

TERM PROGECT

Group #3



Contents

[Question & Solution 1](#_Toc25616283)

[Source Code 7](#_Toc25616284)

[Results 14](#_Toc25616285)

[Benchmark 15](#_Toc25616286)

# 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.

1. **void** init\_indicesNlengths(**int**\* indices, **int**\* lengths, **int** remainder, **int** sub\_rows, **int** num\_procs)
2. {
3. **for** (**int** i = 0; i < num\_procs; i++)
4. {
5. **if** (i < remainder)
6. lengths[i] = sub\_rows + 1;
7. **else**
8. lengths[i] = sub\_rows;
9. }
10. **for** (**int** i = 0; i < num\_procs; i++)
11. indices[i] = i \* sub\_rows + MIN(i, remainder);
12. }

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

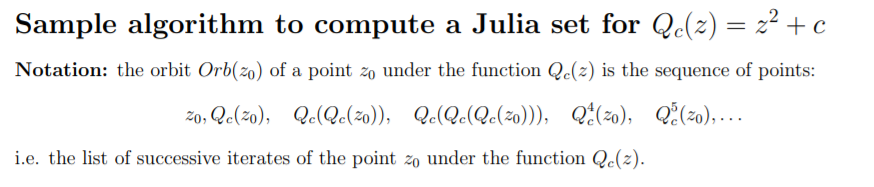
1. MPI\_Init(&argc, &argv);
2. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);
3. MPI\_Comm\_size(MPI\_COMM\_WORLD, &num\_procs);
4. **int** remainder = ROWS % num\_procs;
5. **int** sub\_rows = ROWS / num\_procs;
7. **int**\* indices = malloc(**sizeof**(**int**) \* num\_procs);
8. **int**\* part\_lengths = malloc(**sizeof**(**int**) \* num\_procs);
10. **if** (rank == 0)
11. init\_indicesNlengths(indices, part\_lengths, remainder, sub\_rows, num\_procs);
12. MPI\_Bcast(indices, num\_procs, MPI\_INT, 0, MPI\_COMM\_WORLD);
13. 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



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

1. **for** (**int** i = local\_start; i < local\_end; i++)
2. {
3. **for** (**int** j = 0; j < COLUMNS; j++)
4. {
5. zx = (**float**)(i + 1 - 500) / 250;
6. zy = (**float**)(j + 1 - 500) / 250;
8. color = 0;
10. **while** ((**float**)zx \* zx + zy \* zy <= 4.00 && color <= iterations)
11. {
12. temp = (**float**)zx \* zx - zy \* zy;
13. zy = (**float**)2 \* zx \* zy + cy;
14. zx = (**float**)temp + cx;
15. color++;
16. }
18. local\_colors[counter] = color;
19. counter++;
20. }
21. }

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()

1. MPI\_Gatherv(local\_colors, local\_size, MPI\_INT, colors, color\_part\_lengths, color\_indices, MPI\_INT, 0, MPI\_COMM\_WORLD);
3. **if** (rank == 0)
4. {
5. array\_to\_file(colors);
7. free(colors);
8. free(color\_indices);
9. free(color\_part\_lengths);
10. }

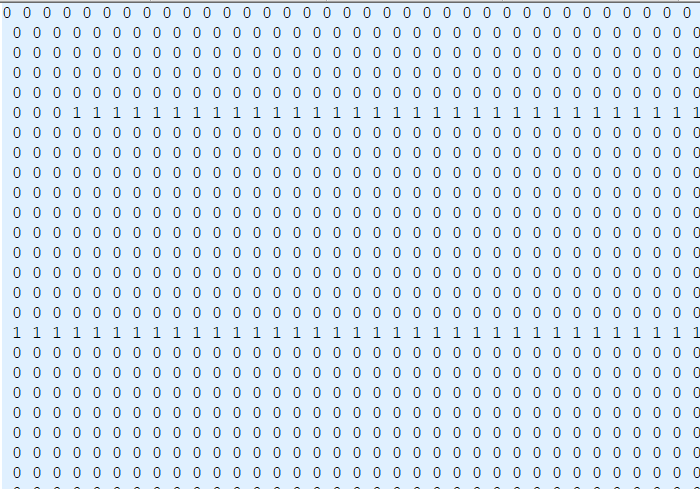
***Question 4:***

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

1. **void** array\_to\_file(**int**\* array)
2. {
3. **FILE**\* out = fopen("result.txt", "w");
5. **int** i = 0;
6. **for** (i = 0; i < ROWS \* COLUMNS; i++) {
7. fprintf(out, "%d ", array[i]);
8. }
10. fclose(out);
11. }



