CSCI 455: Lab #1 — Mandelbrot Example

Darwin Jacob Groskleg

Winter 2020

Contents

Part I: Parallel Compute Mandelbrot Set

Program Outputs and Results

```
ubuntu@ip-172-31-43-252:~/csci455/Lab1-Mandelbrot$ make run1
mpicxx -std=c++14 -Wpedantic -Wall -Wno-missing-braces -Wextra -g -D_GLIBCXX_DEBUG -O0    mandelbrot_parallel.cpp  -lm
 -o mandelbrot_parallel
Platform: Linux (96 cpu cores recognized)
mpirun --use-hwthread-cpus -np 96 ./mandelbrot_parallel image.ppm
Time elapsed during calculation: 0.920786 secs.
Time elapsed total: 1.71778 secs
ubuntu@ip-172-31-43-252:~/csci455/Lab1-Mandelbrot$
```

Figure 1: Single run of the parallelized Mandelbrot set filling program, executed over 96 compute nodes.

```
ubuntu@ip-172-31-43-252:~/csci455/Lab1-Mandelbrot$ make sample_mandelbrot
Platform: Linux (96 cpu cores recognized)
for nodeprocs in 2 4 8 12 16 24 ; do \
        echo "\nMPIRUN mandelbrot_parallel with $nodeprocs node processes:" ; \
        mpirun --use-hwthread-cpus -np $nodeprocs  ./mandelbrot_parallel ./image-$nodeprocs.ppm ; \
done

MPIRUN mandelbrot_parallel with 2 node processes:
Time elapsed during calculation: 73.5165 secs.
Time elapsed total: 74.1561 secs

MPIRUN mandelbrot_parallel with 4 node processes:
Time elapsed during calculation: 22.448 secs.
Time elapsed total: 23.0629 secs

MPIRUN mandelbrot_parallel with 8 node processes:
Time elapsed during calculation: 10.07 secs.
Time elapsed total: 10.7105 secs

MPIRUN mandelbrot_parallel with 12 node processes:
Time elapsed during calculation: 6.53995 secs.
Time elapsed total: 7.19503 secs

MPIRUN mandelbrot_parallel with 16 node processes:
Time elapsed during calculation: 4.8003 secs.
Time elapsed total: 5.45317 secs

MPIRUN mandelbrot_parallel with 24 node processes:
Time elapsed during calculation: 3.139 secs.
Time elapsed total: 3.79641 secs
ubuntu@ip-172-31-43-252:~/csci455/Lab1-Mandelbrot$
```

Figure 2: Many runs (2, 4, 8, 12, 16, 24 nodes) of the parallelized mandelbrot program on a 96 core host.

