MONASH University
Information Technology

ASSESSMENT COVER SHEET

Student ID number	
Given Name	
Family name	

Unit Name and Code:			
Campus:			
Assignment Title:			
Name of Lecturer:			
Name of Tutor:			
Tutorial Day and Time:			
Phone Number:			
Email Address:			
Has any part of this assignment been previously submitted as part of another unit/course? ☐ Yes ☐ No			
Due Date:		Date Submitted:	

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 The material used can be from any source (staff, students or the internet, published and unpublished works).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 ..... Date……

* 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*

Updated: 17 Jun 2014

Design and Implementation of the Mandelbrot Set using Message Passing Interface

Prepared by:

K. Sharsindra Pratheen

School of Information Technology, Monash University.

***Abstract*: The iteration of complex mathematical functions effectuates pulchritudinous fractal figures. It is due to this nature of the Mandelbrot Set that it has garnered enormous acknowledgement outside of the world of Mathematics. This report proposes a partitioning strategy for parallelizing the Mandelbrot Set program with the aid of diagrams and methodical description about the partitioning strategy, namely the Round Robin Row Segmentation-Based Partitioning Scheme. The parallelizing based on the partition scheme is done by utilizing the Message Passing Interface (MPI) library in C Programming. To parallelize the sequential algorithm, we needed to look for operations that can be performed independently of each other, this is where the Bernstein's Conditions were utilized. In addition to that, Amdahl's Law was also used to obtain the theoretical speedup in latency of the parallelizing of the Mandelbrot Set using the Round Robin Row Segmentation-Based Partitioning Scheme. The Program was first run serially on a multi-core computer and was also later run in parallel using the partitioning scheme on a multi-core computer but with different numbers of logical processors. The ramification of the implementation exhibits with evidence, significant acceleration of the program for the proposed Round Robin Row Segmentation-Based Partitioning Scheme using MPI on C Programming as compared to the serial implementation. Hence, the results indicate the triumphant parallelization of the program.**

***Keywords:* Mandelbrot Set, MPI, Parallel Computing, C Programming, Scalability.**

I. INTRODUCTION

Named after its discoverer Mathematician Benoit Mandelbrot [1], The Mandelbrot Set is a fractal. In mathematics, a fractal (see Figure 1) is a subset of Euclidean space for which the Hausdorff dimension [2] strictly exceeds the topological dimension. Fractals tend to retain their form at varying levels, this causes them to be encountered ubiquitously in. nature. Fractals are referred to as self-similar [3] which means they exhibit likewise patterns at increasingly minute scales and are similar to one or more of its parts. The complex patterns of fractals

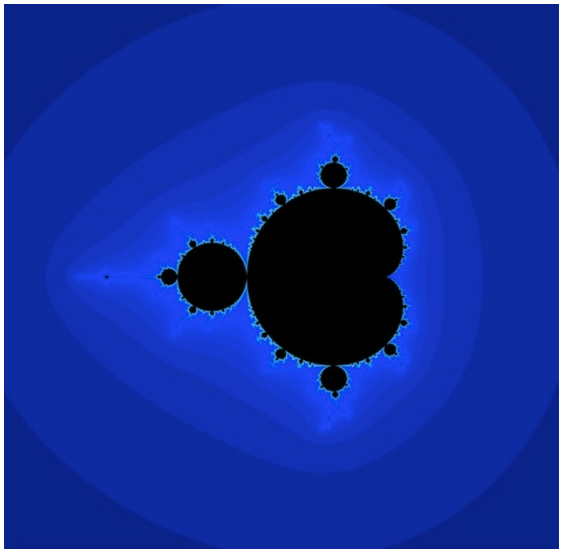

Figure 1. Image of Fractal

are often coloured, resulting in magnificent looking computer generated images. Fractals are able to generate these complex patterns because a fractal is a set consisting of complex numbers and a formula is required to compute numbers in the fractal set. That being said, the Mandelbrot Set is generated by using the complex quadratic polynomial (1). C is a complex parameter, so we start iterating from z = 0 to produce a sequence of values as shown in (2) which will gradually either diverge to infinity or converge to a point recursively depending on the equation. Different function values can be generated by using different iteration numbers.

$$f_c(z) = z^2 + c \quad (1)$$

$$z_0 = 0, z_1 = z_0^2 + c = c, z_2 = z_1^2 + c = c^2 + c\ldots. \quad (2)$$

Generating a Mandelbrot Set is ideal for parallel computing due to each point's calculation being independent of other points. Parallel Computing [4] is a process in which multiple computing resources are used to solve a problem concurrently. In other words, it's an effective method of improving the processing ability and speed of a computer.