**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.

1. **int** get\_color(**int** ite\_time)
2. {
3. **if** (ite\_time - 1 == 50)
4. **return** 0;
6. **int** temp = (ite\_time - 1) % iterations;
8. **if** (temp >= 0 && temp <= 10)
9. **return** 1;
10. **else** **if** (temp > 10 && temp <= 20)
11. **return** 2;
12. **else** **if** (temp > 20 && temp <= 30)
13. **return** 3;
14. **else** **if** (temp > 30 && temp <= 40)
15. **return** 4;
16. **else**
17. **return** 5;
18. }

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

1. **void** display(**void**)
2. {
3. glClear(GL\_COLOR\_BUFFER\_BIT);
5. // read the file
6. **FILE**\* input = fopen("result.txt", "r");
8. printf("Starting to read file...\n");
10. **int** num;
11. **int** color;
12. **for** (**int** i = 0; i < ROWS; i++)
13. {
14. **for** (**int** j = 0; j < COLUMNS; j++)
15. {
16. fscanf(input, "%d", &num);
18. color = get\_color(num);
20. **if** (color == 0)
21. glColor3f(0.0, 0.0, 0.0);
22. **else** **if** (color == 1)
23. glColor3f(0.0, 0.0, 1.0);
24. **else** **if** (color == 2)
25. glColor3f(0.0, 1.0, 0.0);
26. **else** **if** (color == 3)
27. glColor3f(0.0, 1.0, 1.0);
28. **else** **if** (color == 4)
29. glColor3f(1.0, 0.0, 0.0);
30. **else**
31. glColor3f(1.0, 0.0, 1.0);
33. glBegin(GL\_POINTS);
34. glVertex2i(i, j);
35. glEnd();
36. }
37. }

***Question 6:***

How to output a picture as a .jpg file.

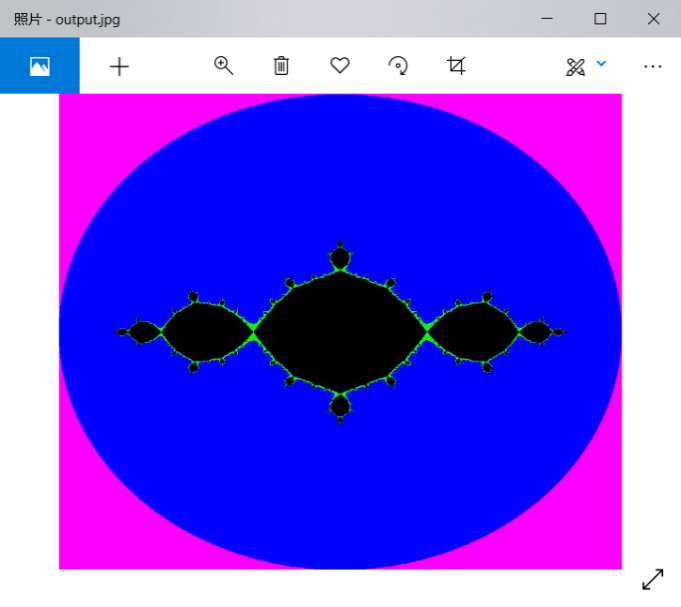
**Solution:**

We used the “FreeImage” library

And create the “grab” function to get the final output .jpg file

1. **void** grab()
2. {
3. unsigned **char** \*mpixels = malloc(**sizeof**(unsigned **char**) \* ROWS \* ( COLUMNS - 155 ) \* 3);
4. glReadBuffer(GL\_FRONT);
5. glReadPixels(0, 0, ROWS, ( COLUMNS - 155 ), GL\_RGB, GL\_UNSIGNED\_BYTE, mpixels);
6. glReadBuffer(GL\_BACK);
7. **for**(**int** i = 0; i < (**int**)ROWS\*( COLUMNS - 155 )\*3; i += 3)
8. {
9. mpixels[i] ^= mpixels[i+2] ^= mpixels[i] ^= mpixels[i+2];
10. }
11. FIBITMAP\* bitmap = FreeImage\_Allocate(ROWS, ( COLUMNS - 155 ), 24, 8, 8, 8);
13. **for**(**int** y = 0 ; y < FreeImage\_GetHeight(bitmap); y++)
14. {
15. **BYTE** \*bits = FreeImage\_GetScanLine(bitmap, y);
16. **for**(**int** x = 0 ; x < FreeImage\_GetWidth(bitmap); x++)
17. {
18. bits[0] = mpixels[(y\*COLUMNS+x) \* 3 + 0];
19. bits[1] = mpixels[(y\*COLUMNS+x) \* 3 + 1];
20. bits[2] = mpixels[(y\*COLUMNS+x) \* 3 + 2];
21. bits += 3;
22. }
23. }
24. FreeImage\_Save(FIF\_JPEG, bitmap, "output.jpg", JPEG\_DEFAULT);
25. FreeImage\_Unload(bitmap);
26. }

