Design and Implementation of The Mandelbrot Set Algorithm Using Message Passing Interface

Linh Hoang Le
Bachelor of Computer Science
Monash University
Melbourne, Australia
llee0010@student.monash.edu

***Abstract*—The Mandelbrot set is a set of complex number c for which the function $f_c(z) = z^2 + c$ does not diverge from the bounding domain through many iterations of the same calculations on itself using the result from its last calculation and feed them into the next iteration of the same calculation starting from z = 0. This document presents a way to parallelize the computation of the Mandelbrot set using the Message Passing Interface (MPI) library in C language. Bernstein condition is introduced in this document to verify the parallelizability of the Mandelbrot set algorithm. The Amdahl's law will be used to compute and analyze the performance of the parallelized program. The partition scheme that is used to parallelize the computation of the Mandelbrot set is the row by row segmentation scheme and is tested on a single multicores' computer. The result of the segmentation scheme shows a linear time reduction in the performance of the Mandelbrot set computation using MPI compared to the serial implementation scheme. This indicates the successful parallelization of the calculation of the Mandelbrot set.** *(Abstract)*

Keywords-Mandelbrot Set; Message Passing Interface (MPI); distributed computing; parallel processing.

I. INTRODUCTION *(HEADING 1)*

The Mandelbrot set is a mathematical curiosity that creates beautiful images. There is no non-artistic real-life application of the Mandelbrot set however, fractals in general have many uses such as shoreline analysis, data compression, antenna design, etc...

Fractals are a never-ending pattern and the pattern are infinitely complex generated by repeating a simple calculation over and over in an ongoing feedback loop typically driven by recursion much like the Mandelbrot set where it is created by a computer calculation a simple equation in a feedback loop. The computation process of the Mandelbrot set can be significantly long when done serially if the data is given is big enough, so there is a high demand to have an algorithm that can speed up the calculation process.

The Mandelbrot set images is generated by inspecting each point c on the complex plain whether the sequence f(0), f(f(0)), f(f(f(0))), … to infinity, diverges or converges. In the real-world, the computation of each point would not go to infinity but instead iterates through a predetermined number of times (typically over a large number of iterations). If the point on the plain converges and stay within a bounded domain and the number through the iterations does not get larger, then the point is within the Mandelbrot set and is color coded black. Inversely, if the iterations at a point c causes the number to diverge, the color at that particular point will be colored blue in our case of generating the Mandelbrot set image. A picture generated by the Mandelbrot set is shown below in figure 1.

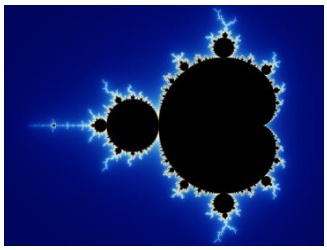


Fig. 1 Image generated through the computation of the Mandelbrot set

Figure 2 shows the computational process of the Mandelbrot set.

$$f_c(z) = z^2 + c$$

$$z = 0 \rightarrow f_c(0) = z^2 + c$$

$$f_c(f_c(0))$$

$$f_c(f_c(f_c(0)))$$

$$\vdots$$

$$Number\ of\ iterations$$

The parallelization of the Mandelbrot set is computed using the Message Passing Interface (MPI) library in C language. The partitioning scheme for the parallelization of the Mandelbrot set is the row by row interweaving segmentation which divide each row of the computation to

each processor. The performance of the parallelized Mandelbrot set will then be evaluated based on the execution time of the program and will be used to compare against the theoretical speed up time obtained by the analysis using Amdahl's law.

II. THEORETICAL SPEED UP ANALYSIS FOR MANDELBROT SET

A. *Parallelizability of the Mandelbrot set*

Bernstein's conditions are used to determine whether the Mandelbrot set problem can be parallelized, that is whether two process can be calculated independently. The three Bernstein's conditions are:

$$I_0 \cap O_1 = \emptyset \text{ (anti dependency)}$$

$$I_1 \cap O_0 = \emptyset \text{ (flow dependency)}$$

$$O_0 \cap O_1 = \emptyset \text{ (output dependency)}$$

I(0) and O(0) is the input and output of the first processor and I(1) and O(1) is the input and output of the second processor. If the three conditions are satisfied, the two processors can perform computations independently and hence able to compute the problem in parallel.

