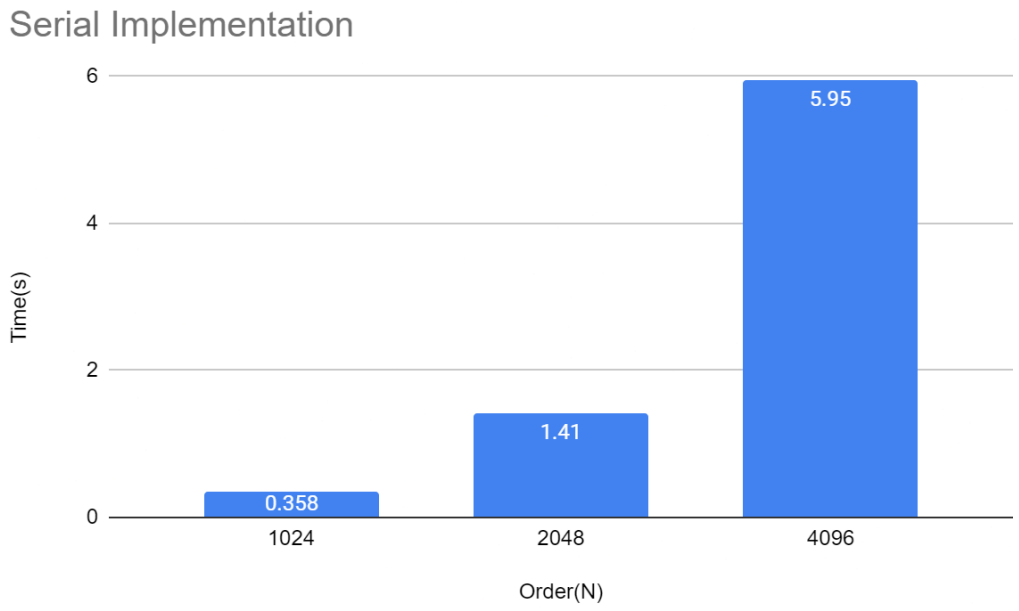
Project 2 Report

Christopher Chen, Kyungbong Ko

Mandelbrot Serial

Much of the initialization code was adapted from file heat.c of the previous project. This included the parsing of CLI arguments and parts of saving the result into the appropriate output file. In this project specifically, we followed the PGM format and wrote to the file accordingly.

The main part of this implementation was the loop that calculated the z_n values for each pixel. We implemented a standard double for loop to loop through each pixel and for each pixel, iterated through the equation $z_{n+1} = z_n + c$ until the magnitude was greater than 2 or the number of iterations hit the cutoff value. A few things to note is that we opted to use two doubles to represent each complex number, which allowed us to encode equations that would calculate the real and imaginary part of z_{n+1} for each z_n. Additionally, we initially thought that the cutoff would provide the ratio to calculate the grayscale pixel value(out of 255), but we realized that the number of iterations directly correlated to the pixel value and the cutoff just provided a boundary. The graph below shows the increase in time for doubling the order(zoom level 15, cutoff at 255):



Mandelbrot OpenMPI

The main paradigm we aimed to implement was the manager-workers approach where the root process(with rank 0) would configure and send the appropriate data to the rest of the "worker" processes and handle the organization of the data after they return the data.

