Mandelbrot Generation Partitioning Scheme using Message Passing Interface

Priscilla Ai Ching Tha
Monash University
Sunway, Malaysia
ptha0007@student.monash.edu

***Abstract*—The Mandelbrot Set is a two-dimensional representation of an iterated function $f(x) = x^2 + c$ on the complex plane, where c is a constant complex number. This paper discussed the parallelizability of the expensive computation of the Mandelbrot Set and compares three partitioning schemes namely the row segmentation-based, tile segmentation based and the round Robbin implementation for the most efficient scheme. Comparisons utilizes Amdahl's law on speed up factors. These partitioning schemes are realized using Message Passing Interface (MPI) library in C language.**

Keywords—Mandelbrot Set, MPI, distributed computing, parallel processing.

I. INTRODUCTION

The Mandelbrot Set was first given a definition in 1978 by Robert W. Brooks and Peter Matelski. It has been widely explored for the last few decades which contributes to its implementation in graphics and design, complex dynamics and cryptography. Concurrently, technology has advanced in the application of parallel computing theory to improve on performance and speed. Parallelising the computation of the Mandelbrot Set allows technological advantage to cover a wider scope due to its general nature. Following the fatal attack of WannaCry and on Equifax , the ideal encryption scheme that is based on the Mandelbrot Set could be achieved faster with parallelisation during research.

Mathematically, the polynomial function seems very simple.

$$f(x) = x^2 + c$$

forming the sequence:

$$(0, f(0), f(f(0), f(f(f(0))), \dots)$$

The set default size of the complex plane is 8000×8000. A default iteration = 2000 is decided upon to generate an accurate black and white based Mandelbrot Set on the plane. Then, the iteration formula is:

$$\text{No. of elements} = 8000 \times 8000 \times 2000$$

The big values in the equation above makes the process computationally intensive for a single central processing unit (CPU) to handle. Parallelisation exploits the development in technology to benefit from the multiple cores built into the machine and smaller transistors in the CPU nowadays. A few partitioning schemes is discussed to fulfil this demand.

Firstly, these partitioning schemes are realized using Message Passing Interface (MPI) library in C language [2]. The most common partitioning schemes are row segmentation and tile segmentation-based partitioning. Those aside, another – round Robbin – partition scheme is suggested, and their performance is evaluated and compared as reference in this report. Analysis is drawn with influence from the Amdahl's law analysis by calculating their speed up factor based on the execution time duration of the parallelized Mandelbrot Set generation process. Further results are also attached at the end of the report for personal interpretation and judgement.

II. PRELIMINARY ANALYSIS

A. Parallelizability of the Mandelbrot Set Generation

Is the Mandelbrot Set parallelizable? The values must be reviewed to ensure compliance to Bernstein's conditions. Bernstein's measures the capability of executing two processors to perform simultaneously [4]. The conditions are:

$$I_0 \cap O_1 = \emptyset$$

$$I_1 \cap O_0 = \emptyset$$

$$O_0 \cap O_1 = \emptyset$$

with I_0 and O_0 represent the input and output for processor P_0. I_1 and O_1 on the other hand represent the input and output for processor P_1. The processors are mutually exclusive of each other if the three conditions are satisfied thus, parallelizable.

B. Theoretical Speed Up of the Mandelbrot Set Generation

A serial based Mandelbrot Set generation is the basis of the comparison on speed up factor. The algorithm includes: (1) Open a binary file ready for writing, (2) Perform Mandelbrot Set computation and (3) Write the computed colours into the binary file steps. Table A in the appendices records the time taken to complete the algorithm serially.

Subsequently, the following equation is the Amdahl's law:

$$S(p) = \frac{1}{r_s + \frac{r_p}{p}} \quad \textbf{(4)}$$

with

r_s = serial ratio (non-parallelizable portion)

r_p = parallel ratio (parallelizable portion)

p = number of processors

used to calculate the theoretical speed up factor when computation task is divided to multiple processors. Table I shows the calculated theoretical speed up factors S(p) for the number of processors p = 8 and p = 24.

TABLE I. THEORETICAL SPEED UP FACTOR USING AMDAHL'S LAW

Number of processors, p	8	16
Speed up factor, S(p)	7.760295332	15.0067081

The r_s is the file writing portion whereas r_p is the portion from opening the file (1) until the end of computation (2) because the computation is parallelized among the multiple processors whereas the writing is only done by one processor after the rest send data over.