The Message Passing Interface (MPI) [5] is a practical, flexible and efficient standard used to generate parallel programs based on message passing. The parallelization of The Mandelbrot Set is brought to life by utilizing the MPI library in C Programming. A Round Robin Row Segmentation-Based Partitioning Scheme is adopted to divide the rows into sub-rows and assigning them various logical processors which will compute them better. The performance of the parallelized program for the partitioning scheme is evaluated by computing the speed-up factor in relation to the operation time of the parallelized program. The analysis is compared with the theoretical values obtained by manipulating Amdahl's Law [6]. A graph is then plot to observe the scalability of the algorithm for an increasing number of logical processors.

II. PRELIMINARY ANALYSIS

A. Parallelizability of Mandelbrot Set

A reliable method to detect parallelism in a sequential algorithm is to find out if the equations of the algorithm can be performed independently of each other. This is where Bernstein's Conditions are used since they can be interchanged without modifying the results of the program. The three conditions that make up the constitution of the Bernstein's conditions are the following:

$$I(S_1) \cap O(S_2) = \phi \quad \text{flow independence}$$

$$O(S_2) \cap O(S_1) = \phi \quad \text{output independence}$$

$$I(S_2) \cap O(S_1) = \phi \quad \text{anti-independence}$$

This shows that S_1 and S_2 can be equated in parallel ($S_1 \parallel S_2$) if $[I(S_1) \cap O(S_2)] \cup [O(S_2) \cap O(S_1)] \cup [I(S_2) \cap O(S_1)] = \phi$ and these conditions surface at different levels of granularity. As far as it is concerned, if these three conditions are met, the processes can be concluded to be independent of each other proving that it is greatly parallelizable.

$$f_c(z+1) = (z+1)^2 + c \quad (3)$$

$$c = x + iy \quad (4)$$

Equations (1) and (3) represent the function calculations of 2 consecutive values of the Mandelbrot set. Hence,

$$I(S_1) = f_c(z) \qquad O(S_1) = z^2 + c$$

$$I(S_2) = f_c(z+1) \qquad O(S_2) = (z+1)^2 + c$$

We then obtain the following three equations by applying the Bernstein's conditions (5)(6)(7).

$$f_c(z) \cap (z+1)^2 + c = \phi \quad \text{flow independence} \quad (5)$$

$$(z+1)^2 + c \cap z^2 + c = \phi \quad \text{output independence} \quad (6)$$

$$f_c(z+1) \cap z^2 + c = \phi \quad \text{anti-independence} \quad (7)$$

The above equations shows the proof of Bernstein's conditions. Therefore, both processes S_1 and S_2 can be executed in parallel, i.e. ($S_1 \parallel S_2$). Therefore, it is implied with proof and evidence that the Mandelbrot Set is parallelizable in C programming while utilizing the MPI library.

B. Theoretical Speed Up of Mandelbrot Set

In computer architecture, the theoretical speedup in latency of the execution of a task at fixed workload can be calculated using the Amdahl's Law. Our aim is to calculate the theoretical speed up of the Mandelbrot Set Algorithm if we were to use multiple logical processors based on the time we obtain while running it on a single processor core. Parallel computing with multiple processors is only beneficial for highly parallelizable programs and since our algorithm is, it's useful. The computational time and the Overall Time was recorded 5 times and an average was obtained for an accurate read up. All the values are reflected in Appendix A in the Appendices.

$$S(p) = \frac{1}{Rs + \frac{Rp}{p}} \quad (8)$$

Rp = (Computational Time/Overall Time)

Rs = 1-(Rp)

p = Number of processors

The above equation and explanation actually reflects Amdahl's Law (8) which can used to calculate the theoretical speed up factor if we were to parallelize our algorithm with a multi-processor core. Table 1 also shows the calculated value for the theoretical speed up factor, S(p) for both 4 logical processors and 8 logical processors.

