Design and Implementation of Mandelbrot set plot algorithms using Message Passing Interface

Initial report prepared by:

Cheng Zeng

***Abstract*—This report explains how to design and implement two partition schemes to realize the parallelized Mandelbrot set plot algorithm using the Message Passing Interface library. Bernstein Condition is employed to check whether the Mandelbrot set plot algorithm can be parallelized, and Amdahl's is used to analyze speed up factor. It is concluded that the speedup of Mandelbrot set plot program can be achieved by employing two partition schemes although there is difference between the actual and the theoretical speed up factors.**

Keywords—Mandelbrot set, MPI, distributed computing, parallel processing. Introduction

I. INTRODUCTION

The Mandelbrot set is a famous fractal or a type of iteration, in which a process can be repeated over and over again. The set is mathematically fascinating because of its mysteries. In mathematics, it can also be explained as a mathematical function which can be written in the form that $f(x)=x^2+c$. In this function, c can either be constant or not, which can decide whether the orbit of 0 go to infinity or not (Devaney, 2019). Although the Mandelbrot set hasn't had any practical applications so far, it can create some beautiful images.

Parallel algorithm is an algorithm which can perform multiple operations simultaneously on different nodes (i.e. processors, devices) and combine all the individual results to produce the final output (Parallel Algorithm-Introduction, 2019). When using computer to draw Mandelbrot set, it involves a large amount of computation. A parallel algorithm can distribute the computation of Mandelbrot set to multiple processors in order to speed up the process. The parallel partition scheme plays an important role in parallel algorithm, it not only determines how to distribute the workload, but also impact the final performance of the parallel program. Points in the Mandelbrot set causes more computation, so these points are distributed more equally, the performance of the parallel Mandelbrot set plot program should be better.

In this report, the parallelization of plotting Mandelbrot set is achieved using the Message Passing Interface (MPI) library in C language. Two partition schemes are considered to distribute computation of Mandelbrot set to different number of processors which is responsible for sub-computation of Mandelbrot set that are assigned to them. These two partition schemes are row segmentation-based and pixel-based. The performance of implementation of these two partition schemes are assessed through recording the execution time of the parallelized Mandelbrot set program. The row segmentation-based partition scheme delivers a better performance, and its actual speed up factor is calculated to compare with the theoretical speed up factor gained from the Amdahl's law analysis.

II. PRELIMINARY ANALYSIS

A. Parallelizability of Mandelbrot set computation

Bernstein's conditions are usually applied to detect parallelism in a serial algorithm, it can help us to find operations that can be conducted independently (BravoYusuf, 2019). Bernstein's conditions set out three conditions that are able to decide whether two operations can be performed without influencing each other. Two operations O_i and O_j are independent if the following three conditions are met:

$I(O_j) \cap O(O_i) = \varnothing$ flow independence
$O(O_i) \cap O(O_j) = \varnothing$ output independence
$I(O_i) \cap O(O_j) = \varnothing$ anti independence

That is, two operations can be performed in parallel if (1) the input of second operation and the output of first operation are disjoint (2) the output of first operation and the output of second operation are disjoint (3) the input of first operation and the output of second operation are disjoint.

Equation (1) and (2) denote the operation of computing Mandelbrot set of two adjacent points $P0(x_0, y_0)$, $P1(x_1, y_1)$.

$$color_0 = compute\_mandelbrot\ (x_0, y_0) \quad (1)$$

$$color_1 = compute\_mandelbrot\ (x_1, y_1) \quad (2)$$

Whereby

$I(O_i) = (x_0, y_0)$ $O(O_i) = color_0$
$I(O_j) = (x_1, y_1)$ $O(O_j) = color_1$

The independence of these two operations is assessed by applying Bernstein's conditions

$$(x_1, y_1) \cap color_0 = \varnothing$$
$$color_0 \cap color_1 = \varnothing$$
$$(x_0, y_0) \cap color_1 = \varnothing$$

Process P_0 and P_1 can be run independently, because all Bernstein's conditions are satisfied. Therefore, the parallel algorithm can be applied to Mandelbrot set plot.

B. Theoretical Speed up of Mandelbrot set plot

In order to compute the theoretical speed up of the Mandelbrot set plot applying parallel algorithm with different number of processors, a given sequential-based algorithm can be divided it into two portions: (1) write the Mandelbrot set into the binary file (2) Run Mandelbrot set computation. In the given C language file, these two portions

are executed sequentially, but only the total process time is recorded. The original file is modified to record both the total process time and the Mandelbrot set computation time, the result is provided in Table A in the appendices.