III. DESIGN OF PARTITION SCHEMES

All schemes started with declaring variables and opening a binary file for writing. Subsequently, the values computed from the next step is stored in a one-dimensional dynamic array. It is useful to note here that the usage of one-dimensional array reduces space complexity by offering contiguous memory storing and smaller excess from allocation and deallocation, and time complexity by faster array access. The values are the three colours code in RGB so the next pixel on plane is the values after the first three in the array. At the end of computation, the program writes the content of the array to a binary file starting from the root to the other processors. The root performs the process after receiving data from the others in an orderly manner.

A. Row Segmentation-based Partition Scheme

In this scheme, the utilisation of CPUs is done by dividing each row of pixels to the number of processors. Each processor calculates their respective starting and ending point of a row segment. Such (Figure I) shows the partition between processors using the following formula.

$$Rows\ per\ processor = \frac{Number\ of\ rows\ in\ plane}{Number\ of\ processors} \quad (5)$$

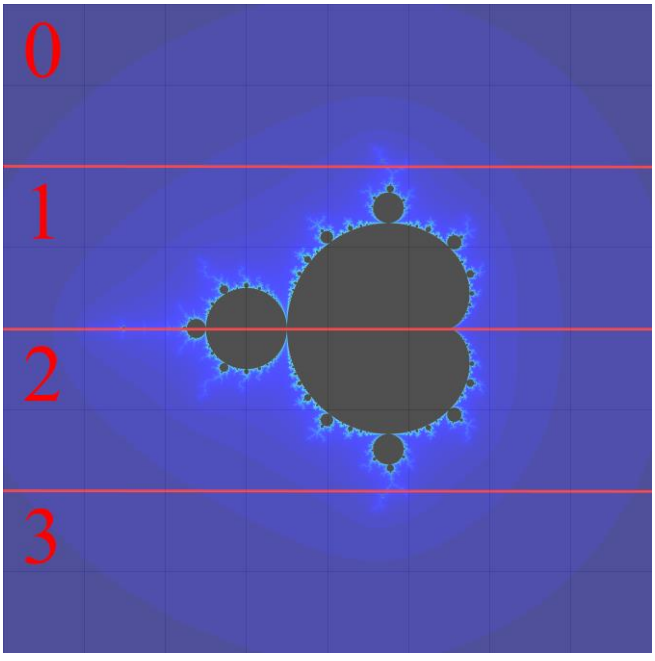

Figure I Row segmentation-based partition when p = 4.

Now, all the processors compute their respective row segment. A call to MPI_Send sends each of their array – except the root node – to the root for writing once complete. The root node first writes its own array before using MPI_Recv to receive the data and allocate its array pointer to the received array before proceeding to write the new contents. This communication and writing process will be repeated size - 1 time with size = number of processors. Figure II illustrates the program for row segmentation-based partition scheme.

