Parallel Programming HW5 Write Up
Due Monday March 1, 2021, 11:59 pm

Connor Osborne
A01880782

Monday March 1, 2021

1 Description

For homework 5, I wrote a parallel program that creates a 512x512-pixel mandelbrot image in ppm format and got timing results from running your program.

2 Implementation

The implementation for this program fully utilizes the mbrot.cpp provided by Dan Watson with added utilities in between. First in main() the MPI variables are set and the MPI processes are started. Then an array for holing the information for each pixel is created. It holds the x, y, red, blue, and green values of a given pixel.

```
    int rank, size;
    int DIM = 512;

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

    Complex c1,c2,c;
    c1.r = -1;
    c1.i = -1;
    c2.r = 1;
    c2.i = 1;

    int pixel[5];

    int iters;
```

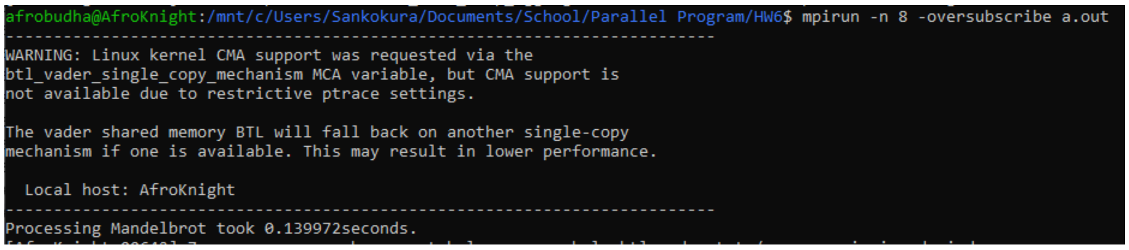
The rest of the program is based on the boss-worker paradigm. Process zero creates a 2d array of arrays of size 3 for storing the finished pixel information. Next process 0 starts a timer for timing purposes. Then in a nested for loop process zero sends a pixel to be set up by each process and then it stores the pixels into the 2d array. Once process zero has gone through every pixel and added their info to the 2d array it sends a stop message to every other process by marking the first value of pixel as 5000. Once this is done Process 0 stops the time and prints out how long creating the Mandelbrot took.

```
    if (rank == 0){
        //do stuff
        int dest = 1;
        std::chrono::high_resolution_clock::time_point t1 =
            std::chrono::high_resolution_clock::now();
        ofstream fout;
        fout.open("testing5.ppm");

        int picture[DIM][DIM][3];

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

        for(int j=0;j<DIM;++j){
            for(int i=0;i<DIM;++i){
                // calculate one pixel of the DIM x DIM image
                pixel[0] = i;
                pixel[1] = j;
                pixel[2] = 0;
                pixel[3] = 0;
                pixel[4] = 0;
                if(dest == size){
                    int myFlag;
                    MPI_Iprobe(MPI_ANY_SOURCE, MPI_ANY_TAG, MCW, &myFlag, &mystatus);
                    while(myFlag){
                        MPI_Recv(pixel, 5, MPI_INT, MPI_ANY_SOURCE, 1, MCW, MPI_STATUS_IGNORE);
                        picture[pixel[1]][pixel[0]][0] = pixel[2];
                        picture[pixel[1]][pixel[0]][1] = pixel[3];
                        picture[pixel[1]][pixel[0]][2] = pixel[4];
                        MPI_Iprobe(MPI_ANY_SOURCE, MPI_ANY_TAG, MCW, &myFlag, &mystatus);
                    }
                    dest = 1;
                }
                MPI_Send(pixel, 5, MPI_INT, dest, 0, MCW);
                dest++;
            }
        }

        pixel[0] = 5000;
        for (int i = 1; i < size; ++i) {
            MPI_Send(pixel, 1, MPI_INT, i, 0, MCW);
        }

        for(int j=0;j<DIM;++j){
            for(int i=0;i<DIM;++i){

                fout << picture[j][i][0] <<" ";
                fout << picture[j][i][1] <<" ";
                fout << picture[j][i][2] <<" ";
            }
            fout << endl;
        }
        fout.close();

        std::chrono::high_resolution_clock::time_point t2 =
            std::chrono::high_resolution_clock::now();
        std::chrono::duration<double> time_span =
            std::chrono::duration_cast<std::chrono::duration<double>>(t2 - t1);

        cout << "Processing Mandelbrot took " << time_span.count() << "seconds." << endl;
    }
```

All processes besides process 0 provide the computations of the Mandelbrot. First they receive the pixel information from process 0. Then they the real and imaginary parts of the complex number needed for generating the correct color by calling mbrot() using c. Finally each process will compute the red, blue and green values for the pixel and send the pixel array back to process 0.

```
    else{
        while(1){
            MPI_Recv(pixel, 5, MPI_INT, 0, 0, MCW, &mystatus);
            if(pixel[0] == 5000){
                break;
            }
            else{
                c.r = (pixel[0]*(c1.r-c2.r)/DIM)+c2.r;
                c.i = (pixel[1]*(c1.i-c2.i)/DIM)+c2.i;
                iters = mbrot(c,255);
                pixel[2] = rcolor(iters);
                pixel[3] = gcolor(iters);
                pixel[4] = bcolor(iters);
                MPI_Send(pixel, 5, MPI_INT, 0, 1, MCW);
            }
        }
    }
}
```

3 Input and Output with 8 processes: results vary per run of the program.



```
afrobudha@AfroKnight:/mnt/c/Users/Sankokura/Documents/School/Parallel Program/HW6$ mpirun -n 8 -oversubscribe a.out
--------------------------------------------------------------------------
WARNING: Linux kernel CMA support was requested via the
btl_vader_single_copy_mechanism MCA variable, but CMA support is
not available due to restrictive ptrace settings.

The vader shared memory BTL will fall back on another single-copy
mechanism if one is available. This may result in lower performance.

  Local host: AfroKnight
--------------------------------------------------------------------------
Processing Mandelbrot took 0.139972seconds.
```