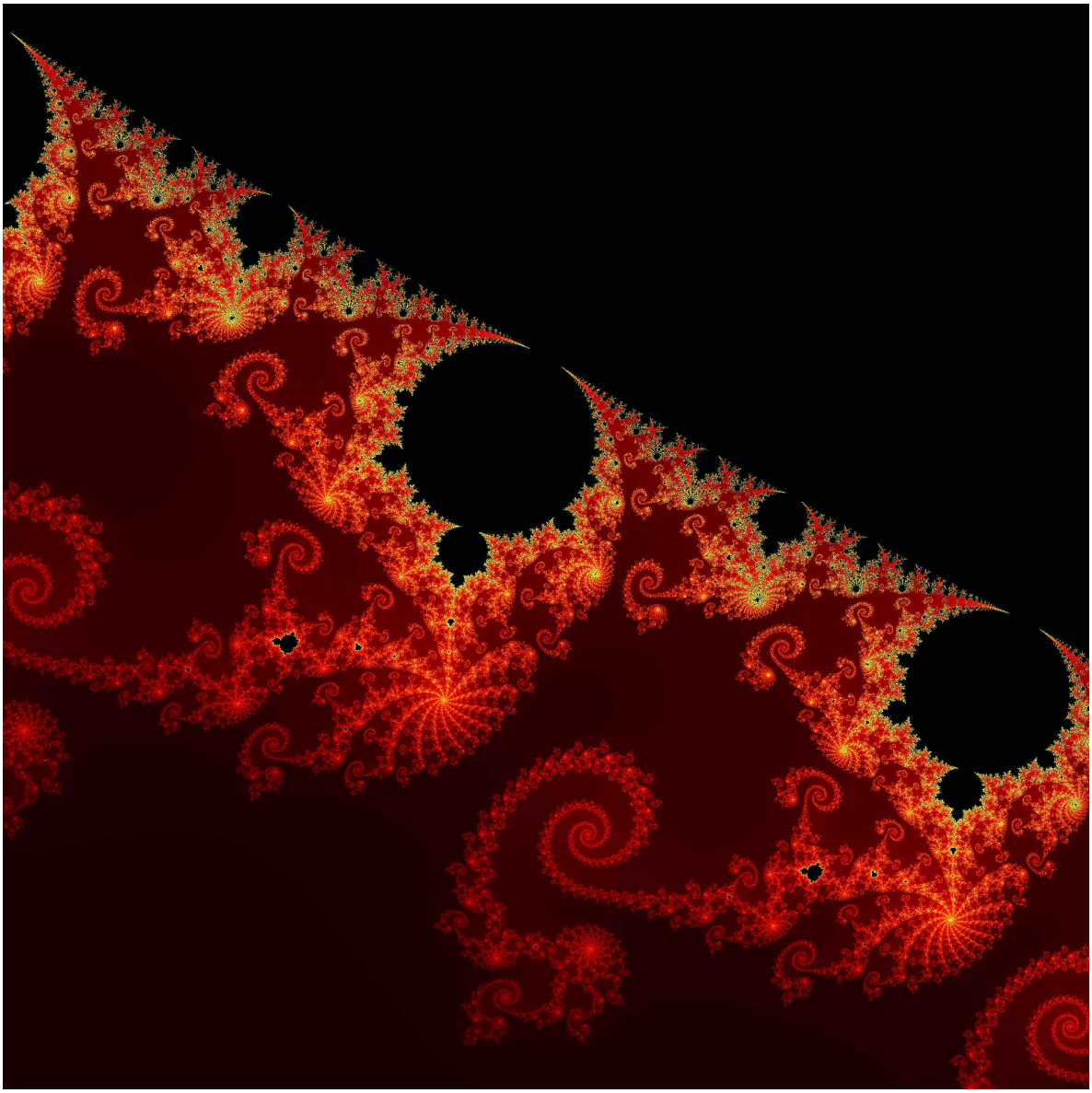

Figure 3: Generated mandelbrot set image (lo-res)