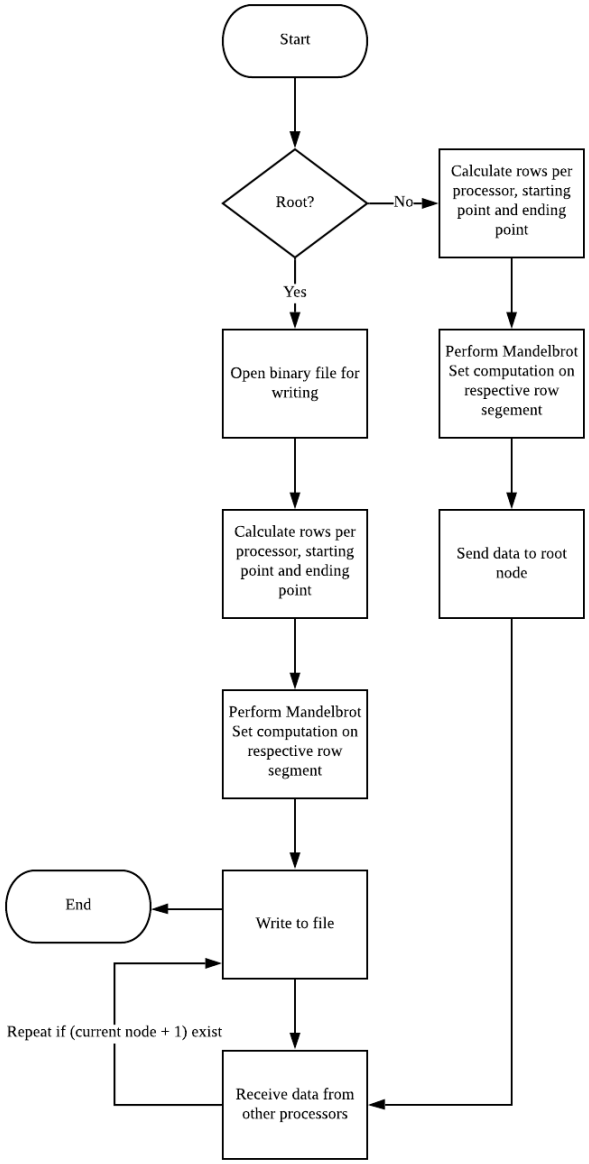

Figure II Flowchart of row segmentation-based partition scheme.

However, there may be the case where the rows cannot be divided equally among the processors. The last few rows are then dealt by the last processor in rank to compute as it was dealing with the preceding rows, after the computation of its supposed row segment. Thus, writing can take place in the correct order without the need to determine order using message tag and sender identification.

B. Tile Segmentation-based Partition Scheme

In this scheme, the utilisation of CPUs is done by dividing each tile of pixels to the number of processors. Each processor calculates their respective starting and ending point of rows and columns. Such (Figure III) shows the partition between processors using the following formula.

$$Rows\ per\ processor = \frac{Number\ of\ rows\ in\ plane}{Number\ of\ processors/_2} \quad (6)$$

$$Columns\ per\ processor = \frac{Number\ of\ columns\ in\ plane}{2} \quad (7)$$

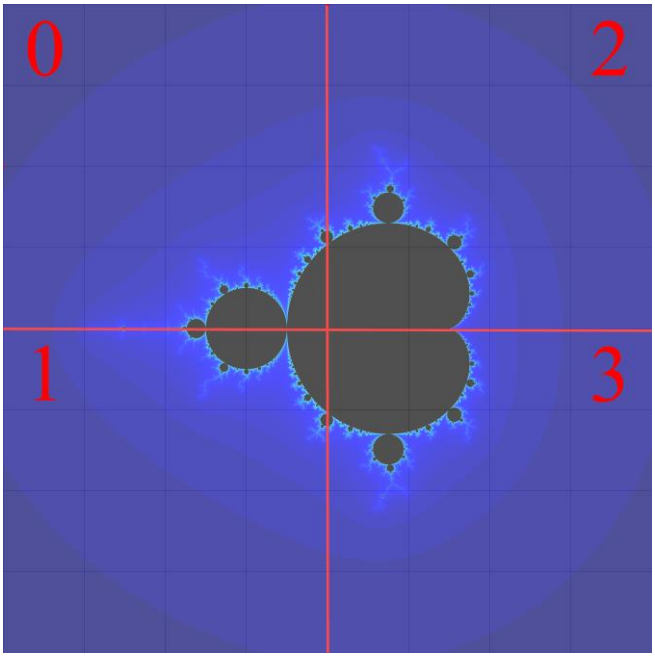

Figure III Tile segmentation-based partition when p = 4.

Now, all the processors compute their respective tile segment. A call to MPI_Send sends each of their array – except the root node – to the root for writing once complete. Tile segmentation-based writing is different from row segmentation-based. Since moving the file writing pointer to another position would cost a great image distortion, tiles next to each other must be written concurrently.

The root node first uses MPI_Recv to receive data from a processor doing the tile on the right (as the program always starts from the left and divides the columns in the plane into two) and allocate the newly created temporary array pointer to the received array. It then proceeds to write row by row of its computed values and the received computed values next to each other before moving on to the next row. This communication and writing process will be repeated size - 2 time with size = number of processors. Figure IV illustrates the program for tile segmentation-based partition scheme.