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**

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <mpi.h>
4. #include <math.h>
6. #define ROWS 1000
7. #define COLUMNS 1000
8. #define iterations 50
10. #define MIN(a,b) (a < b ? a : b)
12. **void** array\_to\_file(**int**\* array)
13. {
14. **FILE**\* out = fopen("result.txt", "w");
16. **int** i = 0;
17. **for** (i = 0; i < ROWS \* COLUMNS; i++) {
18. fprintf(out, "%d ", array[i]);
19. }
21. fclose(out);
22. }
24. **void** init\_indicesNlengths(**int**\* indices, **int**\* lengths, **int** remainder, **int** sub\_rows, **int** num\_procs)
25. {
26. **for** (**int** i = 0; i < num\_procs; i++)
27. {
28. **if** (i < remainder)
29. lengths[i] = sub\_rows + 1;
30. **else**
31. lengths[i] = sub\_rows;
32. }
34. **for** (**int** i = 0; i < num\_procs; i++)
35. indices[i] = i \* sub\_rows + MIN(i, remainder);
36. }
38. **int** main(**int** argc, **char**\*\* argv) {
39. **int** num\_procs;
40. **int** rank;
42. **float** cx = atof(argv[1]);
43. **float** cy = atof(argv[2]);
45. MPI\_Init(&argc, &argv);
46. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);
47. MPI\_Comm\_size(MPI\_COMM\_WORLD, &num\_procs);
48. MPI\_Barrier(MPI\_COMM\_WORLD);
49. **double** elapsed\_time = -MPI\_Wtime();
51. **int** remainder = ROWS % num\_procs;
52. **int** sub\_rows = ROWS / num\_procs;
54. **int**\* indices = malloc(**sizeof**(**int**) \* num\_procs);
55. **int**\* part\_lengths = malloc(**sizeof**(**int**) \* num\_procs);
57. **if** (rank == 0)
58. init\_indicesNlengths(indices, part\_lengths, remainder, sub\_rows, num\_procs);
60. MPI\_Bcast(indices, num\_procs, MPI\_INT, 0, MPI\_COMM\_WORLD);
61. MPI\_Bcast(part\_lengths, num\_procs, MPI\_INT, 0, MPI\_COMM\_WORLD);
63. **int** local\_row = part\_lengths[rank];
64. **int** local\_size = local\_row \* COLUMNS;
65. **int**\* local\_colors = malloc(**sizeof**(**int**) \* local\_size);

68. **int** local\_start = indices[rank];
69. **int** local\_end = local\_start + local\_row;
71. **float** zx, zy;
72. **float** temp;
74. **int** color;
75. **int** counter = 0;
76. **for** (**int** i = local\_start; i < local\_end; i++)
77. {
78. **for** (**int** j = 0; j < COLUMNS; j++)
79. {
80. zx = (**float**)(i + 1 - ROWS / 2) / (ROWS / 4);
81. zy = (**float**)(j + 1 - COLUMNS / 2) / (COLUMNS / 4);
83. color = 0;
85. **while** ((**float**)zx \* zx + zy \* zy <= 4.00 && color <= iterations)
86. {
87. temp = (**float**)zx \* zx - zy \* zy;
88. zy = (**float**)2 \* zx \* zy + cy;
89. zx = (**float**)temp + cx;
90. color++;
91. }
93. local\_colors[counter] = color;
94. counter++;
95. }
96. }

99. **int**\* colors;
100. **int**\* color\_indices;
101. **int**\* color\_part\_lengths;
102. **if** (rank == 0)
103. {
104. colors = malloc(**sizeof**(**int**) \* ROWS \* COLUMNS);
105. color\_indices = malloc(**sizeof**(**int**) \* num\_procs);
106. color\_part\_lengths = malloc(**sizeof**(**int**) \* num\_procs);
108. **for** (**int** i = 0; i < num\_procs; i++)
109. {
110. color\_part\_lengths[i] = part\_lengths[i] \* COLUMNS;
111. color\_indices[i] = indices[i] \* COLUMNS;
112. }
113. }
115. MPI\_Gatherv(local\_colors, local\_size, MPI\_INT, colors, color\_part\_lengths, color\_indice s, MPI\_INT, 0, MPI\_COMM\_WORLD);
117. **if** (rank == 0)
118. {
120. array\_to\_file(colors);
122. free(colors);
123. free(color\_indices);
124. free(color\_part\_lengths);
125. }
126. free(indices);
127. free(part\_lengths);
128. free(local\_colors);
130. printf("\nProcessor %d done!\n", rank);
132. // get the elapsed time for benchmarking
133. elapsed\_time += MPI\_Wtime();
134. printf("The elapsed time is: %lf\n", elapsed\_time);
136. MPI\_Finalize();
138. **return** 0;
139. }

