CS 5500
Spring 2021
Homework 4
By Griffin Hackley
A02224681

My implementation of this homework was done by using a leader and worker configuration. The leader sends each worker a line of the image to work on. The worker takes this line and does the iteration calculation. The worker then sends the data back to the leader and is given a new line to work on if there are any left. After each line has been calculated the leader takes the iteration data and determines the colors for each pixel and adds them to the file

The timing of the program execution is also done by the leader. It starts after some of the values are initialized in the beginning of the program, but before any work is sent to the workers. The end time is taken after the leader has put each pixel into the output file.

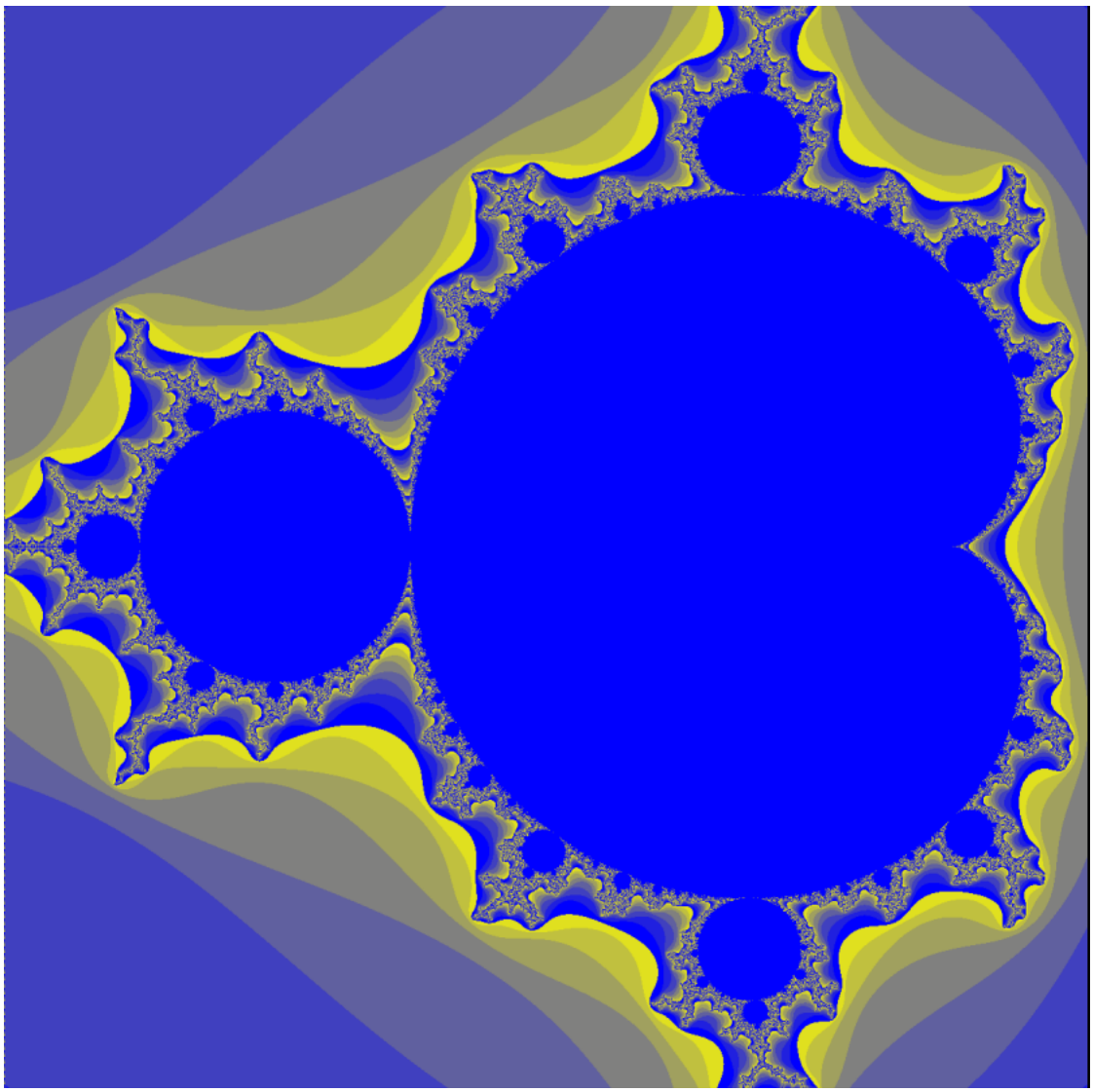
I decided that I liked the blue and yellow color scheme. This color scheme was obtained by flipping the blue color value and leaving the green and red the same.