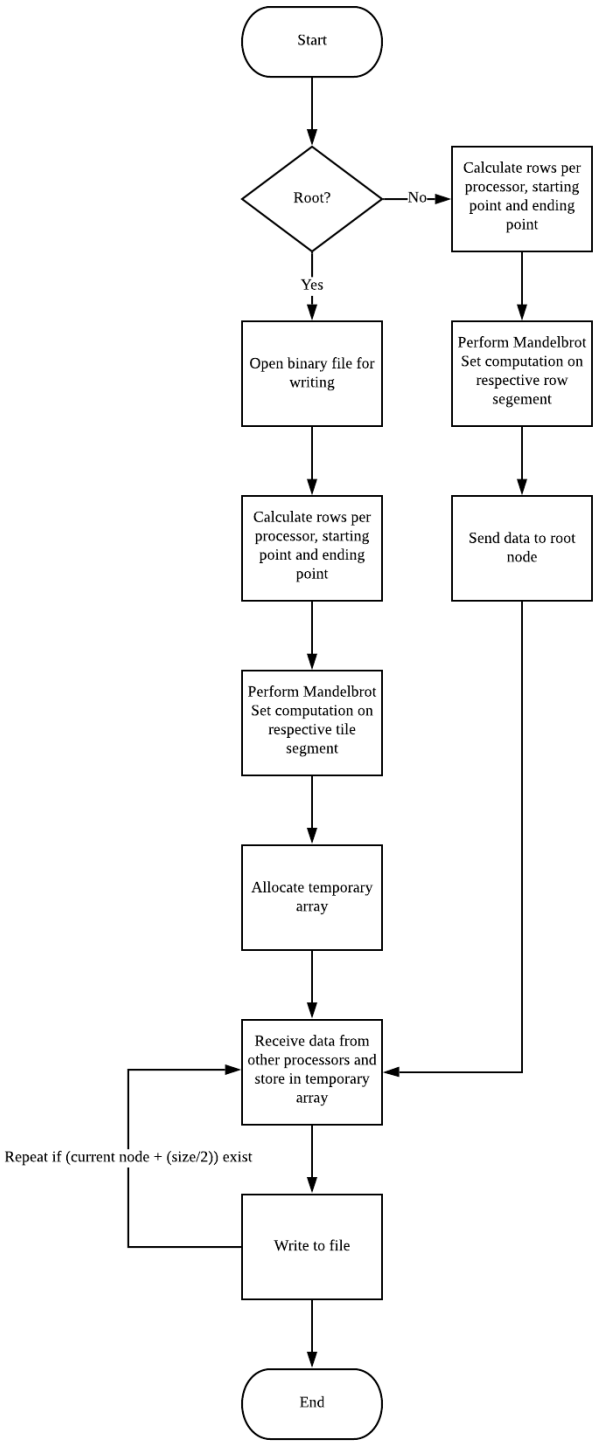

Figure IV Flowchart of tile segmentation-based partition scheme.

However, there may be the case where the tiles cannot be divided equally among the processors. The last few rows and columns are then dealt by the last processor in rank to compute as it was dealing with the preceding rows, after its supposed tile segment. Thus, writing can take place in the correct order without the need to determine order using message tag and sender identification. The report has decided to neglect this situation instead, due to the increase

in measurements from the (unparalleled) extra computation deeming it irrelevant for the discussion following.

C. Round Robbin Partition Scheme

In this scheme, the utilisation of CPUs is done by applying arithmetic progression to processor on row assignment. Each processor starts from different rows in a serial manner and moves on to the successive nth row of the sequence such that the difference between the consecutive rows is size with size = number of processors. Such (Figure V) shows the partition between processors using (4) and the following formula to assign rows.