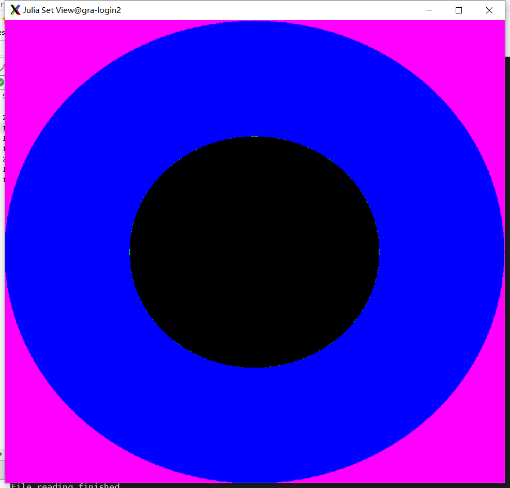
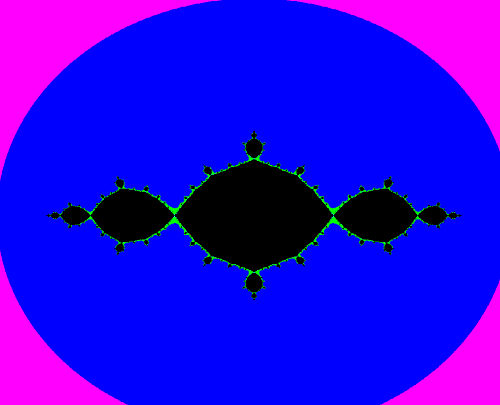
**points.c**

1. #include <GL/glut.h>
2. #include <stdlib.h>
3. #include <stdio.h>
4. #include <string.h>
5. #include <stddef.h>
6. #include <FreeImage.h>
8. #define ROWS 1000
9. #define COLUMNS 1000
10. #define iterations 50
12. **void** init(**void**)
13. {
14. glClearColor(0.0, 0.0, 0.0, 1.0);
15. glMatrixMode(GL\_PROJECTION);
16. gluOrtho2D(0.0, 1000.0, 0.0, 1000.0);
17. }
19. **int** get\_color(**int** ite\_time)
20. {
21. **if** (ite\_time - 1 == 50)
22. **return** 0;
24. **int** temp = (ite\_time - 1) % iterations;
26. **if** (temp >= 0 && temp <= 10)
27. **return** 1;
28. **else** **if** (temp > 10 && temp <= 20)
29. **return** 2;
30. **else** **if** (temp > 20 && temp <= 30)
31. **return** 3;
32. **else** **if** (temp > 30 && temp <= 40)
33. **return** 4;
34. **else**
35. **return** 5;
36. }
38. **void** grab()
39. {
40. unsigned **char** \*mpixels = malloc(**sizeof**(unsigned **char**) \* ROWS \* ( COLUMNS - 155 ) \* 3);
41. glReadBuffer(GL\_FRONT);
42. glReadPixels(0, 0, ROWS, ( COLUMNS - 155 ), GL\_RGB, GL\_UNSIGNED\_BYTE, mpixels);
43. glReadBuffer(GL\_BACK);
44. **for**(**int** i = 0; i < (**int**)ROWS\*( COLUMNS - 155 )\*3; i += 3)
45. {
46. mpixels[i] ^= mpixels[i+2] ^= mpixels[i] ^= mpixels[i+2];
47. }
48. FIBITMAP\* bitmap = FreeImage\_Allocate(ROWS, ( COLUMNS - 155 ), 24, 8, 8, 8);
50. **for**(**int** y = 0 ; y < FreeImage\_GetHeight(bitmap); y++)
51. {
52. **BYTE** \*bits = FreeImage\_GetScanLine(bitmap, y);
53. **for**(**int** x = 0 ; x < FreeImage\_GetWidth(bitmap); x++)
54. {
55. bits[0] = mpixels[(y\*COLUMNS+x) \* 3 + 0];
56. bits[1] = mpixels[(y\*COLUMNS+x) \* 3 + 1];
57. bits[2] = mpixels[(y\*COLUMNS+x) \* 3 + 2];
58. bits += 3;
59. }
60. }
62. FreeImage\_Save(FIF\_JPEG, bitmap, "output.jpg", JPEG\_DEFAULT);
64. FreeImage\_Unload(bitmap);
65. }
67. **void** display(**void**)
68. {
69. glClear(GL\_COLOR\_BUFFER\_BIT);
71. // read the file
72. **FILE**\* input = fopen("result.txt", "r");
74. printf("Starting to read file...\n");
76. **int** num;
77. **int** color;
78. **for** (**int** i = 0; i < ROWS; i++)
79. {
80. **for** (**int** j = 0; j < COLUMNS; j++)
81. {
82. fscanf(input, "%d", &num);
84. color = get\_color(num);