Number of processors, p	2	4	6	8
Speed Up Factor, S(p)	1.99	3.99	6.06	7.99

TABLE 1

THEORETICAL SPEED UP USING AMDAHL'S LAW

III. DESIGN OF PARTITION SCHEMES

To parallelize the sequential algorithm, we needed to look for operations that can be performed independently of each other, this is where the Bernstein's Conditions were utilized. In addition to that, Amdahl's Law was also used to obtain the theoretical speedup in latency of the parallelizing of the Mandelbrot Set using the Round Robin Row Segmentation-Based Partitioning Scheme. The Program was first run serially on a multi-core computer and was also later run in parallel using the partitioning scheme on a multi-core computer but with different numbers of logical processors.

Round Robin Row Segmentation-Based Partitioning Scheme

In this scheme, a Master and slave implementation is sub-implemented where the Root node becomes the Master and carries out the Receive and Write operations while all the available logical processors are given equal amount of work to gather and send to Master for Mandelbrot Set generation.

$$\textbf{Rows per processor} = \frac{\textbf{Number of Matrix Rows}}{\textbf{Number of Processors}} \quad (9)$$

The rows per processor can be calculated using the equation above (9) where the beginning and last indices would determine the number of iterations that the segmentation would have to occur for.

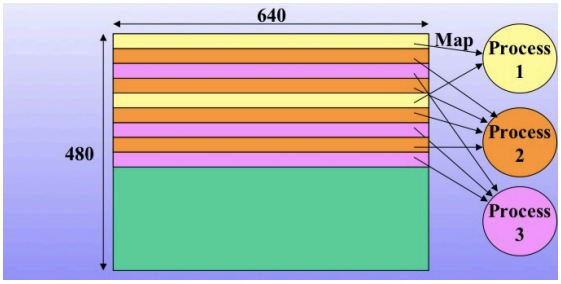

Figure 2. Row Segmentation-based Partition Scheme

After being done with segmenting all the rows. The processors (slaves) will then send their generated sub-rows back to the Root node (Master) and after receiving all the sub-rows from all the processors, the Master/Root Node will then finally write onto the .ppm file the image of the Mandelbrot Set that obeys the Algorithm. The Flow Chart below may help understand this process in a clearer manner.

IV. RESULTS AND DISCUSSIONS

The time taken for the parallelized Mandelbrot Set Algorithm to execute successfully was taken down and recorded for all 2, 4, 6 and 8 logical processors. All of this data is illustrated in Appendix B in the Appendices. From there, we used the values to calculate the experimental speed up factor and that data is clearly shown in Table 2. This experimental speed up factor is calculated using equation (10) below as well.

Number of processors, p	2	4	6	8
Speed Up Factor, S(p)	1.04	2.05	3.03	3.93

TABLE 2

EXPERIMENTAL SPEED UP FACTOR USING AMDAHL'S LAW

$$S(p) = \frac{Ts}{Tp} \quad (10)$$

Ts = Execution time using one processor

Tp = Execution time using multiple processors

All the time values that were taken down was recorded in the table that is so eloquently made and stored at Appendix B under Appendices. I have also taken the liberty to plot a Graph of Experimental Speed Up VS Theoretical Speed Up and the Graph is illustrated in Appendix C under the Appendices section as well. From Appendix C, it can be observed that the experimental speed up factor for the Round Robin Row Segmentation-Based Partitioning Scheme was analysed although it performed well against other partitioning schemes, it was still far below the corresponding theoretical speed up factor. This comparison can also be done by observing Table 1 and Table 2.

