TERM PROGECT

Group #3

Contents

Question & Solution

Question 1:

Load balancing issues

Solution:

Our solution is to divide the 1000 × 1000 matrix into different matrices according to the rows, and assign these split matrices to different processors.

The processor processes the points in the assigned matrix, and after the Orb (z0) operation, the final result of each point is obtained.

```c
void init_indicesNlengths(int* indices, int* lengths, int remainder, int sub_rows, int num_procs)
{
    for (int i = 0; i < num_procs; i++)
    {
        if (i < remainder)
            lengths[i] = sub_rows + 1;
        else
            lengths[i] = sub_rows;
    }
    for (int i = 0; i < num_procs; i++)
        indices[i] = i * sub_rows + MIN(i, remainder);
}
```

Then processor 0 broadcasts the initialized array information to all processors.

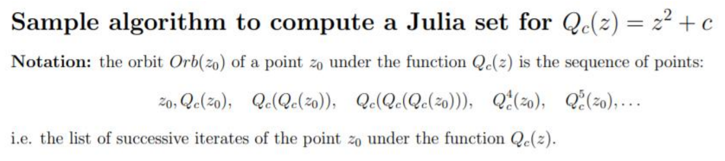
```c
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
int remainder = ROWS % num_procs;
int sub_rows = ROWS / num_procs;

int* indices = malloc(sizeof(int) * num_procs);
int* part_lengths = malloc(sizeof(int) * num_procs);

if (rank == 0)
    init_indicesNlengths(indices, part_lengths, remainder, sub_rows, num_procs);
MPI_Bcast(indices, num_procs, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(part_lengths, num_procs, MPI_INT, 0, MPI_COMM_WORLD);
```

Question 2:

How to implement Orb (z0) in code and how to map the value of a point to a graph

Solution:

According to the PDF



Sample algorithm to compute a Julia set for $Q_c(z) = z^2 + c$

Notation: the orbit $Orb(z_0)$ of a point z_0 under the function $Q_c(z)$ is the sequence of points:

$$z_0, Q_c(z_0), \quad Q_c(Q_c(z_0)), \quad Q_c(Q_c(Q_c(z_0))), \quad Q_c^4(z_0), \quad Q_c^5(z_0), \ldots$$

i.e. the list of successive iterates of the point z_0 under the function $Q_c(z)$.

So after implementing the formula, we can iterate and calculate the required value.

```
for (int i = local_start; i < local_end; i++)
    {
        for (int j = 0; j < COLUMNS; j++)
        {
            zx = (float)(i + 1 - 500) / 250;
            zy = (float)(j + 1 - 500) / 250;

            color = 0;

            while ((float)zx * zx + zy * zy <= 4.00 && color <= iterations)
            {
                temp = (float)zx * zx - zy * zy;
                zy = (float)2 * zx * zy + cy;
                zx = (float)temp + cx;
                color++;
            }

            local_colors[counter] = color;
            counter++;
        }
    }
```

During the iteration, if the points obtained after the iteration are outside the specified range, the iteration is stopped, and the number of iterations is stored in the one-dimensional array of the processor.

If after 50 iterations, the obtained point is still not out of range, stop the iteration and also store times of the iterations in the corresponding position of dimensional array.

Question 3:

How to centralize the data obtained by each processor.

Solution:

Just like we always do, we transfer the data obtained by the processors to PROCESS_0 and let it integrate the data.
And we use MPI_Gatherv()

```c
MPI_Gatherv(local_colors, local_size, MPI_INT, colors, color_part_lengths, color_indices, MPI_INT, 0, MPI_COMM_WORLD);

if (rank == 0)
{
    array_to_file(colors);

    free(colors);
    free(color_indices);
    free(color_part_lengths);
}
```

