CSCI 455: Lab #1 — Mandelbrot Example

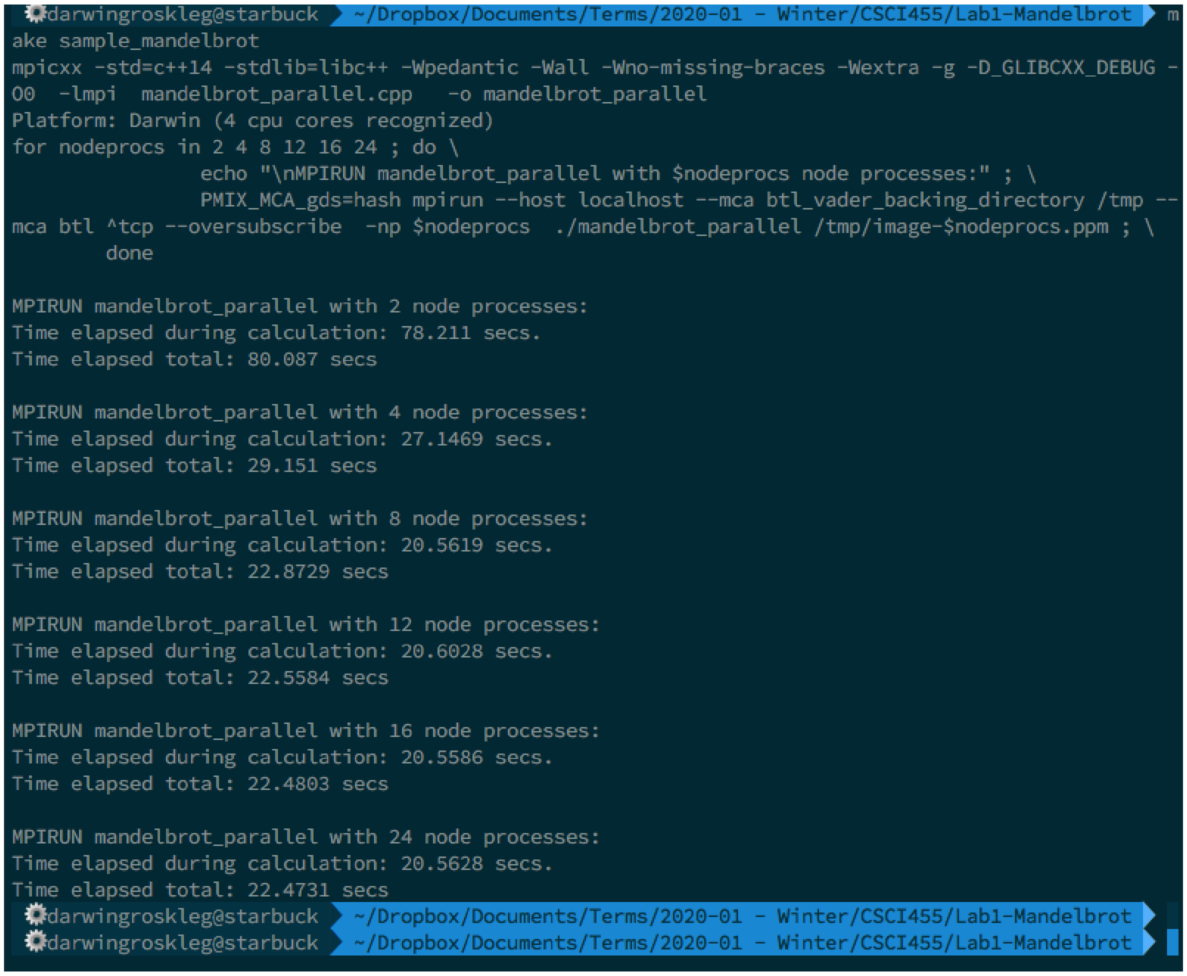
Darwin Jacob Groskleg

Winter 2020

Contents

Part I: Parallel Compute Mandelbrot Set

Program Outputs and Results



```
 darwingroskleg@starbuck  ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab1-Mandelbrot  m
ake sample_mandelbrot
mpicxx -std=c++14 -stdlib=libc++ -Wpedantic -Wall -Wno-missing-braces -Wextra -g -D_GLIBCXX_DEBUG -
O0  -lmpi  mandelbrot_parallel.cpp   -o mandelbrot_parallel
Platform: Darwin (4 cpu cores recognized)
for nodeprocs in 2 4 8 12 16 24 ; do \
                echo "\nMPIRUN mandelbrot_parallel with $nodeprocs node processes:" ; \
                PMIX_MCA_gds=hash mpirun --host localhost --mca btl_vader_backing_directory /tmp --
mca btl ^tcp --oversubscribe  -np $nodeprocs  ./mandelbrot_parallel /tmp/image-$nodeprocs.ppm ; \
        done

MPIRUN mandelbrot_parallel with 2 node processes:
Time elapsed during calculation: 78.211 secs.
Time elapsed total: 80.087 secs

MPIRUN mandelbrot_parallel with 4 node processes:
Time elapsed during calculation: 27.1469 secs.
Time elapsed total: 29.151 secs

MPIRUN mandelbrot_parallel with 8 node processes:
Time elapsed during calculation: 20.5619 secs.
Time elapsed total: 22.8729 secs

MPIRUN mandelbrot_parallel with 12 node processes:
Time elapsed during calculation: 20.6028 secs.
Time elapsed total: 22.5584 secs

MPIRUN mandelbrot_parallel with 16 node processes:
Time elapsed during calculation: 20.5586 secs.
Time elapsed total: 22.4803 secs

MPIRUN mandelbrot_parallel with 24 node processes:
Time elapsed during calculation: 20.5628 secs.
Time elapsed total: 22.4731 secs
 darwingroskleg@starbuck  ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab1-Mandelbrot 
 darwingroskleg@starbuck  ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab1-Mandelbrot 
```