mandelbrot_parallel.cpp

```cpp
/* mandelbrot_parallel.cpp
 * ----------------------
 * Authors: Darwin Jacob Groskleg, Laurence T. Yang
 * CSCI 455 Lab 1
 *
 * Purpose:  Compute and draw the pixels of an image of the Mandelbrot Set,
 *           using MPI parallelize the work.
 *
 * Question: Rerun the computation using different numbers of processors
 *           (2, 4, 8, 12, 16, 24) and  list the results in a table.
 */
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>

#include <math.h>
#include <time.h>
#include <stdlib.h>

#include <mpi.h>

using namespace std;

const int imgX=3000;             // horizontal image resolution
const int imgY=3000;             // vertical image resolution
const int iter_n=3000;           // max. iteration number
const double yMin= -0.135;       // Mandelbrot scene y - range
const double yMax= -0.115;
const double xMin= -0.79;        // Mandelbrot scene x -range
const double xMax= -0.77;
int img_array[imgX][imgY] = {0}; // our MAndelbrot set values array
int img_line[imgY] = {0};

int converges (double cx, double cy);
void usage(std::string program);

int main(int argc, char **argv)
{
    //variables for MPI communication:
    int id, nproc;
    MPI_Status status;
    int answer[2];
    double question;

    double resX=0;  // Resolution of our iteration steps
    double resY=0;  // this will be calculated by (yMax-yMin) / imgY later..
    // calculation will start at this point and we will change this dynamically
    double cx=xMin;
    double cy=yMin;
    double s_time, e_time; // we will show some timing data

    // Initialize MPI:
    MPI_Init(&argc, &argv);
    // Get my rank:
```

```
56      MPI_Comm_rank(MPI_COMM_WORLD, &id);
57      // Get the total number of processors:
58      MPI_Comm_size(MPI_COMM_WORLD, &nproc);
59
60      MPI_Barrier(MPI_COMM_WORLD); // for precise timing
61
62
63      // Master
64      if (id == 0) {
65          if (argc<2)
66              usage(argv[0]);
67
68          ofstream myfile;  // we will write to this file
69          string filename1(argv[1]);
70          // filename1 = "mandel.ppm";
71          char *fileName1 = (char*)filename1.c_str();
72
73          //prepare the step resolution
74          resX = (xMax-xMin) / imgX;
75          resY = (yMax-yMin) / imgY;
76
77          s_time = MPI_Wtime(); // we get a time value at this point
78
79          // we do the calculation for every point of our complex plane,
80          // thus on our 2D image with appropriate steps
81          for (int i=0; i<imgX; i++) {
82              MPI_Recv(answer, 2, MPI_INT, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD,
83                      &status);
84              // answer[0] -- from whom
85              // answer[1] -- '-1' or the X coordinate
86
87              if(answer[1]>=0) { // not the first answer
88                  MPI_Recv(&img_array[answer[1]][0], imgY, MPI_INT, answer[0],
89                           2, MPI_COMM_WORLD, &status);
90              }
91              MPI_Send(&i,  1, MPI_INT,    answer[0], 3, MPI_COMM_WORLD);
92              MPI_Send(&cx, 1, MPI_DOUBLE, answer[0], 4, MPI_COMM_WORLD);
93
94              cx=cx+resX;
95          }
96
97          // the remaining answers:
98          int term = -1;
99          for (int i=1; i<nproc; ++i) {
100             MPI_Recv(answer, 2, MPI_INT, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD,
101                     &status);
102             MPI_Recv(&img_array[answer[1]][0], imgY, MPI_INT, answer[0], 2,
103                     MPI_COMM_WORLD, &status);
104
105             //sending the termination signal
106             MPI_Send(&term, 1, MPI_INT, answer[0], 3, MPI_COMM_WORLD);
107         }
108
109         // we get another time at this point, so we can calculate the elapsed
110         // time for the calculation
111         e_time = MPI_Wtime();
```

```
112
113         cout << "Time elapsed during calculation: "
114             << e_time-s_time << " secs."
115             << endl;;
116
117         // file IO
118         myfile.open(fileName1);
119         myfile << "P3\r\n";
120         myfile << imgX;
121         myfile << " ";
122         myfile << imgY;
123         myfile << "\r\n";
124         myfile << "255\r\n";
125
126         // We have to colour our dataset. Actually, the members of the
127         // Mandelbrot set are used to be the same colour (black?) and have from
128         // point of visualisations view no interest.
129         //
130         // The outer points are represented by assigning colours to their
131         // iteration steps and this generates vivid forms and colors.
132         for (int i=0; i<imgX; i++) {
133
134             for (int j=0; j<imgY; j++) {
135                 // We go from black to red in this range
136                 if ( img_array[i][j] < 256) {
137                     myfile << img_array[i][j] << " 0 0";
138                     // (int)(84*pow(img_array[i][j],0.2)) << " 0 0";
139                     // //myfile << img_array[i][j] << " 0 0";
140                 }
141                 // we go from red to yellow in this range
142                 else if ( img_array[i][j] < 512) {
143                     myfile << "255 " << img_array[i][j]-256 << " 0";
144                 }
145                 // we go from yellow to white in this range
146                 else if ( img_array[i][j] < 768) {
147                     myfile << "255 255 " << img_array[i][j]-512;
148                 }
149
150                 /*
151                 // we could refine our palette for more resolution,
152                 //  more iteration-step images
153                 else if ( img_array[i][j] < 1024) {
154                     myfile << 1024-img_array[i][j] << " 255 255";
155                 }
156                 else if ( img_array[i][j] < 1280) {
157                     myfile << "0 "  << 1280-img_array[i][j] << " 255";
158                 }
159                 else if ( img_array[i][j] < 1536) {
160                     myfile << "0 0 "  << 1536-img_array[i][j];
161                 }
162                 */
163
164                 else {  // everything else is black
165                     myfile << "0 0 0 ";
166                 }
167                 myfile << " ";
```

```
            }
            myfile << "\r\n";
        }
        myfile.close();  // we close our file

        // Give another elapsed time info (IO included)
        e_time = MPI_Wtime();
        cout << "Time elapsed total: " << e_time-s_time << " secs \r\n";
    }

    // Slave
    else {
        // Prepare the step resolution
        resX = (xMax-xMin) / imgX;
        resY = (yMax-yMin) / imgY;

        int i;
        answer[0] = id;
        answer[1] = -1;
        MPI_Send(answer, 2, MPI_INT, 0, 1, MPI_COMM_WORLD);

        while (1) {
            MPI_Recv(&i, 1, MPI_INT, 0, 3, MPI_COMM_WORLD, &status);
            if (i<0) break; // got termination command!

            answer[1] = i;

            MPI_Recv(&question, 1, MPI_DOUBLE, 0, 4, MPI_COMM_WORLD, &status);

            // at every new step in X direction, we start at the first Y value
            cy = yMin;

            for (int j=0; j<imgY; j++) {
                img_line[j] = converges(question,cy);
                cy = cy + resY;
            }
            MPI_Send(answer, 2, MPI_INT, 0, 1, MPI_COMM_WORLD);
            MPI_Send(img_line, imgY, MPI_INT, 0, 2, MPI_COMM_WORLD);
        }
    }

    // Terminate MPI:
    MPI_Finalize();

    return 0;
}

// convergation function - base of the Mandelbrot set value generation
//
// it will get two parameters (x and y coordinates) and will give an iteration
// count in return
int converges (double cx, double cy) {
    int n=0;
    double zx=0;
    double new_zx=0;
    double zy=0;
```

```
224     // we iterate until max. iteration count iter_n, or until z^2 (complex!)
225     // runs over 4 - this means, our series will run to infinity,
226     // so it's not part of the set
227     while ( (n<iter_n) && (zx*zx + zy*zy)<4 ) {
228         // we work with complex numbers
229         // z * z => new_zx = (zx*zx - zy*zy)  new_zy = (zx*zy + zx*zy)
230         // z*z + c = zx^2 - zy^2 +cx   +  i(zx*zy*2 + cy)
231         new_zx = zx*zx - zy*zy + cx;
232         zy = 2*zx*zy + cy;
233         zx = new_zx;
234         n++;
235     }
236     return n;
237 }
238 
239 void usage(std::string program) {
240     cout << "Usage: " << endl;
241     cout << program << " out_file.ppm" << endl;
242     MPI_Abort(MPI_COMM_WORLD, 1);
243     exit(1);
244 }
```