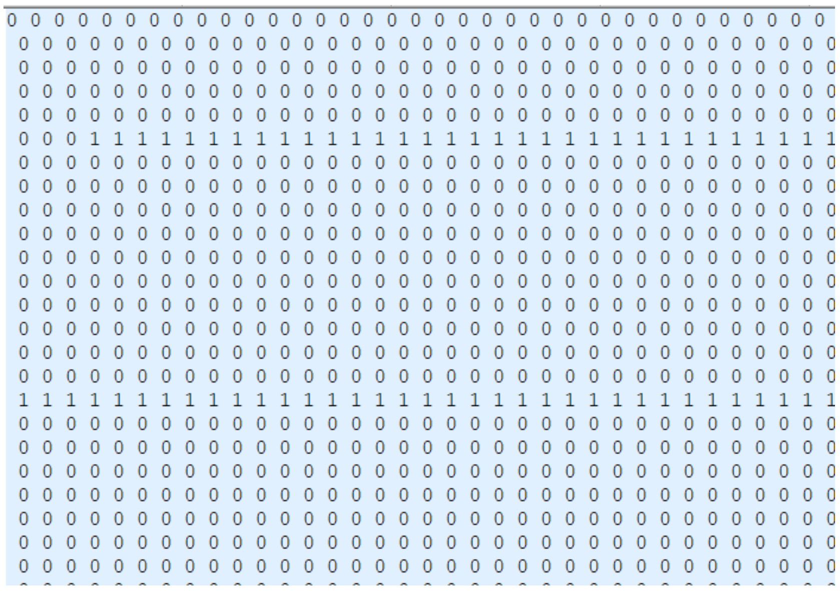
Question 4:

How to store data in files for later image processing.

Solution:

After storing the data in an array, use fopen to store the data in a file

```c
void array_to_file(int* array)
{
    FILE* out = fopen("result.txt", "w");

    int i = 0;
    for (i = 0; i < ROWS * COLUMNS; i++) {
        fprintf(out, "%d ", array[i]);
    }

    fclose(out);
}
```



```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Partial screenshot of the data file we got

Question 5:

How to draw a picture based on the value of each point in the file.

Solution:

First, we decide which color is based on the number of iteration times.

```c
int get_color(int ite_time)
{
    if (ite_time - 1 == 50)
        return 0;

    int temp = (ite_time - 1) % iterations;

    if (temp >= 0 && temp <= 10)
        return 1;
    else if (temp > 10 && temp <= 20)
        return 2;
    else if (temp > 20 && temp <= 30)
        return 3;
    else if (temp > 30 && temp <= 40)
        return 4;
    else
        return 5;
}
```

Second, integrate and draw pictures based on the obtained colors using OpenGL.

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    // read the file
    FILE* input = fopen("result.txt", "r");

    printf("Starting to read file...\n");

    int num;
    int color;
    for (int i = 0; i < ROWS; i++)
    {
        for (int j = 0; j < COLUMNS; j++)
        {
            fscanf(input, "%d", &num);

            color = get_color(num);

            if (color == 0)
                glColor3f(0.0, 0.0, 0.0);
            else if (color == 1)
                glColor3f(0.0, 0.0, 1.0);
            else if (color == 2)
                glColor3f(0.0, 1.0, 0.0);
            else if (color == 3)
                glColor3f(0.0, 1.0, 1.0);
            else if (color == 4)
                glColor3f(1.0, 0.0, 0.0);
            else
                glColor3f(1.0, 0.0, 1.0);

            glBegin(GL_POINTS);
            glVertex2i(i, j);
            glEnd();
        }
    }
```

Question 6:

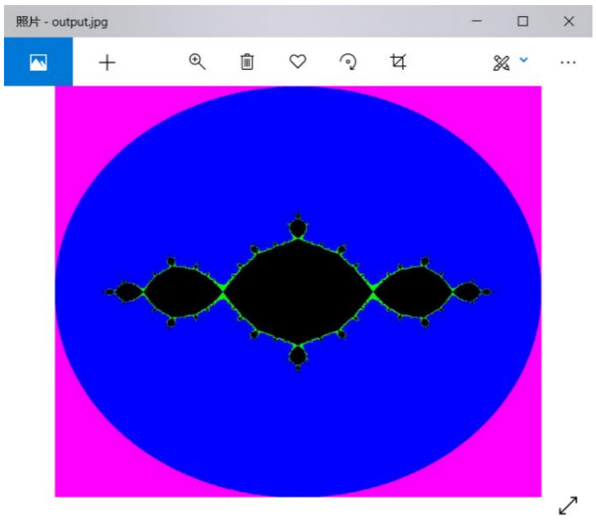
How to output a picture as a .jpg file.