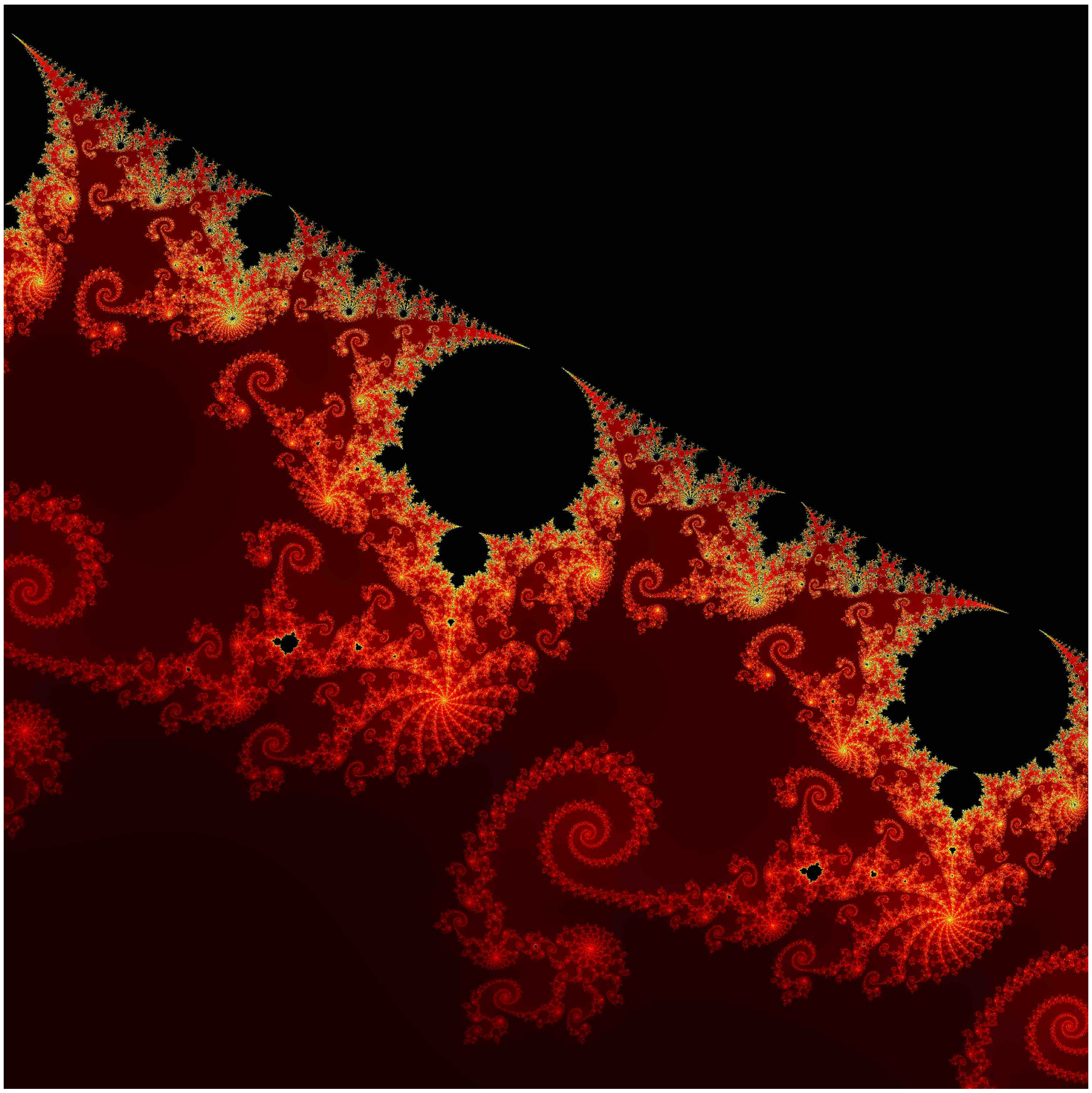


Figure 1: Generated mandelbrot set image (lo-res)