$$nth\ row = starting\ row + (n-1)(size) \quad (8)$$

with

size = number of processors

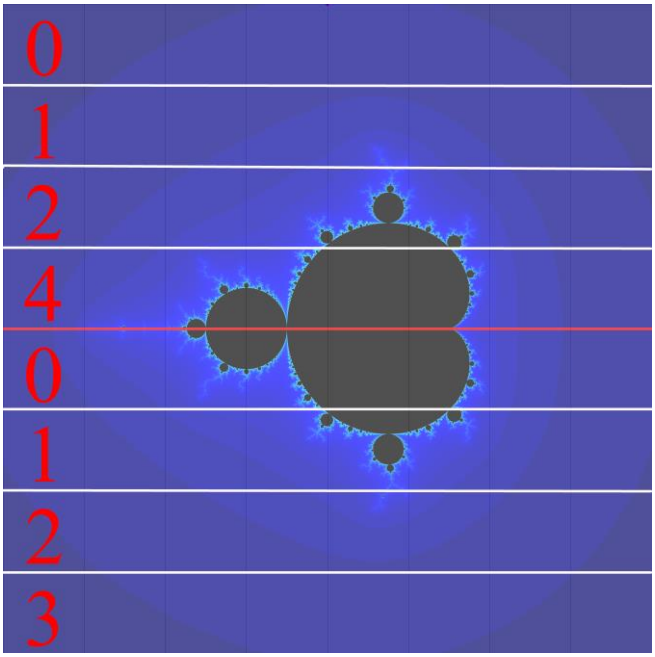

Figure V Round Robbin partition when p = 4. The height/width of the pixels are greatly exaggerated for readability.

Now, all the processors compute their respective rows. A call to MPI_Send sends each of their array – except the root node – to the root for writing once complete. Round Robbin based writing is different from row and tile segmentation-based. Since moving the file writing pointer to another position would cost a great image distortion, rows next to each other must be written concurrently. .

The root node first allocate memory for a temporary array and uses MPI_Recv to receive data from all the processors storing the received arrays values into the temporary array. It then proceeds to write row by row so if the nth row consists of 8000 pixels then the next row is in the position (8000×3)+rows per processor of the temporary array. It serves to remind that the next row is computed by the next processor in rank which array is appended to the temporary array after the array from the preceding processor in rank. Figure VI illustrates the program for round Robbin partition scheme.

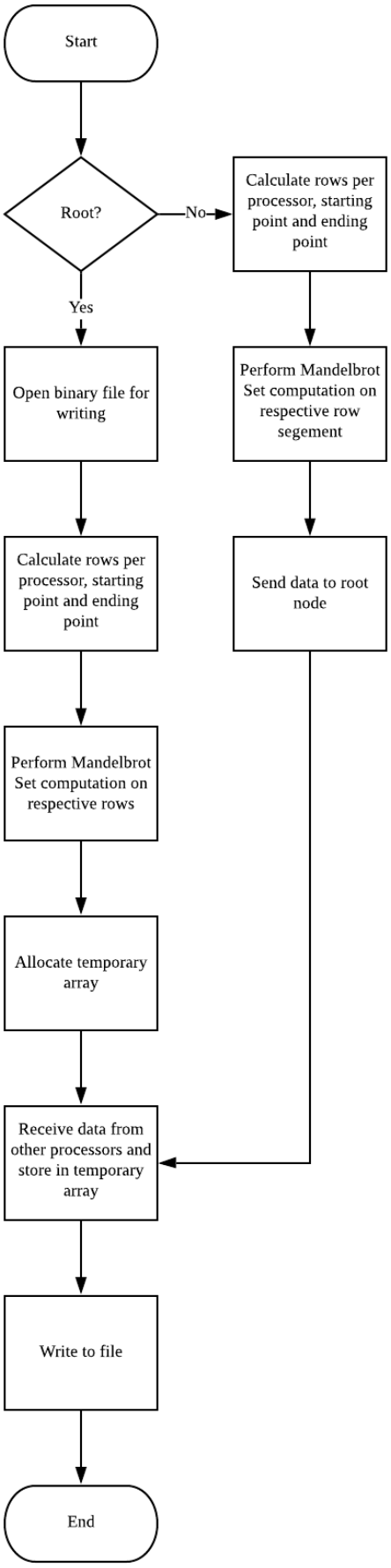

Figure VI Flowchart of round Robbin partition scheme.

However, there may be the case where the rows cannot be divided equally among the processors. The last few rows are then dealt by the last processor in rank to compute as it was dealing with the preceding row. Thus, writing can take place in the correct order without the need to determine order using message tag and sender identification.

IV. RESULTS AND DISCUSSION

For all the partition schemes described in Section III, that time taken for the generation of the Mandelbrot Set using up to 16 logical processors is recorded in Table A in the appendices. Subsequently, the following equation is used to calculate the experimental speed up factors of the generation of the Mandelbrot Set using all the partitioning schemes.

$$S(p) = \frac{t_s}{t_p} \quad \textbf{(9)}$$

with

t_s = execution time using one processor

t_p = execution time using multiple processor

Table II shows the calculated experimental speed up factors S(p) for the number of processors p = 8 and p = 24.