Solution:

We used the “FreeImage” library
And create the “grab” function to get the final output .jpg file

```
void grab()
{
    unsigned char *mpixels = malloc(sizeof(unsigned char) * ROWS * ( COLUMNS - 155 ) * 3);
    glReadBuffer(GL_FRONT);
    glReadPixels(0, 0, ROWS, ( COLUMNS - 155 ), GL_RGB, GL_UNSIGNED_BYTE, mpixels);
    glReadBuffer(GL_BACK);
    for(int i = 0; i < (int)ROWS*( COLUMNS - 155 )*3; i += 3)
    {
        mpixels[i] ^= mpixels[i+2] ^= mpixels[i] ^= mpixels[i+2];
    }
    FIBITMAP* bitmap = FreeImage_Allocate(ROWS, ( COLUMNS - 155 ), 24, 8, 8, 8);

    for(int y = 0 ; y < FreeImage_GetHeight(bitmap); y++)
    {
        BYTE *bits = FreeImage_GetScanLine(bitmap, y);
        for(int x = 0 ; x < FreeImage_GetWidth(bitmap); x++)
        {
            bits[0] = mpixels[(y*COLUMNS+x) * 3 + 0];
            bits[1] = mpixels[(y*COLUMNS+x) * 3 + 1];
            bits[2] = mpixels[(y*COLUMNS+x) * 3 + 2];
            bits += 3;
        }
    }
    FreeImage_Save(FIF_JPEG, bitmap, "output.jpg", JPEG_DEFAULT);
    FreeImage_Unload(bitmap);
}
```

Then we can get the picture



The output image is a .jpg file

Source Code

Dear professor, please be careful when copying the code to run on your own computer. It may have some syntax errors because of the copying operation. But it will usually work I think.

Julia.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <math.h>

#define ROWS 1000
#define COLUMNS 1000
#define iterations 50

#define MIN(a,b) (a < b ? a : b)

void array_to_file(int* array)
{
    FILE* out = fopen("result.txt", "w");

    int i = 0;
    for (i = 0; i < ROWS * COLUMNS; i++) {
        fprintf(out, "%d ", array[i]);
    }

    fclose(out);
}

void init_indicesNlengths(int* indices, int* lengths, int remainder, int sub_rows, int num_
procs)
{
    for (int i = 0; i < num_procs; i++)
    {
        if (i < remainder)
            lengths[i] = sub_rows + 1;
        else
            lengths[i] = sub_rows;
    }

    for (int i = 0; i < num_procs; i++)
        indices[i] = i * sub_rows + MIN(i, remainder);
}

int main(int argc, char** argv) {
    int num_procs;
    int rank;

    float cx = atof(argv[1]);
    float cy = atof(argv[2]);

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
    MPI_Barrier(MPI_COMM_WORLD);
    double elapsed_time = -MPI_Wtime();

    int remainder = ROWS % num_procs;
    int sub_rows = ROWS / num_procs;

    int* indices = malloc(sizeof(int) * num_procs);
    int* part_lengths = malloc(sizeof(int) * num_procs);

    if (rank == 0)
        init_indicesNlengths(indices, part_lengths, remainder, sub_rows, num_procs);

    MPI_Bcast(indices, num_procs, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(part_lengths, num_procs, MPI_INT, 0, MPI_COMM_WORLD);

    int local_row = part_lengths[rank];
    int local_size = local_row * COLUMNS;
    int* local_colors = malloc(sizeof(int) * local_size);


    int local_start = indices[rank];
    int local_end = local_start + local_row;

    float zx, zy;
    float temp;

    int color;
    int counter = 0;
    for (int i = local_start; i < local_end; i++)
    {
        for (int j = 0; j < COLUMNS; j++)
        {
            zx = (float)(i + 1 - ROWS / 2) / (ROWS / 4);
            zy = (float)(j + 1 - COLUMNS / 2) / (COLUMNS / 4);

            color = 0;

            while ((float)zx * zx + zy * zy <= 4.00 && color <= iterations)
            {
                temp = (float)zx * zx - zy * zy;
                zy = (float)2 * zx * zy + cy;
                zx = (float)temp + cx;
                color++;
            }