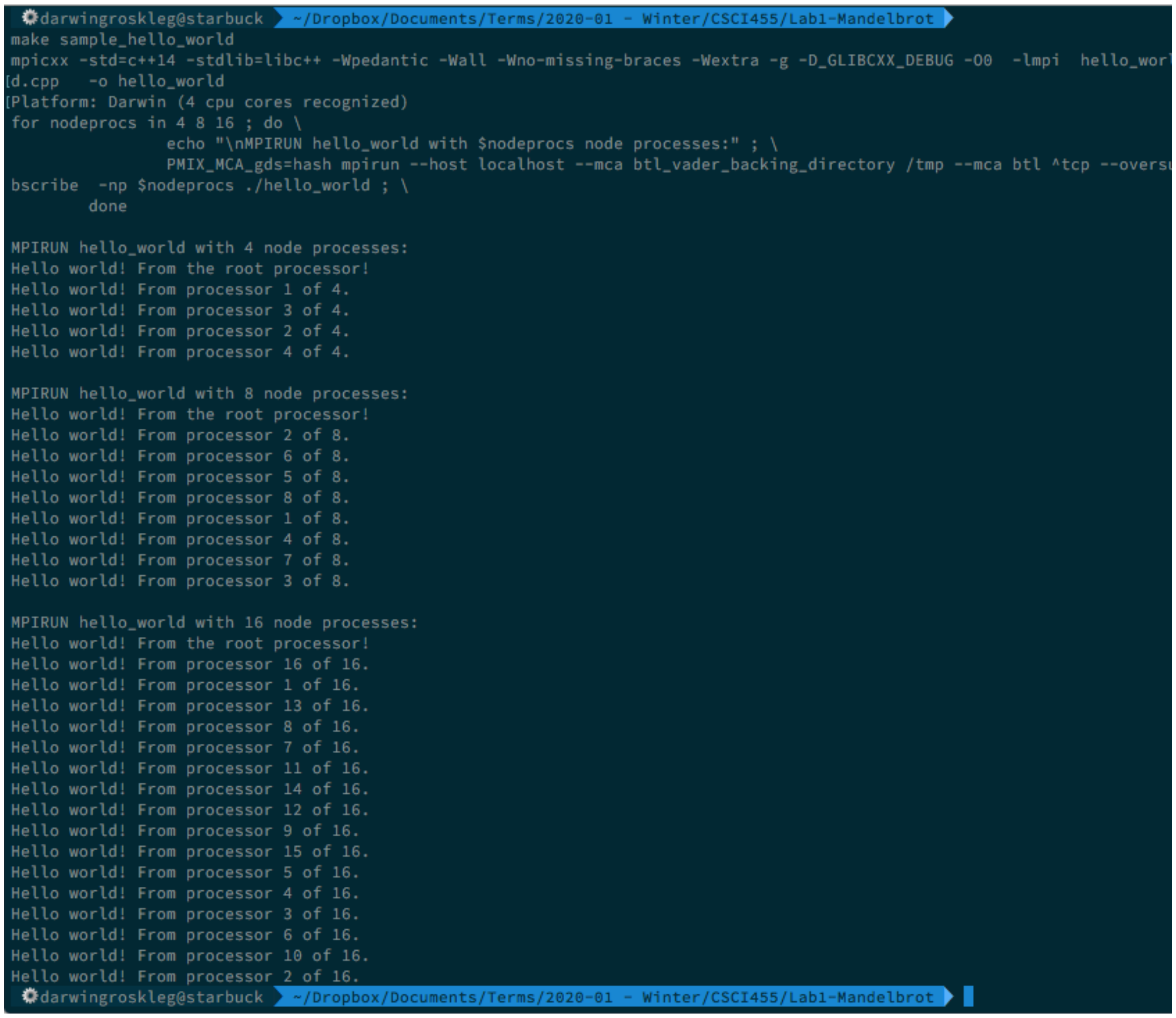
mandelbrot_parallel.cpp

```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:
```

```
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    // Get the total number of processors:
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);

    MPI_Barrier(MPI_COMM_WORLD); // for precise timing


    // Master
    if (id == 0) {
        if (argc<2)
            usage(argv[0]);

        ofstream myfile;  // we will write to this file
        string filename1(argv[1]);
        // filename1 = ”mandel.ppm”;
        char *fileName1 = (char*)filename1.c_str();

        //prepare the step resolution
        resX = (xMax-xMin) / imgX;
        resY = (yMax-yMin) / imgY;

        s_time = MPI_Wtime(); // we get a time value at this point

        // we do the calculation for every point of our complex plane,
        // thus on our 2D image with appropriate steps
        for (int i=0; i<imgX; i++) {
            MPI_Recv(answer, 2, MPI_INT, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD,
                    &status);
            // answer[0] -- from whom
            // answer[1] -- '-1' or the X coordinate

            if(answer[1]>=0) { // not the first answer
                MPI_Recv(&img_array[answer[1]][0], imgY, MPI_INT, answer[0],
                         2, MPI_COMM_WORLD, &status);
            }
            MPI_Send(&i,  1, MPI_INT,    answer[0], 3, MPI_COMM_WORLD);
            MPI_Send(&cx, 1, MPI_DOUBLE, answer[0], 4, MPI_COMM_WORLD);

            cx=cx+resX;
        }

        // the remaining answers:
        int term = -1;
        for (int i=1; i<nproc; ++i) {
            MPI_Recv(answer, 2, MPI_INT, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD,
                    &status);
            MPI_Recv(&img_array[answer[1]][0], imgY, MPI_INT, answer[0], 2,
                    MPI_COMM_WORLD, &status);

            //sending the termination signal
            MPI_Send(&term, 1, MPI_INT, answer[0], 3, MPI_COMM_WORLD);
        }

        // we get another time at this point, so we can calculate the elapsed
        // time for the calculation
        e_time = MPI_Wtime();

        cout << "Time elapsed during calculation: "
            << e_time-s_time << " secs."
            << endl;;

        // file IO
        myfile.open(fileName1);
        myfile << "P3\r\n";
        myfile << imgX;
        myfile << " ";
        myfile << imgY;
        myfile << "\r\n";
        myfile << "255\r\n";

        // We have to colour our dataset. Actually, the members of the