The Mandelbrot set calculates its sequence iteratively on each coordinate within the 8000x8000 canvas. So the input at the point C(0) will not give the output of the point C(1) and vice versa, this ensures the anti-dependency and flow-dependency conditions. And the output at C(0) will also be independent from the output at point C(1), so the output-dependency condition is also satisfied. Thus the Mandelbrot Set can be computed in parallel.

B. *Maintaining the Integrity of the Specifications*

The theoretical speed up of the Mandelbrot set can be computed using Amdahl's law:

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

Where

$$r_p = parallel\ ratio$$
$$r_s = serial\ ratio$$
$$p = number\ of\ processors$$

The serial-based computation of the Mandelbrot set problem is given and it can be used to calculate the theoretical speed up time of the parallel program of the Mandelbrot Set. The serial program of the Mandelbrot Set can be divided into two portions: (1) declaring variables, create a file to write to and write a header to the file, close the file and printing outputs, (2) perform the Mandelbrot set calculation and writing the data into the file.

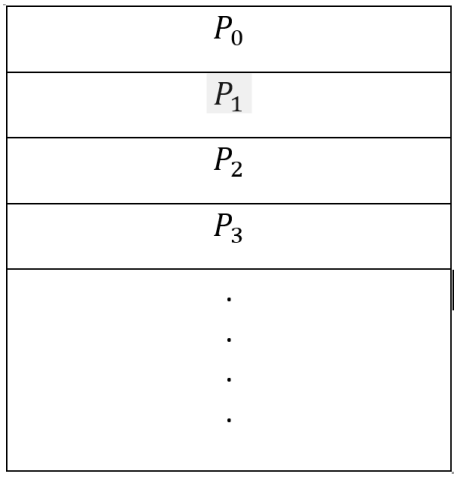
The table below shows the theoretical speed up factor S(p) for p = 2 (2 logical processors) and p = 4 (4 logical processors).

Number of processors <p>	2	4
Speed up factor <S(p)>	1.998	3.994

The r(s) is the declarations of variables and the r(p) is the computation of the Mandelbrot set which is parallelized using multiple processors.

III. DESIGN OF PARTITION SCHEMES

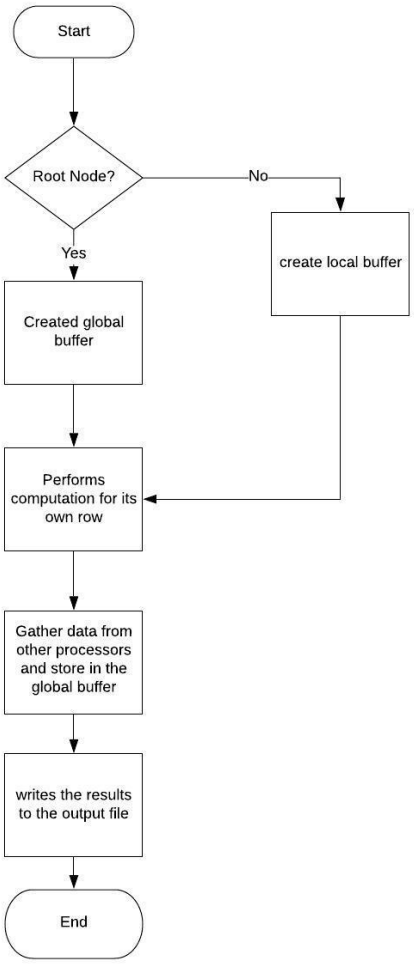
The row by row partitioning scheme splits up the computation along the Y axis of the canvas. It iterates through the max number of rows in the direction of the Y axis and increment by the number of processors we assigned to run the program in parallel. The result after calculating a row by each processor is then stored in a 1D array that is local to each processor, then the root node (i.e. rank 0) will collect the information that has just been calculated by the other processors, store it in a global buffer and immediately write the information stored in the global buffer to the output file. Below is the visual representation of the row by row



P0 = processor 0
P1 = processor 1
P2 = processor 2
P3 = processor 3

segmentation scheme.

