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

```
  int processor_id;
  int num_of_proc;
  MPI_Comm_rank(MPI_COMM_WORLD, &processor_id);
  MPI_Comm_size(MPI_COMM_WORLD, &num_of_proc);
  int initial_position;
  int final_position;

if (processor_id == 0) {

    setup_return = setup(width, height, &windata.display, &windata.win,
                         &windata.gc, &min_color, &max_color);


    MPI_Bcast(&windata.win, 1, MPI_UNSIGNED_LONG, 0, MPI_COMM_WORLD);
    MPI_Bcast(&min_color, 1, MPI_UNSIGNED_LONG, 0, MPI_COMM_WORLD);
    MPI_Bcast(&max_color, 1, MPI_UNSIGNED_LONG, 0, MPI_COMM_WORLD);


    initial_position = 0;
    final_position =(height / num_of_proc);
} else {

    MPI_Bcast(&windata.win, 1, MPI_UNSIGNED_LONG, 0, MPI_COMM_WORLD);
    MPI_Bcast(&min_color, 1, MPI_UNSIGNED_LONG, 0, MPI_COMM_WORLD);
    MPI_Bcast(&max_color, 1, MPI_UNSIGNED_LONG, 0, MPI_COMM_WORLD);
    windata.display = XOpenDisplay(NULL);


    windata.gc = XCreateGC(windata.display, windata.win, 0, NULL);

    if(processor_id==num_of_proc-1){
       initial_position = (height / num_of_proc)*processor_id;
```

```
       final_position = height;

    }
    else{
        initial_position = (height / num_of_proc)*processor_id;
        final_position = (height / num_of_proc)*(processor_id+1);
    }

}


  scale_real = (double) (real_max - real_min) / (double) width;
  scale_imag = (double) (imag_max - imag_min) / (double) height;

  scale_color = (double) (max_color - min_color) / (double) (maxiter - 1);

  ilines = calloc( width*height, sizeof(uint) );

  for ( row=initial_position; row<final_position; ++row ) {

    c.imag = imag_min + ((double) (height-1-row) * scale_imag);

    for ( col=0; col<width; ++col ) {
      c.real = real_min + ((double) col * scale_real);

      z.real = z.imag = 0.0;
      count = 0;
      do {
        temp   = z.real*z.real - z.imag*z.imag + c.real;
        z.imag = 2.0*z.real*z.imag + c.imag;
        z.real = temp;

        lengthsq = z.real*z.real + z.imag*z.imag;
        ++count;
      } while ( lengthsq<(N*N) && count<maxiter );

```

```
      color = (ulong) ((count-1) * scale_color) +
         min_color;

      ilines[width*row+col] = color;
    }


    drawLine( row, width, ilines, &windata );

  }

    XFlush(windata.display);
    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

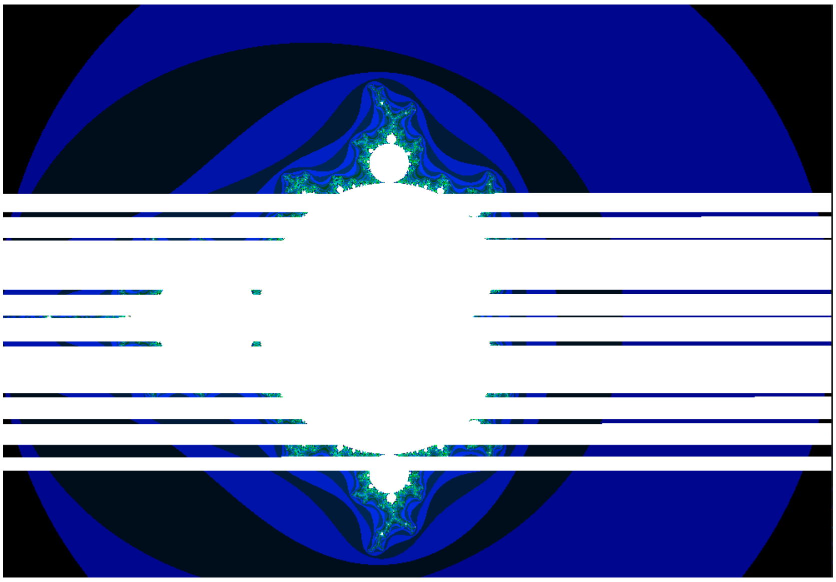

Figure 1: Variable processor drawing architecture

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.

```
 int processor_id;
int num_of_proc;
MPI_Comm_rank(MPI_COMM_WORLD, &processor_id);
MPI_Comm_size(MPI_COMM_WORLD, &num_of_proc);
int batch_size = 5;
if (processor_id == 0) {
    setup_return = setup(width, height, &windata.display
       , &windata.win,
                                &windata.gc, &min_color, &
                                   max_color);
    MPI_Bcast(&min_color, 1, MPI_UNSIGNED_LONG, 0,
       MPI_COMM_WORLD);
    MPI_Bcast(&max_color, 1, MPI_UNSIGNED_LONG, 0,
       MPI_COMM_WORLD);
```