        // Mandelbrot set are used to be the same colour (black?) and have from
        // point of visualisations view no interest.
        //
        // The outer points are represented by assigning colours to their
        // iteration steps and this generates vivid forms and colors.
        for (int i=0; i<imgX; i++) {

            for (int j=0; j<imgY; j++) {
                // We go from black to red in this range
                if ( img_array[i][j] < 256) {
                    myfile << img_array[i][j] << " 0 0";
                    // (int)(84*pow(img_array[i][j],0.2)) << " 0 0";
                    // //myfile << img_array[i][j] << " 0 0";
                }
                // we go from red to yellow in this range
                else if ( img_array[i][j] < 512) {
                    myfile << "255 " << img_array[i][j]-256 << " 0";
                }
                // we go from yellow to white in this range
                else if ( img_array[i][j] < 768) {
                    myfile << "255 255 " << img_array[i][j]-512;
                }

                /*
                // we could refine our palette for more resolution,
                //  more iteration-step images
                else if ( img_array[i][j] < 1024) {
                    myfile << 1024-img_array[i][j] << " 255 255";
                }
                else if ( img_array[i][j] < 1280) {
                    myfile << "0 "  << 1280-img_array[i][j] << " 255";
                }
                else if ( img_array[i][j] < 1536) {
                    myfile << "0 0 "  << 1536-img_array[i][j];
                }
                */

                else {  // everything else is black
                    myfile << "0 0 0 ";
                }
                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;
    // we iterate until max. iteration count iter_n, or until z^2 (complex!)
    // runs over 4 - this means, our series will run to infinity,
    // so it's not part of the set
    while ( (n<iter_n) && (zx*zx + zy*zy)<4 ) {
        // we work with complex numbers
        // z * z => new_zx = (zx*zx - zy*zy)  new_zy = (zx*zy + zx*zy)
        // z*z + c = zx^2 - zy^2 +cx   +  i(zx*zy*2 + cy)
        new_zx = zx*zx - zy*zy + cx;
        zy = 2*zx*zy + cy;
        zx = new_zx;
        n++;
    }
    return n;
}

void usage(std::string program) {
    cout << "Usage: " << endl;
    cout << program << " out_file.ppm" << endl;
    MPI_Abort(MPI_COMM_WORLD, 1);
    exit(1);
}
```