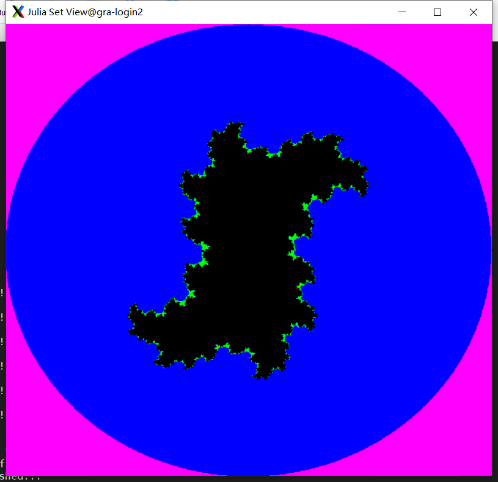
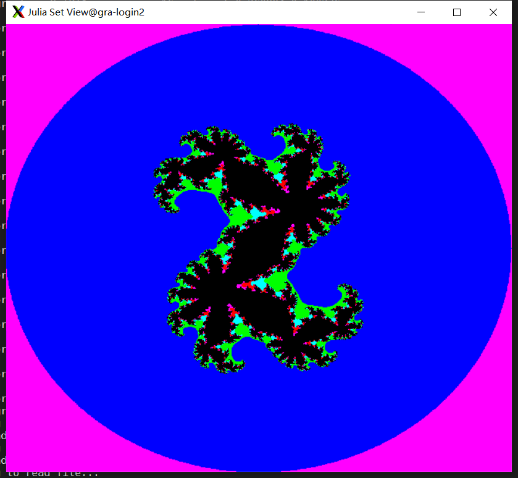
87. **if** (color == 0)
88. glColor3f(0.0, 0.0, 0.0);
89. **else** **if** (color == 1)
90. glColor3f(0.0, 0.0, 1.0);
91. **else** **if** (color == 2)
92. glColor3f(0.0, 1.0, 0.0);
93. **else** **if** (color == 3)
94. glColor3f(0.0, 1.0, 1.0);
95. **else** **if** (color == 4)
96. glColor3f(1.0, 0.0, 0.0);
97. **else**
98. glColor3f(1.0, 0.0, 1.0);
100. glBegin(GL\_POINTS);
101. glVertex2i(i, j);
102. glEnd();
103. }
104. }
105. fclose(input);
107. printf("File reading finished...\n");
109. grab();
111. glFlush();
112. }
113. **int** main(**int** argc, **char** \*argv[])
114. {
115. glutInit(&argc, argv);
116. glutInitDisplayMode(GLUT\_RGB | GLUT\_SINGLE);
117. glutInitWindowPosition(0, 0);
118. glutInitWindowSize(1000, 1000);
119. glutCreateWindow("Julia Set View");
121. init();
122. glutDisplayFunc(display);
124. glutMainLoop();

127. **return** 0;
128. }

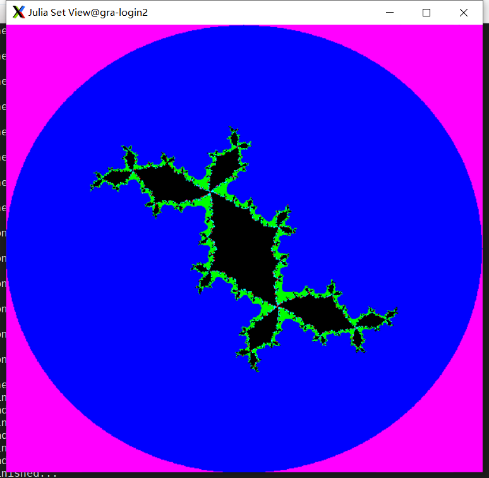
# Results

**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