            local_colors[counter] = color;
            counter++;
        }
    }


    int* colors;
    int* color_indices;
    int* color_part_lengths;
    if (rank == 0)
    {
        colors = malloc(sizeof(int) * ROWS * COLUMNS);
        color_indices = malloc(sizeof(int) * num_procs);
        color_part_lengths = malloc(sizeof(int) * num_procs);
```

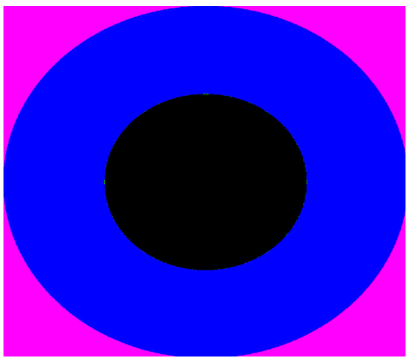
```c

        for (int i = 0; i < num_procs; i++)
        {
            color_part_lengths[i] = part_lengths[i] * COLUMNS;
            color_indices[i] = indices[i] * COLUMNS;
        }
    }

    MPI_Gatherv(local_colors, local_size, MPI_INT, colors, color_part_lengths, color_indices, MPI_INT, 0, MPI_COMM_WORLD);

    if (rank == 0)
    {

        array_to_file(colors);

        free(colors);
        free(color_indices);
        free(color_part_lengths);
    }
    free(indices);
    free(part_lengths);
    free(local_colors);

    printf("\nProcessor %d done!\n", rank);

    // get the elapsed time for benchmarking
    elapsed_time += MPI_Wtime();
    printf("The elapsed time is: %lf\n", elapsed_time);

    MPI_Finalize();

    return 0;
}
```

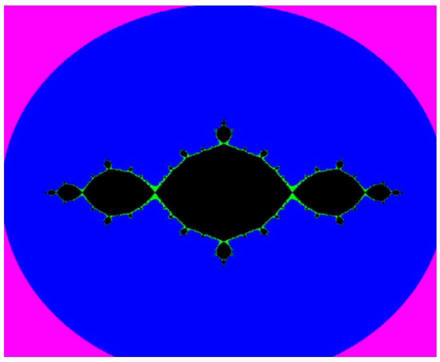
points.c

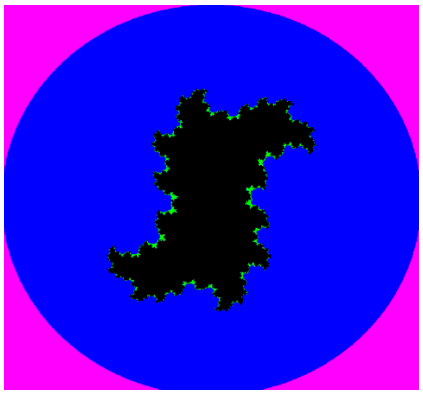
```c
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <FreeImage.h>
```

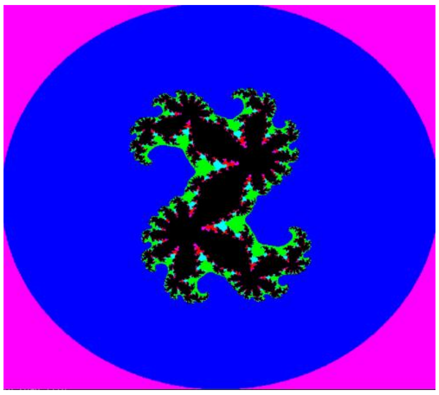
```

#define ROWS 1000
#define COLUMNS 1000
#define iterations 50

void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 1000.0, 0.0, 1000.0);
}

int get_color(int ite_time)
{
    if (ite_time - 1 == 50)
        return 0;

    int temp = (ite_time - 1) % iterations;

    if (temp >= 0 && temp <= 10)
        return 1;
    else if (temp > 10 && temp <= 20)
        return 2;
    else if (temp > 20 && temp <= 30)
        return 3;
    else if (temp > 30 && temp <= 40)
        return 4;
    else
        return 5;
}

void grab()
{
    unsigned char *mpixels = malloc(sizeof(unsigned char) * ROWS * ( COLUMNS - 155 ) * 3);
    glReadBuffer(GL_FRONT);
    glReadPixels(0, 0, ROWS, ( COLUMNS - 155 ), GL_RGB, GL_UNSIGNED_BYTE, mpixels);
    glReadBuffer(GL_BACK);
    for(int i = 0; i < (int)ROWS*( COLUMNS - 155 )*3; i += 3)
    {
        mpixels[i] ^= mpixels[i+2] ^= mpixels[i] ^= mpixels[i+2];
    }
    FIBITMAP* bitmap = FreeImage_Allocate(ROWS, ( COLUMNS - 155 ), 24, 8, 8, 8);