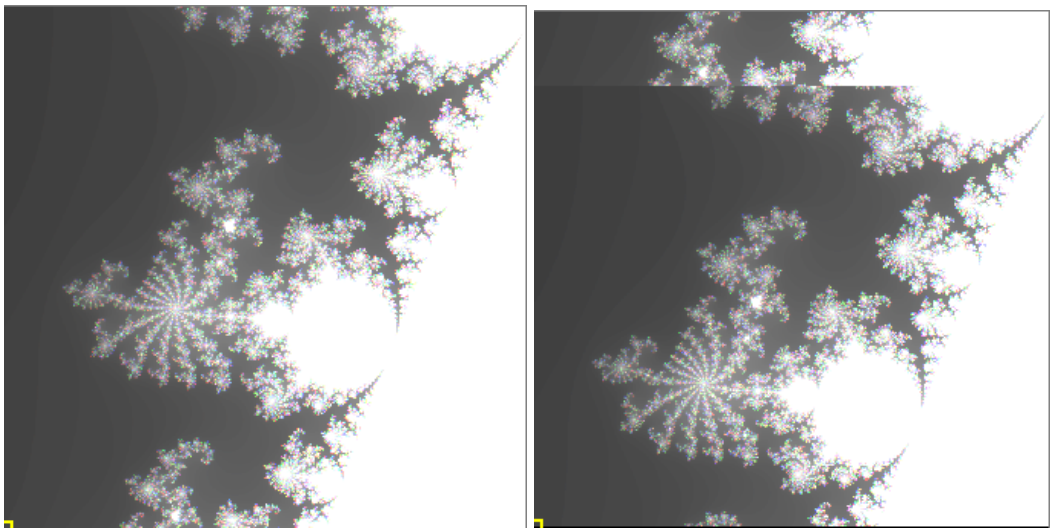
More specifically, we implemented the manager_main() function that used MPI_Bcast() to send the CLI arguments(order, cutoff, coordinates of the center, and zoom level) to each worker process after processing it in the root process, allowing each process to have the correct arguments. In order to accomplish this, we implemented a custom MPI type named mpi_mandelbrot_args to create efficient and simple message broadcasting. The function would also be where the final image and map is processed into a PGM file.

The manager_loop was also in charge of creating the receive buffer and setting up MPI_Gatherv() to receive information from the worker processes and store it into the final map. MPI_Gatherv() was chosen because we noticed that the majority of the communication was in each worker process sending the rows of pixels calculated to the manager process.

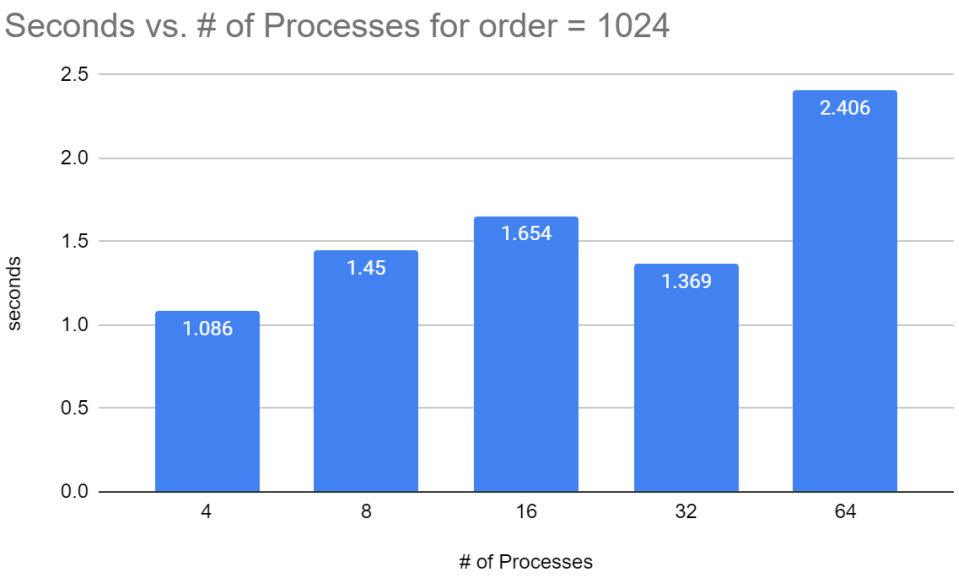
For the worker functions, each worker calculated the rows it was assigned to based on their rank and on the number of active workers/processes. Each worker would have their own temporary buffer where they calculate their section of the map and is shifted back to (0,0) so we can send it through the send buffer of MPI_Gatherv().

We initially calculated the rows for each worker process to calculate and excluded the manager process, where we had the process with rank 1 calculate what would have been calculated for the process with rank 0(manager process) and so on and so forth(process rank n would calculate the pixels for rank n - 1).

However, MPI treated the rows calculated to be placed in rank n was placed in rank n - 1, resulting in the image to be shifted. To fix this, we shifted the calculations back and had the manager process to calculate its share as well.



Shifted Image on the right



When it comes down to the time it seems that having 4 processes yielded the fastest time. We saw a general linear trend when doubling the # of processors, except when the number of processes was 32. This was a surprising result to us since we assumed that as the number of processes increased, the time to calculate the image should decrease but that wasn't the case. We suspect that this may be due to the variance of GPUs and processing powers of the different CSIF computers. Where some computers' processing power is much faster than others and incorporating more would cause a larger variance for the time completed.