The significant difference in both the Theoretical Speed Up and the Experimental Speed Up can be accounted for as a consequence of the latency that is introduced by MPI_Send and MPI_Recv which takes a huge amount of time to send and receive tasks between the available slave processors and also to the Master for writing. Regardless of that reason, the speed up for the partitioning scheme I have introduced still outperformed all other naïve methods such as the grid segmentation-based scheme. Also, it was proven that when the process is carried out with a greater number of processors, the jump in improvement was significant. However, this improvement was non-linear due to the reason that

the increase in number of processors means an increase in overhead timing for receiving and sending too.

V. CONCLUSION AND FUTURE WORK

The proposal submitted by this report is to adopt and implement the Round Robin Row Segmentation-Based Partitioning Scheme to be able to parallelize the Mandelbrot Set Algorithms for the purposes of Scalability. The parallelizing Based on the partition scheme is done by utilizing the Message Passing Interface (MPI) library in C Programming. The ramification of the implementation exhibits with evidence, significant acceleration of the program for the proposed Round Robin Row Segmentation-Based Partitioning Scheme using MPI on C Programming as compared to the serial implementation. Hence, the results indicate the triumphant parallelization of the program.

To conclude, it was an amazing learning experience working with fractals and taking on the unbelievable task of parallelizing a Mandelbrot Set and I can gladly say that the beauty of the Mandelbrot set is unparalleled in its vast range of possibilities. That being said, future works in relation to Mandelbrot Sets would include introducing many other computer science related problems with the guide of Mandelbrot sets.

REFERENCES

[1] MacGregor Campbell, 2013, "5.6 Scaling and the Hausdorff Dimension," at Annenberg Learner:MATHematics iluminated, 5 March 2015.

[2] Adrien Douady and John H. Hubbard, Etude dynamique des polynômes complexes, Prépublications mathémathiques d'Orsay 2/4 (1984 / 1985)

[3] Boeing, G. (2016). "Visual Analysis of Nonlinear Dynamical Systems: Chaos, Fractals, Self-Similarity and the Limits of Prediction". Retrieved December 2, 2016.

[4] S.V. Adve et al. (November 2008). "Parallel Computing Research at Illinois: The UPCRC Agenda" Archived 2018-01-11 at the Wayback Machine (PDF).

[5] The MPI Forum, CORPORATE (November 15–19, 1993). "MPI: A Message Passing Interface". Proceedings of the 1993 ACM/IEEE conference on Supercomputing. Supercomputing '93. Portland, Oregon, USA: ACM. pp. 878–883.

[6] Amdahl, Gene M. (1967). "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities" (PDF). AFIPS Conference Proceedings.

APPENDICES

Appendix A (Serial Implementation)

Computer Specifications	a) CPU Model: Intel Xeon - 2nd Gen b) Number of Logical Processors: 16 c) Memory Size: 64GB RAM d)Network Speed: 1Gbps	
Value of iXmax	8000	
Value of iYmax	8000	
Value of IterationMax	2000	
Test number	Serial Program	
	Overall Time	Computational Time
1	36.200922	36.200919
2	36.393744	36.393740
3	36.095323	36.095320
4	36.099198	36.099194
5	36.132588	36.132585
Average	36.184355	36.184351

Appendix B (Round Robin Row Segmentation-Based Implementation)

Computer Specifications	a) CPU Model: Intel Xeon - 2nd Gen b) Number of Logical Processors: 16 c) Memory Size: 64GB RAM d)Network Speed: 1Gbps			
Value of iXmax	8000			
Value of iYmax	8000			
Value of IterationMax	2000			
Test number	Parallel program			
	2 Logical Processors	4 Logical Processors	6 Logical Processors	8 Logical Processors
1	34.803088	17.601793	11.824574	9.323454
2	34.641767	17.590415	12.108949	9.333527
3	34.644154	17.595952	11.804619	9.110733
4	34.776158	17.594686	12.235621	9.043595
5	34.550659	17.960973	11.785955	9.250827
Average	34.683165	17.668764	11.951944	9.212427

Appendix C (Graph of Experimental Speed Up VS Theoretical Speed Up)