The figure below shows the flow of the row by row partitioning scheme.



IV. Results and Discussion of the Implemented Partitioning Scheme

The algorithm calculates the Mandelbrot set by iterates over a 8000x8000 plane and calculate for each pixel whether or not it is in the Mandelbrot set and assign the color to it. If it is inside the Mandelbrot set then the color will be black, if it is outside the set then the color will be blue. The implementation of the parallelized Mandelbrot program is in C language using Visual Studio Code.

The parallelized Mandelbrot program is tested by running on a single multicore computer with 2 and 4 logical processors.

Partitioning scheme	**Table Column Head**	
	P = 2	**P = 4**
Row by Row segmentation scheme	1.972	3.628

The results in the table above is calculate using the

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

Where

$$t_s = execution\ time\ using\ one\ processor$$

$$t_p = execution\ time\ using\ multiple\ processors$$

equation.

From the result we can see that the experimental speed up time is very close to the theoretical speed up time. However, the experimental speed up time for the parallelized program for 4 processors is slightly lower than the theoretical speed up time. This can be due to the number of times the program takes to write out into the output file. The number of times that the parallelized program takes to writes out to the file is the same as the number of iterations of the program through the Y axis (i.e. 8000 times). However, the difference between the theoretical speed up time and the experimental speed up time does not deviate much from each other. We can conclude that this partitioning scheme produce an improving in the execution time of the program linearly.

V. Conclusion

The report explains how the Mandelbrot set can be segmented and perform in parallel using Message Passing Interface library. Row by row segmentation scheme is introduced as a way to compute the set in parallel. The partitioning scheme gives a linear result that is close to the theorical result using Amdahl's Law. The computation time for the Mandelbrot set has been significantly increase in terms of time using 2 and 4 logical processors and the theoretical speed up time when compared to the experimental speed up time is close to each other. In conclusion, the row by row segmentation scheme will give a linear result when used to compute the Mandelbrot set algorithm in parallel.

References

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

[2] En.wikipedia.org. (2019). *Mandelbrot set*. [online] Available at: https://en.wikipedia.org/wiki/Mandelbrot_set [Accessed 5 Sep. 2019]

[3] Fractalfoundation.org. (2019). *What are Fractals? – Fractal Foundation*. [online] Available at: https://fractalfoundation.org/resources/what-are-fractals/ [Accessed 6 Sep. 2019].

[4] Thomas, D. (2019). [online] Quora.com. Available at: https://www.quora.com/What-are-the-real-life-applications-of-the-Mandelbrot-set [Accessed 6 Sep. 2019].

Table A – Computer Specification

Computer Specification	Intel® Core™ i7-4702HQ CPU @ 2.20GHz 2.20GHz Specify number of logical processor ○ 8 logical processors ○ Ubuntu Linux VM Specify memory size ○ 8GB Specify network speed (e.g., 1 Gb/s) ○ Download: 99.20 Mbps ○ Upload: 189.32 Mbps
Value of iXmax	8,000 (default)
Value of iYmax	8,000 (default)
Value of IterationMax	2,000 (default)

Table B – Serial vs Parallel Mandelbrot Program

	Serial program	**Parallel Program**	
		MPI	
		2 logical processor	4 logical processors
Run 1	60.812500	29.062752	15.986618
Run 2	61.718750	28.836157	15.612306
Run 3	60.953125	28.934454	15.728326
Run 4	60.828125	29.871943	15.556477
Run 5	60.203125	28.814839	15.948398
Average Time	60.903125	29.104029	15.766425