Design and Implementation of Mandelbrot Set Calculation Algorithms using Message Passing Interface

Zhihao Pei

Monash University

Melbourne Australia

zpei0001@student.monash.edu

*Abstract*— Mandelbrot set calculation is one of classic computing problems, and also a perfect example of embarrassingly parallel algorithm. This report aims at finding and implementing an optimal partition scheme to achieve the parallelization of the Mandelbrot set calculation in C language using Message Passing Interface (MPI). This report starts with the introduction to the basic concept, and it is followed by the application of Bernstein Condition and Amdahl’s law to verify the parallelizability and theoretical speed up. By comparing multiple partition schemes, one with the best performance is selected, implemented, and tested on a single multicores computer with multiple processors. After experiments, the actual speed up is compared with the theoretical one. By observing the time taken by each portion of the program, analysis and possible future work are discussed at the end of the report.

Keywords— Mandelbrot set, Parallel computing, MPI, Partition scheme, Speed up prediction and analysis

1. INTRODUCTION

The Mandelbrot set is a fractal which exhibits a complex but self-similar structure of boundary at various scales [1]. However, the small-scale characteristics have been proved not identical to the whole, which means Mandelbrot set is actually infinitely complicated rather than recursively constructed [2]. Because it is regarded as a source of infinite and eternal beauty, the Mandelbrot set is even compared to “the thumbprint of god” [3]. Surprisingly, such an exquisite object is generated from an extremely simple function.

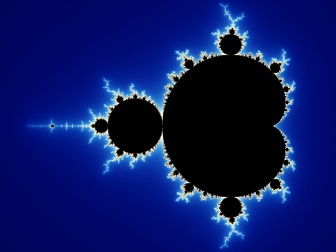


Fig. 1. Mandelbrot set

Mathematically, the Mandelbrot set is a visual representation of a set of complex numbers C for which an iterated function:

does not diverge within N iterations starting with Z0 = 0 [4]. In the other words, for any complex number C, there is a sequence number Z­0 … ZN where

Z0 = 0

Z1 = Z02 + C,

Z2 = Z12 + C,

…,

ZN = ZN-12 + C

If none of the numbers Z0, …, ZN escapes from some predefined bounded neighbourhood of 0, the function is said not diverging and the corresponding C is in the Mandelbrot set, otherwise the C is outside the set. As the calculation for each C is independent, the Mandelbrot set calculation is a perfect example of the embarrassingly parallel algorithm. Therefore, it was chosen as the topic.

This report firstly discusses multiple partition schemas for the Mandelbrot set calculation algorithm and the best one is selected. The chosen one, which determines how all pixels in the image are assigned to each processor to parallelize the calculation, has been implemented by using the Message Passing Interface library in C language. Then the report evaluates the actual performance by comparing it with the theoretical speed up calculated based on the Amdahl’s law. After that, the results and analysis are given to explain the observations. Eventually, the report ends with the conclusion and valid further work.

Two hypotheses are applied in this report. The first one is that the performance of a processor is not affected by the parallelization. It means the computation in the processor should take the same time no matter the processor works alone or in parallel. Only in this case, the Amdahl’s law is valid for the evaluation. The second one is the workload should be balanced for each processor to achieve the best performance [1]. Otherwise, it will be inevitable that some processors wait upon others to finish and therefore leads to the serial computing.

1. PRELIMINARY ANALYSIS
2. *Parallelizability of Mandelbrot Set*

Before the design of parallelization of the Mandelbrot set is introduced, it is necessary to verify the parallelizability of the computing process. Based on Bernstein’s conditions, any two processes Pi and Pj, which are expected to work simultaneously and independently, must satisfy three conditions [5]:

Ii ∩ Oj = Ø

Ij ∩ Oi = Ø

Oi ∩ Oj = Ø

where Ii and Oi are the input and output of the first process Pj, and Ij and Oj are the input and output of the second process Pj.

To draw a visual representation of the Mandelbrot set, the processors must calculate the values of a sequence Z(n) for each number C on the complex plane, and determine the color of the corresponding pixel. Therefore, they have the following iterated functions:

Zy(n) = 2\*Zx(n-1)\*Zy(n-1) + Cy

Zx(n) = Zx(n-1)2 – Zy(n-1)2 + Cx

Zy(0) = 0

Zx(0) = 0

and

Ii = Cyi, Cxi, Oi = Zyi(k), Zxi(k)

Ij = Cyj, Cxj, Oj = Zyj(k), Zxj(k)

Where Z(k) = Zx(k)2 – Zy(k)2 is greater than a predefined value ER2 (160000 in this report) or k is equal to maximum iteration N (2000 in this report).

