

Coursework 2 for M522 Numerical Analysis (Parallel Computing)

Pavel Buklemishev

February 23, 2025

- 1 Use just two MPI processors, divide the computational domain into 2, one MPI processor will work on the top half and the other works on the bottom half. You may either output the results (computed colours for the Mandelbrot set at each display pixel) into files or display the results using the GUI provided. Demonstrate the execution time against a serial execution**

United with the second problem.

Elapsed time = 7.388339e+00 seconds - serial time execution

- 2 Take the number the processors as a variable, and divide the computational domain accordingly, into sections of rows, and get each processor to do the work, output the results either using files or GUI, and demonstrate the execution time**

```

1   int processor_id;
2   int num_of_proc;
3   MPI_Comm_rank(MPI_COMM_WORLD , &processor_id);
4   MPI_Comm_size(MPI_COMM_WORLD , &num_of_proc);
5   int initial_position;
6   int final_position;
7
8 if (processor_id == 0) {
9
10    setup_return = setup(width, height, &windata.display
11                          , &windata.win,
12                          &windata.gc, &min_color, &
13                          max_color);
14
15    MPI_Bcast(&windata.win, 1, MPI_UNSIGNED_LONG, 0,
16               MPI_COMM_WORLD);
17    MPI_Bcast(&min_color, 1, MPI_UNSIGNED_LONG, 0,
18               MPI_COMM_WORLD);
19    MPI_Bcast(&max_color, 1, MPI_UNSIGNED_LONG, 0,
20               MPI_COMM_WORLD);
21
22
23    initial_position = 0;
24    final_position =(height / num_of_proc);
25 } else {
26
27    MPI_Bcast(&windata.win, 1, MPI_UNSIGNED_LONG, 0,
28               MPI_COMM_WORLD);
29    MPI_Bcast(&min_color, 1, MPI_UNSIGNED_LONG, 0,
30               MPI_COMM_WORLD);
31    MPI_Bcast(&max_color, 1, MPI_UNSIGNED_LONG, 0,
32               MPI_COMM_WORLD);
33    windata.display = XOpenDisplay(NULL);
34
35
36    windata.gc = XCreateGC(windata.display, windata.win,
37                           0, NULL);
38
39 if(processor_id==num_of_proc-1){
40     initial_position = (height / num_of_proc)*
41                         processor_id;

```

```

33         final_position = height;
34
35     }
36     else{
37         initial_position = (height / num_of_proc)*
38             processor_id;
39         final_position = (height / num_of_proc)*(
40             processor_id+1);
41     }
42
43
44     scale_real = (double) (real_max - real_min) / (double)
45         width;
46     scale_imag = (double) (imag_max - imag_min) / (double)
47         height;
48
49     scale_color = (double) (max_color - min_color) / (
50         double) (maxiter - 1);
51
52     ilines = calloc( width*height , sizeof(uint) );
53
54     for ( row=initial_position; row<final_position; ++row
55         ) {
56
57         c.imag = imag_min + ((double) (height-1-row) *
58             scale_imag);
59
60         for ( col=0; col<width; ++col ) {
61             c.real = real_min + ((double) col * scale_real);
62
63             z.real = z.imag = 0.0;
64             count = 0;
65             do {
66                 temp = z.real*z.real - z.imag*z.imag + c.real;
67                 z.imag = 2.0*z.real*z.imag + c.imag;
68                 z.real = temp;
69
70                 lengthsq = z.real*z.real + z.imag*z.imag;
71                 ++count;
72             } while ( lengthsq<(N*N) && count<maxiter );
73
74
75     }
76
77 }

```

```
69     color = (ulong) ((count-1) * scale_color) +
70         min_color;
71     ilines[width*row+col] = color;
72 }
73
74 drawLine( row, width, ilines, &windata );
75
76 }
77
78 XFlush(windata.display);
79 MPI_Barrier(MPI_COMM_WORLD);
```