In parallel computing, Amdahl's law is mainly used to predict the theoretical maximum speedup for program processing using multiple processors (Amdahl's Law, 2019). The workload of a task is fixed, and it consists of parallelizable portion and non-parallelizable portion. Since the parallelizable portion of workload can be distributed to different processors, the purpose of speed up in the execution of one task can finally achieved through increasing the number of processors. Equation (3) denotes how to calculate the theoretical speed up factor.

$$S(p) = 1 / (r_s + r_p/r_p) \quad (3)$$

with

r_p = parallel ratio (parallelizable portion)
r_s = serial ratio (non – parallelizable portion)
p = number of processors

The non-parallelizable portion in Mandelbrot set plot is creating and writing a binary file and the parallel portion is Mandelbrot set computation which can be parallelized using multiple processors. Table 1 shows the theoretical speed up factors S(p) calculated based on the recorded execution time (see Table A in the appendices) for the number of processors p from 2 to 8.

TABLE I THERORETICAL SPEED UP FACTOR USING AMDAHL'S LAW

Number of processors, p	2	4	6	8
Speed up factor, S(p)	1.922	3.565	4.986	6.227

III. DESIGN OF PARTITION SCHEMES

No matter what the partition scheme is, the root node creates a binary file to plot Mandelbrot set at first. Each processor creates a 1D dynamic array, it is used to store color values that are computed by a processor. Another 1D dynamic array is created only by the root processor, it is able to store all color values that are used to plot Mandelbrot set. After each processor completes Mandelbrot set computation based on the partition scheme respectively, all data are sent back to the root node. The root node is responsible for writing each pixel color to the binary file serially.

A. Row Segmentation-based Partition Scheme

For row segmentation-based partition scheme (see Figure 1), each processor has a number of rows. The number of rows per processor is calculated using equation (4) whose result is the quotient to the division whereas the number of remained rows is calculated using equation (5) whose result is the remainder of the division.

Rows per processor = Maximum Y value / Number of Processors (4)

Rows remained = Maximum Y value % Number of Processors (5)

A row segmentation-based partition scheme has been applied to accelerate the process of matrix multiplication (Ng & Baskaran, 2019), but the row segmentation-based partition scheme which used here is different. Instead of distributing a number (i.e. *rows per processor*) of adjacent rows of points to each processor, each processor takes turn to take one row of points into its local buffer at a time until all rows are taken. Rows that belongs to a processor are nonadjacent. Upon finishing the Mandelbrot set computation, root node will gather a number (i.e. maximum x value) of color values among processors iteratively. Root node will write all color values into the binary file after these values are gathered. Figure 2 depicts the flowchart of row segmentation-based partition scheme.

p0	p0	p0	p0	p0	p0	p0	p0	p0
p1	p1	p1	p1	p1	p1	p1	p1	p1
p2	p2	p2	p2	p2	p2	p2	p2	p2
p3	p3	p3	p3	p3	p3	p3	p3	p3
p0	p0	p0	p0	p0	p0	p0	p0	p0
p1	p1	p1	p1	p1	p1	p1	p1	p1
p2	p2	p2	p2	p2	p2	p2	p2	p2
p3	p3	p3	p3	p3	p3	p3	p3	p3
p0	p0	p0	p0	p0	p0	p0	p0	p0
p1	p1	p1	p1	p1	p1	p1	p1	p1

Assume maximum x value iXmax = 9, maximum y value iYmax = 9, and number of processors P = 4

