Serial Mandelbrot

Aaron Morgenegg

10/1/18

OpenMPI Serial Mandelbrot

This program generates a mandelbrot image using a single mpi process. Timing the program with a resolution of 512x512 resulted in a runtime of about half a second. The calculatePixel method, which does the bulk of the work, is set up in a way to allow the program to be easily parellelized. I included the generated image below in this report.

```
/*
Example of Compiling and running code.

mpic++ main.cpp
mpiexec -np 2 a.out

#include <iostream>
#include <fstream>
#include <time.h>
#include <stdlib.h>
#include <mpi.h>
#include <unistd.h>
#include <algorithm>
#include <math.h>
#include <string>

const int MAX_ITERATION = 1000;
const int RESOLUTION = 512;
const bool INVERT_COLORS = false;
const std::string OUTPUT_FILE = "serial_mandelbrot.ppm";
```

```
struct Color{
public:
int r;
int g;
int b;

Color(){r,g,b=0;}

Color(int r, int g, int b){
this->r = r;
this->g = g;
this->b = b;
}
};

Color getColor(int iteration){
int r = iteration%145;
int g = iteration%200;
int b = iteration%255;
if(INVERT_COLORS){
r = 255 - r;
g = 255 - g;
b = 255 - b;
}
return Color(r,g,b);
}

void writeToFile(std::string message, std::ofstream &mandelbrot_file){
mandelbrot_file << message;
}

void plotImage(Color plot[][RESOLUTION], std::ofstream &mandelbrot_file){
for(int i = 0; i < RESOLUTION; i++){
for(int j = 0; j < RESOLUTION; j++){
std::string color = std::to_string(plot[i][j].r) + " " + std::to_string(plot[i][j]
writeToFile(color, mandelbrot_file);
}
```

```
writeToFile("\n", mandelbrot_file);
}
}

Color calculatePixel(int px, int py){
double x0 = -2.5 + px * (3.5/RESOLUTION);
double y0 = -2 + py * (3.5/RESOLUTION);
double x = 0.0;
double y = 0.0;
int iteration = 0;
while(x*x + y*y < 4 && iteration < MAX_ITERATION){
double xtemp = x*x - y*y + x0;
y = 2*x*y + y0;
x = xtemp;
iteration += 1;
}
return getColor(iteration);
}

void mandelbrot(std::ofstream &mandelbrot_file){
Color plot[RESOLUTION][RESOLUTION];
for(int i = 0; i < RESOLUTION; i++){
for(int j = 0; j < RESOLUTION; j++){
plot[i][j] = calculatePixel(j, i);
}
}
plotImage(plot, mandelbrot_file);
}

std::ofstream setupFile(){
std::ofstream mandelbrot_file(OUTPUT_FILE);
mandelbrot_file << "P3" << std::endl;
mandelbrot_file << RESOLUTION << " " << RESOLUTION << std::endl;
mandelbrot_file << "255" << std::endl;
return mandelbrot_file;
}
```

```
int main(int argc, char** argv) {
MPI_Init(&argc, &argv);
srand(time(NULL));
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

if(world_rank == 0){
std::ofstream mandelbrot_file = setupFile();
mandelbrot(mandelbrot_file);
mandelbrot_file.close();
}
MPI_Finalize();
return 0;
}
```