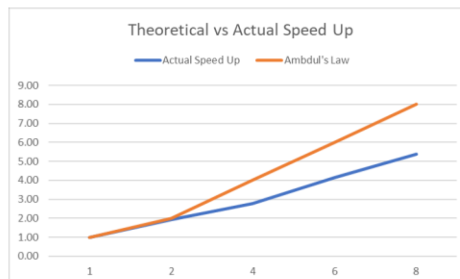
D. Pseudocode of Master/Slave Mandelbrot



III. Results & Discussion

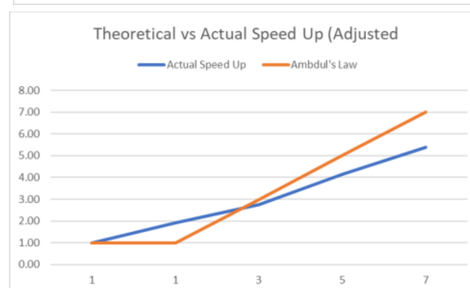
A. Table of Findings

Computer Specifications		a) Intel Core i7-7700k b) 8 Logical Processors c) 16Gb of RAM d) 82 Mbps				
Value of iXmax		8000				
Value of iYmax		8000				
Value of IterationMax		1000				
		Parallel Program				
	Serial Program	MPI				
		1	2	4	6	8
Run #1	37.14	35.27	18.64	13.09	8.61	6.68
Run #2	36.69	35.41	18.89	13.14	8.69	6.77
Run #3	35.75	35.47	19.03	12.98	8.91	6.72
Run #4	36.07	35.42	18.84	13.31	8.59	6.80
Run #5	35.73	36.23	18.67	13.19	9.08	6.76
Average run time	36.28	35.56	18.81	13.14	8.78	6.75
Actual Speed up	3.04	0.98	1.93	2.76	4.13	5.38
Ambdul's Law	4.20	1.00	2.00	4.00	6.00	8.00
Average Theoretical Time	36.28	36.28	18.14	9.07	6.05	4.53
Difference between Actual vs Theoretical		0.28				



(1)

(2)



(3)

B. Disscusion

From the results, you can see a clear increase in speed when implementing the parallel algorithm. However, it does not meet our theoretical analysis of the problem. This however is because of the way the algorithm is designed vs the way the formula works out the theoretical result. As this approach is a master slave algorithm, the algorithm is using 1 core as a master no matter how many cores are added. This means that when the program has access to 6 cores, it is only utilizing 5 cores, which would account for the difference in values from theoretical vs actual speed times. As you can see via the graph (3) when plotted against the adjusted values our speed is much closer to the theoretical amount. The rest of the differences can be purely based on implementation problems due to an inexperienced programmer, CPU interrupts, feedback and a range of other interferences that could alter times.

In terms of scalability, from the graphs and data, it shows a close to linear growth with regards to increase of processors, this gives us strong scalability. If the data didn't suggest that the run time decreased to by a fairly strong amount, then the scalability of the algorithm would be considered weak. These results could be proved further however at the time of testing the MonARCH system was not available, this in turn does not allow the testing of any further calls

IV. Conclusion

The report shows the findings of implementing a row-based segmentation using a Master Slave communication on the Mandelbrot Set. This was implemented with the programming language C as well as a library OpenMPI which allowed the Master Slave Implementation. The results from the algorithm showed that the Mandelbrot set benefits greatly from the use of parallel schemes. It also showed the scheme with the master/slave communication approach scaled greatly when extra processors are used. The small differences between theoretical and actual run time where identified after completing said project.

References

[1] "What is the Mandelbrot set?," 26-Jul-2018. [Online]. Available: https://plus.maths.org/content/what-mandelbrot-set.

[2] D. Dewey, *Introduction to the Mandelbrot Set*, Sep-2002. [Online]. Available: http://www.ddewey.net/mandelbrot/.