Let Si(C) be the set of C values assigned to processor i for calculation. For any two processor i and j, there must be

Si(C) ∩ Sj(C) = Ø

because each C should only be calculated once. Besides, as the sequence Z(n) always starts with 0, the values of Z(n) only depend on Cx and Cy. Therefore, it has:

Cyi, Cxi ∩ Zyj(k), Zxj(k) = Ø

Cyj, Cxj ∩Zyi(k), Zxi(k) = Ø

And

Zyi(k), Zxi(k) ∩ Zyj(k), Zxj(k) = Ø

Since all the conditions are met, the process of the Mandelbrot set calculation is proven parallelizable [5], i.e., any two processors are able to work simultaneously and independently on different C values.

1. *Theoretical Speed Up of Mandelbrot Set Calculation*

Amdahl’s law is applied here to predict the theoretical speed up of the parallelization of the Mandelbrot set calculation [6] based on a serial-based algorithm provided by the supervisor.

where

S(p) = the theoretical speed up

= serial-only section taking time / overall time

= parallelizable section taking time / overall time

P = the number of the processes

To apply the law, the serial-only code and parallelizable computing should be considered separately [6]. This algorithm consists of three parts, (1) writing ASCII header to the file, (2) computing to determine the color for each pixel, (3) writing color to the file. (1) and (3) can be considered as the serial-only code because a single output file is required. The computing algorithm (2) is parallelizable, which has been proven in the last section.

Table I shows the average time taken to run different parts of the serial program. In this case, the rs in the Amdahl’s law refers to the portion for (1) initiation and (3) writing color data into the output file, and the r­p is represented by the portion for (2) parallelizable Mandelbrot set calculation.

Based on the Amdahl’s law, the results are shown in Table II.

TABLE I

TIME TAKEN BY THE SERIAL PROGRAM

|  |  |  |  |
| --- | --- | --- | --- |
| **Section** | Initiation | Calculation | Writing |
| **Time (s)** | Close to 0 | 111.444 | 2.342 |

TABLE II

THEORETICAL SPEED UP FACTOR USING AMDAHL’S LAW

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number of Processors, p** | 2 | 4 | 6 | 8 | 10 |
| **Speed Up Factor, S(p)** | 1.96 | 3.77 | 5.44 | 6.99 | 8.44 |

1. DESIGN OF THE PARTITION SCHEME

Based on the Amdahl’s law introduced in the last section, the only way to speed up a serial algorithm is to divide the parallelizable computing into P processors to be computed simultaneously. Ideally, it only takes each processor t/P time to finish the parallelizable work which takes a serial program t time. In this case, each processor is expected to spend the same time, otherwise there must be a processor taken more than t/P time to finish the computation while others are waiting on it, and therefore compromises the overall performance.

1. *Design of The Partition scheme*

The scheme applied in this report is balanced row segmentation-based partition scheme. First of all, the root node will open and write the ASCII header to the output file. Meanwhile, each processor will find the rows of image assigned to it for computing based on two equations.

(1)

(2)

To achieve the balance of the workload, the remainder will be assigned to the first *row\_remain* processors. Based on the results of (1) and (2), each non-root processor will create an 1D dynamic arrays to store all color data computed in it, except that the root processor needs to hold a large enough array for gathering all the outputs. Using 1D dynamic arrays enhances the ease of data access, allocation, and deallocation. Once the preparation is finished, processors will start the computation following the partition scheme below:

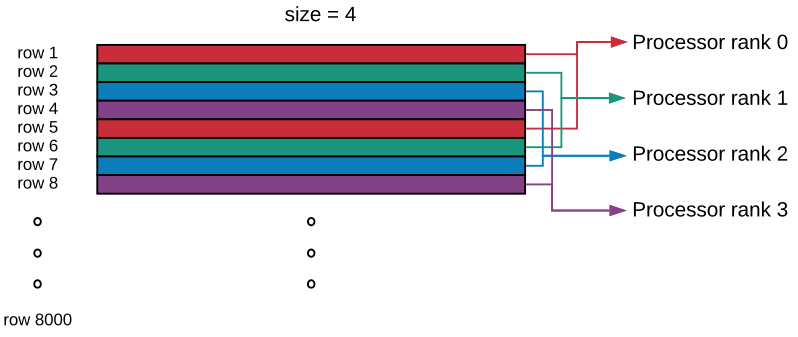


Fig. 2. Balanced Row Segmentation-based Partition Scheme

As shown in the diagram, each processor will be responsible for every *sizeth* row of the image (e.g. *x = rank, rank+size, rank+2\*size, …*). Given that the output image of the Mandelbrot set is symmetric about x-axis and has 8,000, it should be enough to achieve highly balanced workload distribution for each processor. After finishing the computation, all non-root processors will send the data stored locally to the root node. The root node will write the data into the output file in the correct order after it receives all the data. The flow chart below captures the details of the algorithm.