Part II: Hello World!

Program Outputs and Results



```
 ⚙darwingroskleg@starbuck  ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab1-Mandelbrot 
make sample_hello_world
mpicxx -std=c++14 -stdlib=libc++ -Wpedantic -Wall -Wno-missing-braces -Wextra -g -D_GLIBCXX_DEBUG -O0  -lmpi  hello_wor
[d.cpp   -o hello_world
[Platform: Darwin (4 cpu cores recognized)
for nodeprocs in 4 8 16 ; do \
                echo "\nMPIRUN hello_world with $nodeprocs node processes:" ; \
                PMIX_MCA_gds=hash mpirun --host localhost --mca btl_vader_backing_directory /tmp --mca btl ^tcp --oversu
bscribe  -np $nodeprocs ./hello_world ; \
        done

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

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

MPIRUN hello_world with 16 node processes:
Hello world! From the root processor!
Hello world! From processor 16 of 16.
Hello world! From processor 1 of 16.
Hello world! From processor 13 of 16.
Hello world! From processor 8 of 16.
Hello world! From processor 7 of 16.
Hello world! From processor 11 of 16.
Hello world! From processor 14 of 16.
Hello world! From processor 12 of 16.
Hello world! From processor 9 of 16.
Hello world! From processor 15 of 16.
Hello world! From processor 5 of 16.
Hello world! From processor 4 of 16.
Hello world! From processor 3 of 16.
Hello world! From processor 6 of 16.
Hello world! From processor 10 of 16.
Hello world! From processor 2 of 16.
 ⚙darwingroskleg@starbuck  ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab1-Mandelbrot 
```

hello_world.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;
}
```