Guillermo Alvarez
0949951

Assignment 2

Assignment A2 was done in a similar fashion to assignment one. The main non-parallel program julia.c was modified and then wrapped by a parallel program parallel-julia.c. Main.c starts the program and delivers input data to the other processess.

The algorithm to devide the load is kept very simple. The total number of pixels is divided among the processes evenly. For instance if the image is 3x3 pixels wide, and there are 3 processors, each process would get 3 pixels to process on. P0 would get pixels 1-3, P1 gets pixels 4-6 and P2 gets pixels 7-9. Each process, knowing the width and height of the image, is able to calculate the pixel's x and y coordinate for each pixel they are processing on. Conveniently, MPIGather returns all of the calculated data in the correct order.

Because the load-division algorithm is simple, it is optimal in only some cases. If in a set there are many points which iterate infinitely, i.e we may see white spots or a very large white areas on the image, then those processes which are tasked to compute those pixels will take a long time and in some cases become the bottleneck process. This is the case with image4.bmp which included in this submission. From looking at the stats4.txt file, we see that Processes which focus on pixels in the center of the image take the longest while those on the outside take a very short amount of time. In images 1, 2 and 3 there are not a lot of pixels which iterate infinitely or if there are they are spread out. Image2.bp shows similar results to image4.bmp but with fewer white spots.

Several things could be done to improve performance.Instead of dividing load by pixel count have a process compute entire lines at a time. The root process could act as a server to tell which lines are available to be processesed. Processes which are done their lines could request more lines. Of course, this increases the code complexity and raises overhead of MPI calls. However it would distribute the load among the processes more evenly regardless of the presense of infinitely iterating points.