CS5500 HW6

Chase Mortensen A01535275

October 2019

1 Description

Make your mandelbrot program run as quickly as possible by taking advantage of MPI library routines. Get some timing results from running your program..

My implementation is based off of the mbrot.cpp file that was discussed in class. It takes three arguments: re0, im0, and re1. It generates a mandelbrot image in a ppm format. The coloring is created using a combination of modulus, square root, and division operations. The program checks up to 1024 iterations. The program is timed using the `MPI_Wtime` function. The parallel version below involves different processes

2 Program

```
#include <iostream>
#include <fstream>
#include <math.h>
#include <mpi.h>
#include <vector>

#define MCW MPI_COMM_WORLD
using namespace std;

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 mbrotIters(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;
}

int main(int argc, char **argv){
    double re0 = atof(argv[1]);
    double im0 = atof(argv[2]);
    double re1 = atof(argv[3]);

    double t1, t2;
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);

    int pixels = 512;
    vector<int> vec(pixels);

    Complex pixel;
    int tmp;

    ofstream fout("mandelbrot.ppm");
    if (!rank){
        fout << "P3" << endl;
        fout << pixels << " " << pixels << endl;
        fout << "256" << endl;

        t1 = MPI_Wtime();
    }
```

```
    MPI_Barrier(MCW);

    for (int i = 0; i < pixels; i++){
        // if (!rank) cout << i%size << endl;
        if (i % size == rank) {
            for (int j = 0; j < pixels; j++){

                pixel.r = ((re0 + i / 512.0) * 2) - re1;
                pixel.i = ((im0 + j / 512.0) * 2) - 1;

                tmp = mbrotIters(pixel, 1024);
                vec[j] = tmp;
            }
            if (rank) {
                MPI_Send(&vec[0],1,MPI_INT,0,0,MCW);
            }

        }

        if (!rank) {
            if (i % size != 0) {
                MPI_Recv(&vec[0],1,MPI_INT,i%size,0,MCW,MPI_STATUS_IGNORE);
            }
            for (int i = 0; i < pixels; i++) {
                fout << " " << 256 - (vec[i] % 256) << " " << 256 - floor(vec[i] / 4)
                << " " << floor(sqrt(vec[i]) * 8) - 1 << "  ";
            }
            fout << endl;
        }
    }

    if (!rank) {
        t2 = MPI_Wtime();

        cout << "Time: " << t2 - t1 << endl;
    }

    MPI_Finalize();

    return 0;
}
```

3 Output

```
$ mpic++ mandelbrot_parallel.cpp
$ mpirun -np 1 ./a.out 0 0 1.5
Time: 3.76386

$ mpic++ mandelbrot_parallel.cpp
$ mpirun -np 2 ./a.out 0 0 1.5
Time: 2.16489

$ mpic++ mandelbrot_parallel.cpp
$ mpirun -np 3 ./a.out 0 0 1.5
Time: 1.77713

$ mpic++ mandelbrot_parallel.cpp
$ mpirun -np 4 ./a.out 0 0 1.5
Time: 1.73888
```