    for(int y = 0 ; y < FreeImage_GetHeight(bitmap); y++)
    {
        BYTE *bits = FreeImage_GetScanLine(bitmap, y);
        for(int x = 0 ; x < FreeImage_GetWidth(bitmap); x++)
        {
            bits[0] = mpixels[(y*COLUMNS+x) * 3 + 0];
            bits[1] = mpixels[(y*COLUMNS+x) * 3 + 1];
            bits[2] = mpixels[(y*COLUMNS+x) * 3 + 2];
            bits += 3;
        }
    }

    FreeImage_Save(FIF_JPEG, bitmap, "output.jpg", JPEG_DEFAULT);

    FreeImage_Unload(bitmap);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    // read the file
    FILE* input = fopen("result.txt", "r");

    printf("Starting to read file...\n");

    int num;
    int color;
    for (int i = 0; i < ROWS; i++)
    {
        for (int j = 0; j < COLUMNS; j++)
        {
            fscanf(input, "%d", &num);

            color = get_color(num);


            if (color == 0)
                glColor3f(0.0, 0.0, 0.0);
            else if (color == 1)
                glColor3f(0.0, 0.0, 1.0);
            else if (color == 2)
                glColor3f(0.0, 1.0, 0.0);
            else if (color == 3)
                glColor3f(0.0, 1.0, 1.0);
            else if (color == 4)
                glColor3f(1.0, 0.0, 0.0);
            else
                glColor3f(1.0, 0.0, 1.0);

            glBegin(GL_POINTS);
                glVertex2i(i, j);
            glEnd();
        }
    }
    fclose(input);

    printf("File reading finished...\n");

    grab();

    glFlush();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(1000, 1000);
    glutCreateWindow("Julia Set View");

    init();
    glutDisplayFunc(display);

    glutMainLoop();


    return 0;
}
```

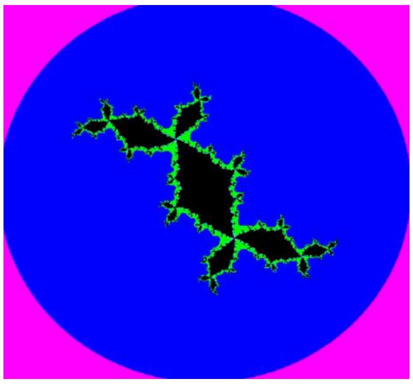
Results


c = 0


c = −1

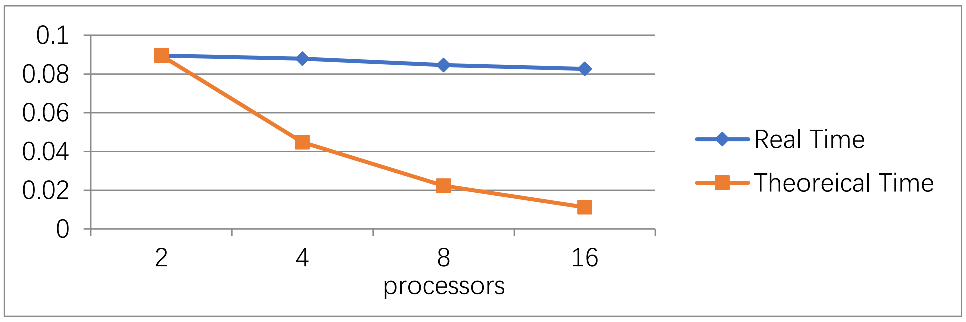

c = 0.3 − 0.4i


c = 0.360284 + 0.100376i,


c = −0.1 + 0.8i,

Benchmark

These values are derived on Nov 26, 2019 (About 15:00pm). Sometimes, the values change according to different time when we run the program. But the trend would not change.



Conclusion:

As the graph shows, the execution time decreases as we add processors. And it's a "perfect" speed improvement theoretically. But in reality, due to the reduction operations, the theoretical and real execution time are different and the difference grows with adding processors