```
    scale_color = (double)(max_color - min_color) / (double)maxiter;

    int lines_in_work = 0;
    int active_workers = num_of_proc - 1;
    for (int i = 1; i < num_of_proc; i++) {
        if (lines_in_work >= height) {
            int stop_signal = -1;
            MPI_Send(&stop_signal, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
            continue;
        }
        int real_batch_size = min(batch_size, height - lines_in_work);
        MPI_Send(&lines_in_work,   1, MPI_INT, i, 0, MPI_COMM_WORLD);
        MPI_Send(&real_batch_size, 1, MPI_INT, i, 1, MPI_COMM_WORLD);
        lines_in_work += real_batch_size;
    }

    while (active_workers > 0) {
        MPI_Status status;
        int row_start, received_batch_size;

        MPI_Recv(&row_start, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status);

        MPI_Recv(&received_batch_size, 1, MPI_INT, status.MPI_SOURCE, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

        ulong *ilines = malloc(received_batch_size * width * sizeof(ulong));

        MPI_Recv(ilines, received_batch_size * width, MPI_UNSIGNED_LONG,
                 status.MPI_SOURCE, 2, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

        for (int r = 0; r < received_batch_size; r++) {
            int actual_row = row_start + r;

```

```
            drawLine(actual_row, width, &ilines[r * width], &windata);
        }
        XFlush(windata.display);
        free(ilines);

        if (lines_in_work < height) {
            int real_batch_size = min(batch_size, height - lines_in_work);
            MPI_Send(&lines_in_work,   1, MPI_INT, status.MPI_SOURCE, 0, MPI_COMM_WORLD);
            MPI_Send(&real_batch_size, 1, MPI_INT, status.MPI_SOURCE, 1, MPI_COMM_WORLD);
            lines_in_work += real_batch_size;
        } else {
            int stop_signal = -1;
            MPI_Send(&stop_signal, 1, MPI_INT, status.MPI_SOURCE, 0, MPI_COMM_WORLD);
            active_workers--;
        }
    }



}
else {
    MPI_Bcast(&min_color, 1, MPI_UNSIGNED_LONG, 0, MPI_COMM_WORLD);
    MPI_Bcast(&max_color, 1, MPI_UNSIGNED_LONG, 0, MPI_COMM_WORLD);

    scale_real = (double) (real_max - real_min) / (double) width;
    scale_imag = (double) (imag_max - imag_min) / (double) height;

    scale_color = (double) (max_color - min_color) / (double) (maxiter - 1);

    while (1) {
        int row_i;
        int received_batch_size;

```

```

        MPI_Recv(&row_i, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        if (row_i == -1) {
            break;
        }
        MPI_Recv(&received_batch_size, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

        ulong *ilines = malloc(received_batch_size * width * sizeof(ulong));


        for (int r = 0; r < received_batch_size; r++) {
            int row = row_i + r;

            c.imag = imag_min + ((double)(height - 1 - row) * scale_imag);

            for (int col = 0; col < width; ++col) {
                c.real = real_min + ((double)col * scale_real);

                z.real = z.imag = 0.0;
                count = 0;
                do {
                    temp = z.real * z.real - z.imag * z.imag + c.real;
                    z.imag = 2.0 * z.real * z.imag + c.imag;
                    z.real = temp;



                    lengthsq = z.real * z.real + z.imag * z.imag;
                    ++count;

                } while (lengthsq < (N * N) && count < maxiter);



```

```
                    color = (ulong)((count - 1) * scale_color) + min_color;
                    ilines[r * width + col] = color;
                }
            }

            MPI_Send(&row_i, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
            MPI_Send(&received_batch_size, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);

            MPI_Send(ilines, width * received_batch_size, MPI_UNSIGNED_LONG, 0, 2, MPI_COMM_WORLD);

            free(ilines);
        }
}
```

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

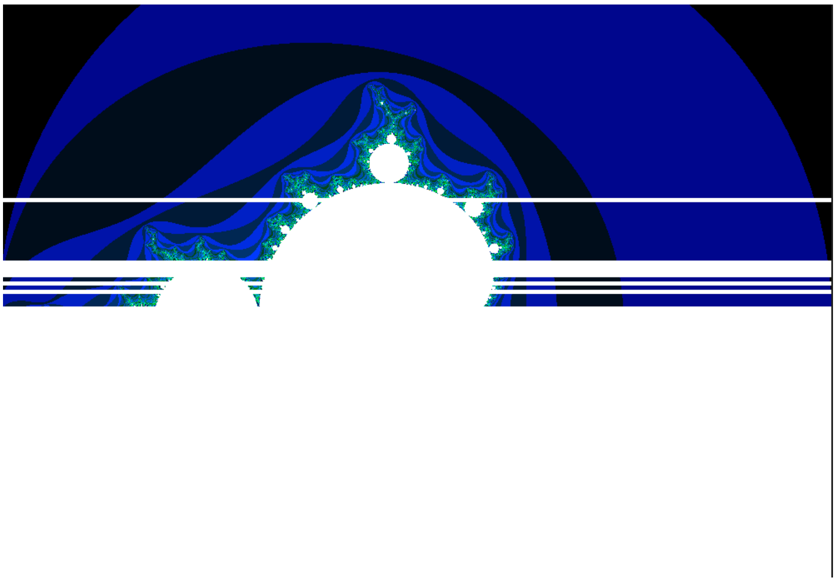

Figure 2: Mandelbrot master-worker architecture