Fig. 1 Row Segmentation-based Partition Scheme

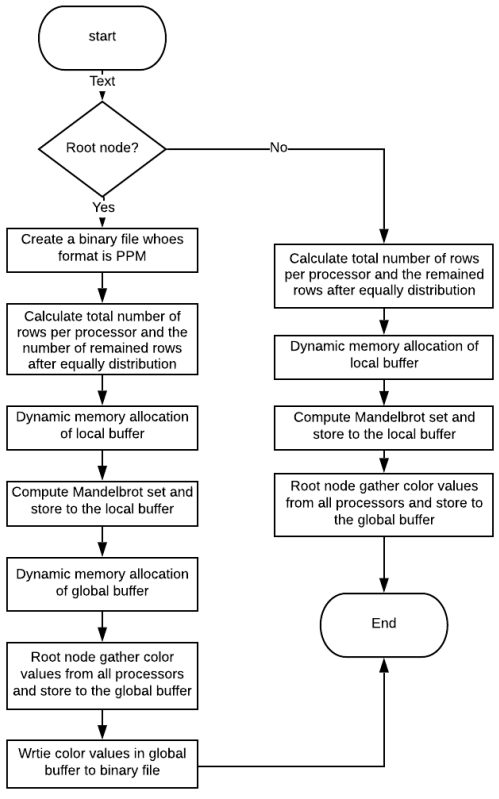


Fig. 2 Flowchart of Row Segmentation-based Partition Scheme

B. *Point-based Partition Scheme*

In point-based partition scheme (see Figure 3), every processor acquires a point at a time sequentially until there is no more point. At last, every processor will have a number of points, the number is determined by *points per processor* and *points remained*. The equation (6) and (7) describe how to calculate them. The *points per processor* is the quotient produced by dividing the total number of points by the number of processors, and the points remained is remainder after dividing the total number of points by the number of

Points per processor = ((Maximum X value*Maximum Y value)/Number of Processors (6)

Points remained = (Maximum X value*Maximum Y value) % Number of Processors (7)

processors. Upon completing Mandelbrot set computation, the root node starts gathering one color value among processors at a time till all data are sent to the root node. As soon as the root node receives all color values from all processors, it will write the content into the binary file to plot Mandelbrot set. The flowchart of points-based partition scheme is illustrated in Figure 4.

p0	p1	p2	p3	p0	p1	p2	p3	p0
p1	p2	p3	p0	p1	p2	p3	p0	p1
p2	p3	p0	p1	p2	p3	p0	p1	p2
p3	p0	p1	p2	p3	p0	p1	p2	p3
p0	p1	p2	p3	p0	p1	p2	p3	p0
p1	p2	p3	p0	p1	p2	p3	p0	p1
p2	p3	p0	p1	p2	p3	p0	p1	p2
p3	p0	p1	p2	p3	p0	p1	p2	p3
p0	p1	p2	p3	p0	p1	p2	p3	p0
p1	p2	p3	p0	p1	p2	p3	p0	p1

Assume iXmax = 9, iYmax = 9, and number of processors P = 4
Fig. 3 Point-based Partition Scheme

IV. IMPLEMENTATION OF PARTITION SCHEMES

The input values that are used to test the performance of Mandelbrot set plot are the maximum x value, the maximum y value and the max iteration number, they are 8000, 8000 and 2000 respectively. The implementation of the parallelized Mandelbrot set plot algorithms is based on C language using of MPICH 4.0.1. The computer is used for testing is iMac. It runs on macOS 10.14.6. The specification of computer is recorded in Table A in the appendices.

These two parallelized Mandelbrot set plot algorithms are tested in four different cases: (1) run these two programs with 2 logical processors (2) run these two programs with 4 logical processors (3) run these two programs with 6 virtual processors (4) run these two programs with 8 virtual processors.

V. RESULTS AND DISSCUSSIONS

The overall time of execution Mandelbrot plot and communication time among processors are recorded and tabulated in Table B and Table C.

The actual speed up factor of the parallelized Mandelbrot

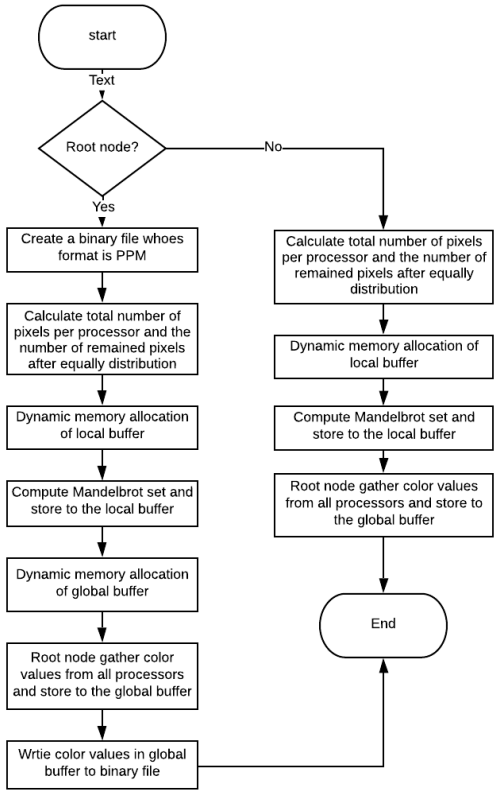


Fig. 4 Flowchart of Point-based Partition Scheme

set plot algorithm can be calculated using equation (8). The calculated actual speed up factors are provided in Table II and Graph I illustrates the difference between theoretical speed up factors and actual speed up factors for both two partition schemes.