TABLE II. EXPERIMENTAL SPEED UP FACTOR

Partition Schemes	Speed up factor, S(p)	
	Number of processors, p	
	8	16
Row Segmentation	2.449882474	3.716399885
Tile Segmentation	2.795870093	3.375442248
Round Robbin	7.299741024	10.9589029

Consequently, the values in Table II do not match with the values in Table I using both p = 8 and p = 16 with p = number of processors for all partition schemes. The theoretical speed up factor shown in Table I are S(8) = … and S(16) = … whereas the corresponding values in Table II for partition schemes are far lower. The difference in values between the theoretical and experimental speed up factor may be due to the latent overhead from the communication process using MPI_Send and MPI_Recv between multiple processors. This happened during sending data to the root node and receiving data from other processors for writing the file. Nevertheless, the time taken to complete each partition schemes from p = 2 to p = 8 to p = 16 demonstrated significant decrease although the sudden rise from unequal division of rows and/or columns when p is not a factor of 8000 (row segmentation-based). Round Robbin however, demonstrated continuous decrease despite the mentioned condition.

Furthermore, the fundamental theorem of arithmetic states that each integer > 1 can be expressed in exactly one way, up to order, as a product of primes. It shows that 8000 can be expressed as

$$8000 = a^b \quad \textbf{(10)}$$

with

a = prime integer

b = prime integer exponent

which indicates these primes as the factors of 8000. It also states that if an integer n > 1 has a divisor, it has a divisor ≤ $\sqrt{n}$, because for any divisor c > $\sqrt{n}$ we also have the divisor n/c. Thus, the primes from 1 to 8000 can be found up to the domain $\sqrt{8000}$ which gives the total number of prime factors be $\sqrt{8000} \leq 90$. Adding to that the total of evens that are not prime = (5×8000÷10) - 1 giving the total number of factors is 3999.

This justifies the significance of round Robbin faster timings which otherwise, using the other two partition schemes, could only use less than half of the plane size possible number of processors to compute the generation of the Mandelbrot Set faster. It will be more obvious when the plane size is bigger where the program execution might need to be forfeited when it takes too long. That said, the consideration of round Robbin towards the situation of using number of processors which are not a factor of 8000 draws the experiment closer towards the motivated value (the theoretical speed up factor) as seen from the consistent increase of the experimental speed up factor as opposed to row segmentation-based (Figure VII).

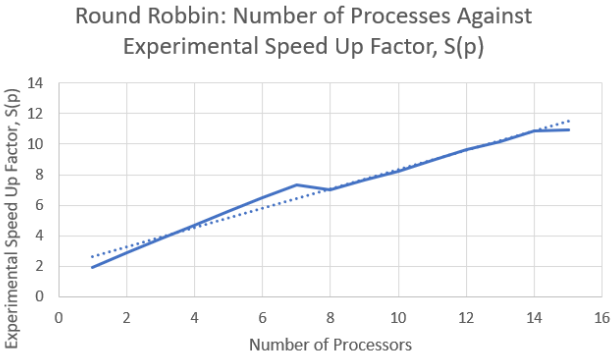

Figure VII Round Robbin: Number of processors against experimental speed up factor, S(p).

On another hand, the characteristic of the Mandelbrot Set requires heavier computation for the part in black as it is generated by the values of c which cross the threshold after the maximum iteration. Row segmentation-based partition scheme divides the rows among processors equally but does not divides the workload among processors equally (Figure I). It is evident that the top and bottom row will finish computing faster than the middle two rows when p = 4. On top of dealing with unequal distribution of rows, the computation of c also reduces the speed up factor which

produce a wavy pattern rather than a near linear graph (Figure VIII).