Part II: Hello World!

Program Outputs and Results

```
ubuntu@ip-172-31-43-252:~/csci455/Lab1-Mandelbrot$ make sample_hello_world
mpicxx -std=c++14 -Wpedantic -Wall -Wno-missing-braces -Wextra -g -D_GLIBCXX_DEBUG -O0    hello_w
orld.cpp  -lm -o hello_world
Platform: Linux (96 cpu cores recognized)
for nodeprocs in 4 8 16 ; do \
        echo "\nMPIRUN hello_world with $nodeprocs node processes:" ; \
        mpirun --use-hwthread-cpus -np $nodeprocs ./hello_world ; \
done

MPIRUN hello_world with 4 node processes:
Hello world! From the root processor!
Hello world! From processor 2 of 4.
Hello world! From processor 3 of 4.
Hello world! From processor 4 of 4.
Hello world! From processor 1 of 4.

MPIRUN hello_world with 8 node processes:
Hello world! From the root processor!
Hello world! From processor 3 of 8.
Hello world! From processor 5 of 8.
Hello world! From processor 6 of 8.
Hello world! From processor 7 of 8.
Hello world! From processor 8 of 8.
Hello world! From processor 1 of 8.
Hello world! From processor 2 of 8.
Hello world! From processor 4 of 8.

MPIRUN hello_world with 16 node processes:
Hello world! From the root processor!
Hello world! From processor 1 of 16.
Hello world! From processor 2 of 16.
Hello world! From processor 3 of 16.
Hello world! From processor 4 of 16.
Hello world! From processor 5 of 16.
Hello world! From processor 6 of 16.
Hello world! From processor 7 of 16.
Hello world! From processor 8 of 16.
Hello world! From processor 9 of 16.
Hello world! From processor 10 of 16.
Hello world! From processor 11 of 16.
Hello world! From processor 12 of 16.
Hello world! From processor 13 of 16.
Hello world! From processor 14 of 16.
Hello world! From processor 15 of 16.
Hello world! From processor 16 of 16.
ubuntu@ip-172-31-43-252:~/csci455/Lab1-Mandelbrot$
```

Figure 4: Many runs (4, 8, 16 nodes) of the parallelized Hello World program on a 96 core host.

hello_world.cpp

```cpp
/* hello_world.cpp
 * ---------------
 * Authors: Darwin Jacob Groskleg, Laurence T. Yang
 * CSCI 455 Lab 1
 *
 * Purpose:  Print a greeting to the root processor, then all processors.
 *
 * Question: Can you complete the following "Hello World"" program and
 *           run with 4, 8 and 16 CPU processors?
 */
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>    /* MPI header file */

int main(int argc, char *argv[]) {
    // Initialize for MPI
    //  (must come before any other calls to MPI routines)
    //  will initialize MPI_COMM_WORLD, a global declared in mpi.h
    MPI_Init(&argc, &argv);

    // Get number of processes,
    //  sets it to nproc
    int nprocs;
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

    // Get this process's number (ranges from 0 to nprocs - 1)
    //  sets it to myid
    int myid;
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    // Please print a greeting with the root processor and all processors,
    //  respectively.
    // Therefore must use SPMD(single program,multi-device) computational model.
    if (myid == 0) { // is root
        printf("Hello world! From the root processor!\n");
    }
    // Block so root prints before all.
    MPI_Barrier(MPI_COMM_WORLD);
    // int fake_data = 0;
    //MPI_Recv(&fake_data, 0, MPI_INT, 0, MPI_ANY_TAG, MPI_COMM_WORLD,
    //          MPI_STATUS_IGNORE)
    // Now from all processors
    printf("Hello world! From processor %d of %d.\n", myid+1, nprocs);


    // Clean up for MPI
    //  (should come after all other calls to MPI routines)
    MPI_Finalize();

    return EXIT_SUCCESS;
}
```