$$S(P) = t_s / t_p \quad (8)$$

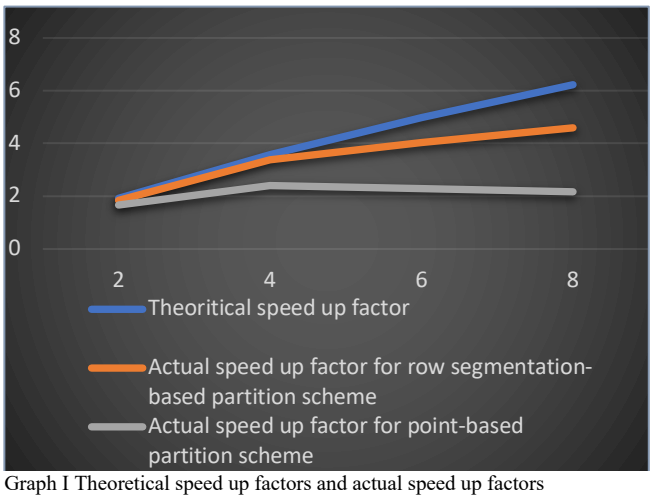
with

t_s: execution time of program implementing serial algorithm

t_p: execution time of program implementing parallel algorithm.

Table II Actual speed up factor
** All times are expressed in seconds

Number of processors	2		4		6		8	
Theoretical Speed up factor	1.922		3.565		4.986		6.227	
	actual speed up factor	difference	actual speed up factor	difference	actual speed up factor	difference	actual speed up factor	difference
Row segmentation-based partition scheme	1.843	4.08%	3.381	5.15%	4.030	19.17 %	4.593	26.24 %
Point-based partition scheme	1.667	13.27 %	2.405	32.54 %	2.291	54.05 %	2.165	65.23 %



Graph I Theoretical speed up factors and actual speed up factors

From Table II and Graph I, it can be noticed that the actual speed up factors of both two different partition schemes tested using different number of processors are all blow the theoretical speed up factors. The reason for the difference between theoretical speed up factor and actual speed up factor is the communication among processors. Amdahl's law does not consider the potential communication overheads as a non - parallelizable portion when calculating the theoretical speed up factor. For the same parallel algorithm, the improvement of speed up factor is non-linear because using more processors cause more communication among them which can lead to more latency. Nevertheless, the parallel algorithm using row segmentation-based partition scheme decreases the execution time by 78.38%.

In addition, it is observed that actual speed up factor using point-based partition scheme is dramatically different from that using row segmentation-based partition scheme. The Mandelbrot set computation time can be roughly calculated using equation (9) and *overall time* and *communicate time* are recorded in Table B and Table C, the results are recorded in Table III.

computation time = overall time – communicate time (9)

Table III Mandelbrot set computation time for two partition schemes
** All times are expressed in seconds

Number of processors	2	4	6	8
Row segmentation-based partition scheme	38.530139	20.968526	17.513204	15.311612
Point-based partition scheme	38.748756	21.258170	17.708879	15.627775

The difference of Mandelbrot set computation time between these two partition schemes is subtle, while the communication overheads of row segmentation-based partition scheme are far less than that of point-based partition scheme. As a result, the row segmentation-based partition scheme produces a better performance.

VI. CONCLUSIONS

The report presents the parallel Mandelbrot set plot algorithm. Two partition schemes are designed and implemented successfully. From the experimental results, it is observed that the speed of Mandelbrot set plot program has been increased around 4 times. The difference between the actual speed up factors and the theoretical speed up factor can be attributed to potential communication overheads among multiple processors. Row segmentation-based partition scheme is more efficient than point-based partition scheme since it reduces the times of gathering data from all processors. This indicates that how processors communicate with each other should be taken into account when designing and developing a parallel algorithm. In conclusion, the object of applying Mandelbrot set plot parallel algorithm has been achieved via utilizing MPI, and the performance improvement has been observed and causes of difference between theoretical speed up factor and actual speed up factor are discovered.

To further optimize the Mandelbrot set plot program, the partition of writing file can be parallelized with the use of parallel I/O. Another possible method which can improve the performance is to apply non-blocking communications instead of blocking communications used in this report.