2.1 n of processors = 2

Elapsed time = 3.643258e+00 seconds

2.2 n of processors = 4

Elapsed time = 3.494976e+00 seconds

2.3 n of processors = 8

Elapsed time = 3.288727e+00 seconds

2.4 n of processors = 32

Elapsed time = 3.018115e+00 seconds

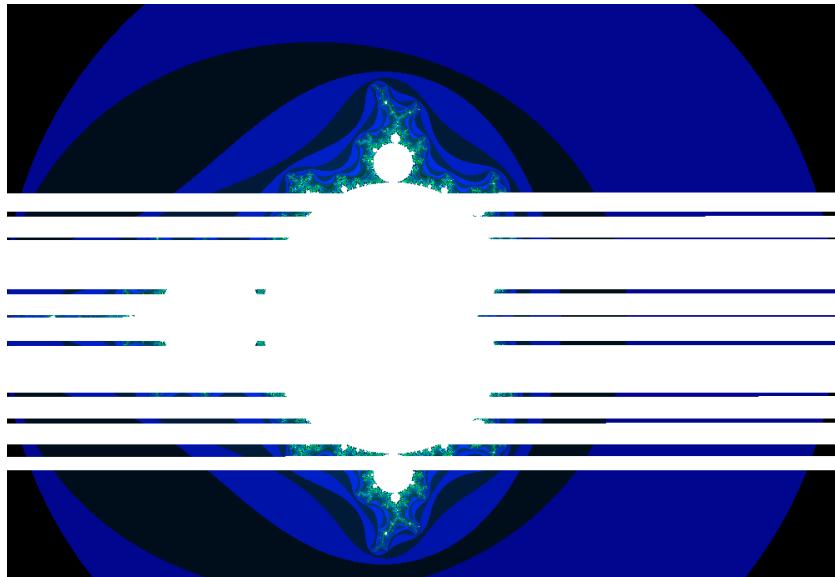


Figure 1: Variable processor drawing architecture

3 Consider a master-workers architecture, the master processor will be dynamically allocating the workloads (i.e. small sections of rows) to workers that are available, demonstrate the execution time.

```
1 int processor_id;
2 int num_of_proc;
3 MPI_Comm_rank(MPI_COMM_WORLD, &processor_id);
4 MPI_Comm_size(MPI_COMM_WORLD, &num_of_proc);
5 int batch_size = 5;
6 if (processor_id == 0) {
7     setup_return = setup(width, height, &windata.display
8                           , &windata.win,
9                           &windata.gc, &min_color, &
10                             max_color);
11     MPI_Bcast(&min_color, 1, MPI_UNSIGNED_LONG, 0,
12               MPI_COMM_WORLD);
13     MPI_Bcast(&max_color, 1, MPI_UNSIGNED_LONG, 0,
14               MPI_COMM_WORLD);
```

```

11 scale_color = (double)(max_color - min_color) / (
12   double)maxiter;
13
14 int lines_in_work = 0;
15 int active_workers = num_of_proc - 1;
16 for (int i = 1; i < num_of_proc; i++) {
17   if (lines_in_work >= height) {
18     int stop_signal = -1;
19     MPI_Send(&stop_signal, 1, MPI_INT, i, 0,
20             MPI_COMM_WORLD);
21     continue;
22   }
23   int real_batch_size = min(batch_size, height -
24   lines_in_work);
25   MPI_Send(&lines_in_work, 1, MPI_INT, i, 0,
26             MPI_COMM_WORLD);
27   MPI_Send(&real_batch_size, 1, MPI_INT, i, 1,
28             MPI_COMM_WORLD);
29   lines_in_work += real_batch_size;
30 }
31
32 while (active_workers > 0) {
33   MPI_Status status;
34   int row_start, received_batch_size;
35
36   MPI_Recv(&row_start, 1, MPI_INT, MPI_ANY_SOURCE,
37             0, MPI_COMM_WORLD, &status);
38
39   MPI_Recv(&received_batch_size, 1, MPI_INT,
40             status.MPI_SOURCE, 1, MPI_COMM_WORLD,
41             MPI_STATUS_IGNORE);
42
43   ulong *ilines = malloc(received_batch_size *
44   width * sizeof(ulong));
45
46   MPI_Recv(ilines, received_batch_size * width,
47             MPI_UNSIGNED_LONG,
48             status.MPI_SOURCE, 2, MPI_COMM_WORLD,
49             MPI_STATUS_IGNORE);
50
51   for (int r = 0; r < received_batch_size; r++) {
52     int actual_row = row_start + r;
53
54     if (actual_row > height)
55       break;
56
57     for (int c = 0; c < width; c++) {
58       if (ilines[r * width + c] >= height)
59         break;
60
61       if (ilines[r * width + c] >= scale_color)
62         color = 1;
63       else
64         color = 0;
65
66       if (color != current_color) {
67         current_color = color;
68         MPI_Put(&current_color, 1, MPI_CHAR, i,
69                 MPI_COMM_WORLD, MPI_PUT_AS_IS);
70       }
71     }
72   }
73
74   MPI_Wait(&status, MPI_STATUS_IGNORE);
75
76   if (status.MPI_SOURCE == MPI_SOURCE_STOPPED)
77     break;
78
79   MPI_Status_clear(&status);
80 }
81
82 MPI_Finalize();
83
84 #endif