The command to compile the code is:
mpic++ hw6.cpp

The command to run the code is:
mpirun -np (num) -oversubscribe a.out

where (num) is the number of processes

Below are the sample results



Full code is on the following pages

griffin > Desktop > HW6 > main.cpp > main(int, char **)

```cpp
void byLine(int DIM, int line, Complex c1, Complex c2, ofstream& fout, int maxIters){
    int iterations[DIM+1];
    iterations[0] = line;
    Complex c;
    for(int i = DIM; i > 0; i--){
        // calculate one pixel of the DIM x DIM image
        c.r = (i*(c1.r-c2.r)/DIM)+c2.r;
        c.i = (line*(c1.i-c2.i)/DIM)+c2.i;
        int iters = mbrot(c,maxIters);
        iterations[i] = iters;
    }
    MPI_Send(iterations, DIM+1, MPI_INT, 0, 0, MCW);
}

int main(int argc, char **argv){
    int size, rank, length, data;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);
    MPI_Status status;
    ofstream fout;

    //below are the variables that can be changed to porduce a different image
    int maxIters = 255;
    Complex c1,c2;
    c1.r = -1.5;
    c1.i = -1;
    c2.r = .5;
    c2.i = 1;
    int DIM = 1250;

    if(rank == 0){
        time_t start = clock();
        cout << "Starting Timer" << endl;

        int storage[DIM][DIM];
        int completed[DIM];
        fout.open("result.ppm");

        fout << "P3"<< endl;
        fout << DIM << " " << DIM << endl;
        fout << "255" << endl;

        //send initial lines
        for(int i = 1; i < size; i++){
            data = DIM-i+1;
            MPI_Send(&data, 1, MPI_INT, i, 0, MCW);
        }

        int iterations[DIM+1];
        for(int i = DIM-size+1; i > 0; i--){
            //recieve lines from processes
            MPI_Recv(iterations, DIM+1, MPI_INT, MPI_ANY_SOURCE, 0, MCW, &status);

            //send new work back until there are no lines left
            data = i;
            MPI_Send(&data, 1, MPI_INT, status.MPI_SOURCE, 0, MCW);

            //get the pixel values for each iteration
            for(int k = DIM-1; k >= 0; k--){
                storage[iterations[0]-1][k] = iterations[k];
            }
            fout << endl;
        }

        //recieve the left overs
        for(int i = 0; i < size-1; i++){
            MPI_Recv(iterations, DIM+1, MPI_INT, MPI_ANY_SOURCE, 0, MCW, MPI_STATUS_IGNORE);
            for(int k = DIM-1; k >= 0; k--){
                storage[iterations[0]-1][k] = iterations[k];
            }
        }

        for(int o = 0; o < DIM; o++){
            for(int k = DIM; k > 0; k--){
                int iters = storage[o][k];
                fout << rcolor(iters, maxIters)<<" ";
                fout << gcolor(iters, maxIters)<<" ";
                fout << bcolor(iters, maxIters)<<" ";
            }
        }

        //kill all processes
        data = -1;
        for(int i = 1; i < size; i++){
            MPI_Send(&data, 1, MPI_INT, i, 0, MCW);
        }
        fout.close();

        //stop timer and print time (timing code was taken from here: https://stackoverflow.com/questions/12231166/timing
        time_t stop = clock();
        double time = difftime(stop, start) / 1000000.0;
        time = ceil(time * 100.0) / 100.0;
        cout << "Execution took " << time << " seconds" << endl;

    //if not process 0
    } else {
        while(data != -1){
            //recieve line to work on
            MPI_Recv(&data, 1, MPI_INT, 0, 0, MCW, MPI_STATUS_IGNORE);

            if(data != -1){
                //do the work on the line
                byLine(DIM, data, c1, c2, fout, maxIters);
            }
        }
    }

    MPI_Finalize();

    return 0;
}
```

```cpp
#include <iostream>
#include <fstream>
#include <mpi.h>
#include <chrono>
#include <cmath>

using namespace std;

#define MCW MPI_COMM_WORLD

struct Complex {
    double r;
    double i;
};

Complex operator + (Complex s, Complex t){
    Complex v;
    v.r = s.r + t.r;
    v.i = s.i + t.i;
    return v;
}

Complex operator * (Complex s, Complex t){
    Complex v;
    v.r = s.r*t.r - s.i*t.i;
    v.i = s.r*t.i + s.i*t.r;
    return v;
}

int rcolor(int iters, int maxIters){
    if(iters == maxIters) return 0;
    return 32*(iters%8);
}

int gcolor(int iters, int maxIters){
    if(iters == maxIters) return 0;
    return 32*(iters%8);
}

int bcolor(int iters, int maxIters){
    if(iters == maxIters) return 255;
    return 255 - (32*(iters%8));
}

int mbrot(Complex c, int maxIters){
    int i=0;
    Complex z;
    z=c;
    while(i<maxIters && z.r*z.r + z.i*z.i < 4){
        z = z*z + c;
        i++;
    }
    return i;
}
```