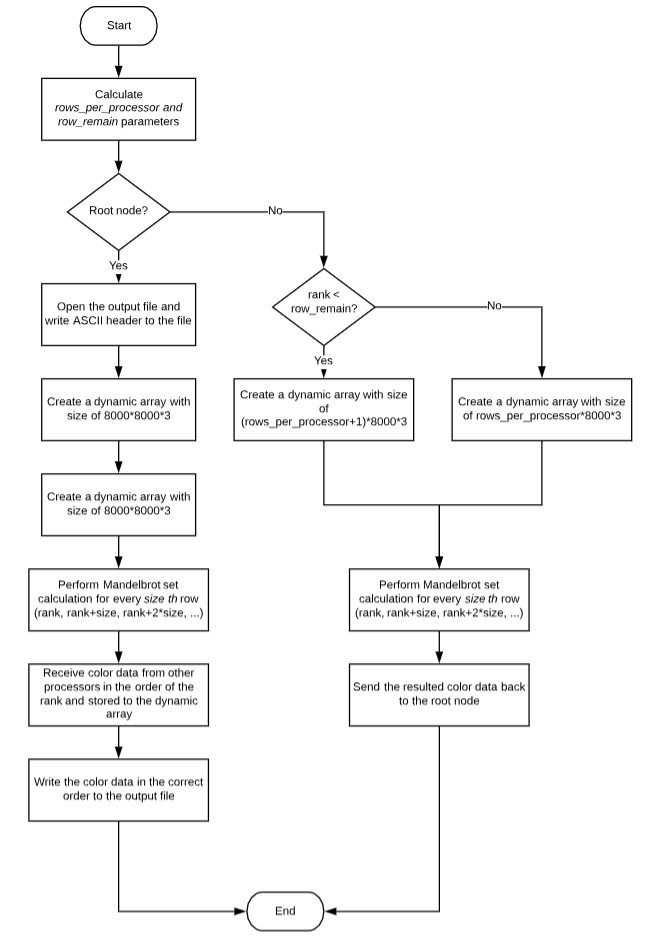


Fig. 3. Flowchart of the Partition Scheme

1. *Comparison with other parallel partitioning designs*

* Row segmentation-based partition. As described at the start of this section, naïve row segmentation-based partition cannot achieve balanced workload distribution.

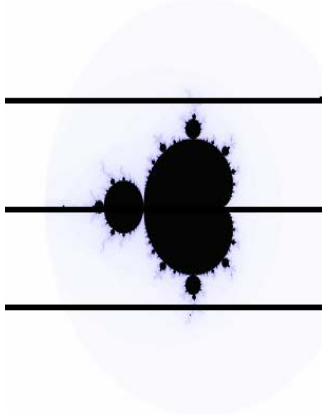


Fig. 4. Row segmentation-based partition

For the Mandelbrot set calculation, black areas will take much more time to compute than the other parts. If running the parallel program with 4 processors as shown in the figure 3, rank 1 and 2 will do the majority of the job while keeping the other two waiting. In this case, the computation section will take almost double time compared with the balanced one.

* Pixel (Tile)-based Partition Scheme. This approach refers to assigning every nth pixel to each processor, where n is the number of the processors. Although it can theoretically achieve the most balanced distribution of the workload, the actual performance is still slightly worse than the selected one. The reason is that this approach reduces the computing time but also increases the writing time. Since adjacent pixels are assigned to different nodes, their color data is not stored contiguously in the memory. Therefore, reading and writing process will be inefficient.
* Parallel writing. Letting all processors write multiple files parallelly seems to be a way to add the parallelizability to the program. However, since one single output file is required, reading and merging all subfiles cannot be avoided, which takes more time.

1. IMPLEMENTATION OF PARTITION SCHEMES

The output Mandelbrot set is predefined of size 8000 × 8000. For each pixel in the image, there is a corresponding C value determining the color. Among all possible C, the x-value is from -2.5 to 1.5, and the y-value is from -2i to 2i. Each of the C value consists of three color components r, g, and b, which are coded from 0 to 255. The color component is of variable type *unsigned char* (i.e. 1 byte), which minimises the size of the storage. Implementation of the parallelization of the Mandelbrot set calculation is based on C language and Message Passing Interface.

In the parallel program, the initiation and writing process is executed serially, while the iterated function computing is assigned to different number of processors (i.e. 2, 4, 6, 8, 10). All the test cases are run 5 times on the same multicores node from MonARCH

1. RESULTS AND DISCUSSIONS

Table A – E in the appendices record the time taken by each part of the programs for each test case. Besides, the average speed up factor for different number processors are listed below

TABLE III

ACTUAL SPEED UP FACTORS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number of Processors, p** | 2 | 4 | 6 | 8 | 10 |
| **Actual Speed Up Factors, S(p)** |  |  |  |  |  |

By comparing Table II and Table III, it can be noticed that the actual speed up factors are always slightly smaller than the theoretical ones, but the function S(p) still converges to linearity.