```

```

43         drawLine(actual_row, width, &ilines[r *  

44             width], &windata);  

45     }  

46     XFlush(windata.display);  

47     free(ilines);  

48  

49     if (lines_in_work < height) {  

50         int real_batch_size = min(batch_size, height  

51             - lines_in_work);  

52         MPI_Send(&lines_in_work, 1, MPI_INT,  

53             status.MPI_SOURCE, 0, MPI_COMM_WORLD);  

54         MPI_Send(&real_batch_size, 1, MPI_INT,  

55             status.MPI_SOURCE, 1, MPI_COMM_WORLD);  

56         lines_in_work += real_batch_size;  

57     } else {  

58         int stop_signal = -1;  

59         MPI_Send(&stop_signal, 1, MPI_INT, status.  

60             MPI_SOURCE, 0, MPI_COMM_WORLD);  

61         active_workers--;  

62     }
63 }
64
65
66
67
68
69
70
71
72
73
74
75

```

```

76
77     MPI_Recv(&row_i, 1, MPI_INT, 0, 0,
78               MPI_COMM_WORLD, MPI_STATUS_IGNORE);
79     if (row_i == -1) {
80         break;
81     }
82     MPI_Recv(&received_batch_size, 1, MPI_INT, 0, 1,
83               MPI_COMM_WORLD, MPI_STATUS_IGNORE);
84
85
86     ulong *ilines = malloc(received_batch_size *
87                             width * sizeof(ulong));
88
89
90     for (int r = 0; r < received_batch_size; r++) {
91         int row = row_i + r;
92
93         c.imag = imag_min + ((double)(height - 1 -
94                                     row) * scale_imag);
95
96         for (int col = 0; col < width; ++col) {
97             c.real = real_min + ((double)col *
98                                   scale_real);
99
100            z.real = z.imag = 0.0;
101            count = 0;
102            do {
103                temp = z.real * z.real - z.imag * z.
104                      imag + c.real;
105                z.imag = 2.0 * z.real * z.imag + c.
106                      imag;
107                z.real = temp;
108
109                lengthsq = z.real * z.real + z.imag
110                          * z.imag;
111                ++count;
112
113            } while (lengthsq < (N * N) && count <
114                     maxiter);
115
116
117
118
119

```

```

110          color = (ulong)((count - 1) *
111              scale_color) + min_color;
112          ilines[r * width + col] = color;
113      }
114
115      MPI_Send(&row_i, 1, MPI_INT, 0, 0,
116                  MPI_COMM_WORLD);
116      MPI_Send(&received_batch_size, 1, MPI_INT, 0, 1,
117                  MPI_COMM_WORLD);
118
119      MPI_Send(ilines, width * received_batch_size,
120                  MPI_UNSIGNED_LONG, 0, 2, MPI_COMM_WORLD);
121
122      free(ilines);
121  }
122 }
```

set batch size = 5

3.1 n of processors = 2(1 workers)

Elapsed time = 6.235297e+00 seconds

3.2 n of processors = 3(2 workers)

Elapsed time = 3.175187e+00 seconds

3.3 n of processors = 4(3 workers)

Elapsed time = 2.258805e+00 seconds

3.4 n of processors = 5(4 workers)

Elapsed time = 1.878168e+00 seconds

3.5 n of processors = 8 (7 workers)

Elapsed time = 1.454936e+00 seconds

3.6 n of processors = 9 (8 workers)

Elapsed time = 1.302795e+00 seconds

3.7 n of processors = 32 (31 workers)

Elapsed time = 1.121481e+00 seconds

3.8 n of processors = 33 (32 workers)

Elapsed time = 1.135950e+00 seconds



Figure 2: Mandelbrot master-worker architecture