REFERENCES

Amdahl's Law. (2019). Retrieved from Techopedia: https://www.techopedia.com/definition/17035/amdahls-law

BravoYusuf. (2019). *Parallelism in Algorithms-Bernstein's Conditions.* Retrieved from SCRIBD: https://www.scribd.com/doc/53645670/Parallelism-in-Algorithms-Bernstein-s-Conditions

Devaney, R. (2019). *What is the Mandelbrot set*. Retrieved from Plus Magazine: https://plus.maths.org/content/what-mandelbrot-set

Ng, C.-K., & Baskaran, V. M. (2019). *Design and implementation of Parallel Matrix Multiplication Algorithms using Message Passing Interface.*

Parallel Algorithm-Introduction. (2019). Retrieved from Tutorialspoint: https://www.tutorialspoint.com/parallel_algorithm/parallel_algorithm_introduction.htm#

APPENDIX

TABLE A (SERIAL IMPLEMENTATION)
** All times are expressed in seconds

Computer specifications	a) CPU model: Intel Core i7-7700k b) Number of logical processors: 4 c) Memory size: 40 GBytes of RAM		
Value of iXmax	8000(default)		
Value of iYmax	8000(default)		
Value of IterationMax	2000(default)		
	Serial Program		
	Write	Mandelbrot set computation	Overall time (computation & write)
Run # 1	2.851555	68.661890	71.513445
Run # 2	2.903204	68.600039	71.503243
Run # 3	2.917741	68.644029	71.561770
Run # 4	2.893223	68.790821	71.684044
Run # 5	3.013517	69.204181	72.216698
Average time	2.915648	68.780192	71.695840

TABLE B (Row Segmentation-based Partition Scheme)
** All times are expressed in seconds

Computer specifications	a) CPU model: Intel Core i7-7700k b) Number of logical processors: 4 c) Memory size: 40 GBytes of RAM			
Value of iXmax	8,000 (default)			
Value of iYmax	8,000 (default)			
Value of IterationMax	2,000 (default)			
	Parallel Program			
	MPI			
	2 logical processors		4 logical processors	
	Communicate time	Overall time	Communicate time	Overall time
Run #1	0.082314	38.612430	0.082388	21.373004
Run #2	0.086158	38.418458	0.082422	20.978286

Run #3	0.093426	38.93592	0.084253	20.990944
Run #4	0.086371	38.261781	0.082914	20.855657
Run #5	0.084725	38.855099	0.089852	21.066578
Average time	0.086599	38.616738	0.084366	21.052894
	6 virtual processors		8 virtual processors	
	Communicate time	Overall time	Communicate time	Overall time
Run #1	0.088204	17.754037	0.300170	15.602742
Run #2	0.164198	17.511341	0.256480	15.446366
Run #3	0.164668	17.466727	0.093728	15.691628
Run #4	0.187885	17.690017	0.171934	15.385143
Run #5	0.148175	17.897028	0.121583	15.376074
Average time	0.150626	17.663830	0.188779	15.500391

TABLE C (Point–based Partition Scheme)
** All times are expressed in seconds

Computer specifications	a) CPU model: Intel Core i7-7700k b) Number of logical processors: 4 c) Memory size: 40 GBytes of RAM			
Value of iXmax	8,000 (default)			
Value of iYmax	8,000 (default)			
Value of IterationMax	2,000 (default)			
	Parallel Program			
	MPI			
	2 logical processors		4 logical processors	
	Communicate time	Overall time	Communicate time	Overall time
Run #1	3.875073	42.624465	8.257795	29.460032
Run #2	4.038245	42.743840	8.084383	28.951957
Run #3	3.818280	41.999712	9.800529	31.380361
Run #4	3.995813	42.983695	10.074945	31.717776
Run #5	4.072763	43.192240	5.505521	26.503897
Average time	3.960035	42.708790	8.344635	29.602805
	6 virtual processors		8 virtual processors	
	Communicate time	Overall time	Communicate time	Overall time
Run #1	13.663233	31.277964	17.211053	33.525076
Run #2	13.715073	31.690808	18.122281	33.979884
Run #3	13.465115	31.121142	18.441204	34.467479
Run #4	13.059542	30.875376	15.529056	30.902884
Run #5	12.926192	30.408262	16.942755	31.509900
Average time	13.365831	31.074710	17.249270	32.877045