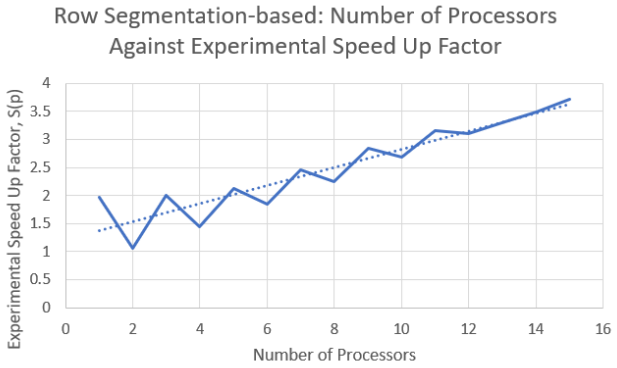

Figure VIII Row segmentation-based: Number of processors against experimental speed up factor.

Tile segmentation-based may ensure fairness when p is small (Figure III) but as the number of processors increases, the tile surface area covered by each processor also decreases. If the last processor computes a larger part of the blacks in the Mandelbrot Set and then assigned to compute the remainders should there is any, the workload of the last processor will increase which then contributes to extra time.

Although the round Robbin implementation is very similar to row segmentation-based, it ensures fairness for any number of processors because the responsibility of each processor throughout is small. Row segmentation-based divides rows such that each processor handles a horizontal chunk of pixels whereas round Robbin divides rows such the horizontal chunk is broken down into one horizontal. Hence, the probability of the blacks to fall into the horizontal chunk of one processor is higher compared to the blacks which falls on multiple rows but handled by multiple processors (Figure IX). All the processors will thus take similar time to finish the computation.

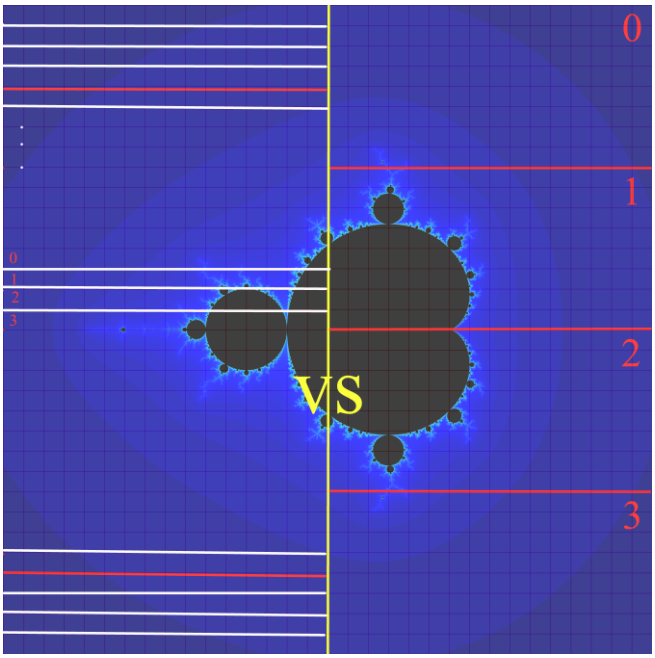

Figure IX Round Robbin vs row segmentation-based workload division.

CONCLUSION

Therefore, the Mandelbrot Set clearly takes lesser time to compute as seen on Table A in the appendices using parallelism. The best partition design scheme from this experiment is round Robbin which faster computational time leads to higher experimental speed up factor. The difference between theoretical and experimental speed up factor using round Robbin is also very small ranging from 3 to 5 seconds only. Aside from justifying the ability to enhance computational performance using parallelism, the experiment also manages to find one of the most optimal and generalized, if not the only one, partition design scheme. The increase in performance using parallelism with round Robbin partitioning scheme will be the pioneer to our development in research in fields based on the Mandelbrot Set.

REFERENCES

[1] Drakopoulos, V., Mimikou, N., & Theoharis, T, An overview of parallel visualisation methods for Mandelbrot and Julia sets, *Computers & Graphics*. pp. 635-646, 2003

[2] Duan, X., Shen, W., & Guo, J., The MPI and OpenMP Implementation of Parallel Algorithm for Generating Mandelbrot Set, *Applied Mechanics and Materials*. Vol 571-572, 2014

[3] Gao, W., Jie, S., Wei, Q., & Zhang, X., Digital image encryption scheme based on generalized Mandelbrot-Julia set, *Optik*. pp. 917-929, 2019.

[4] Feautrier Paul, "Bernstein's Conditions", *Encyclopedia of Parallel Computing*. Springer US, pp. 130-134, 201