Project 3 Report

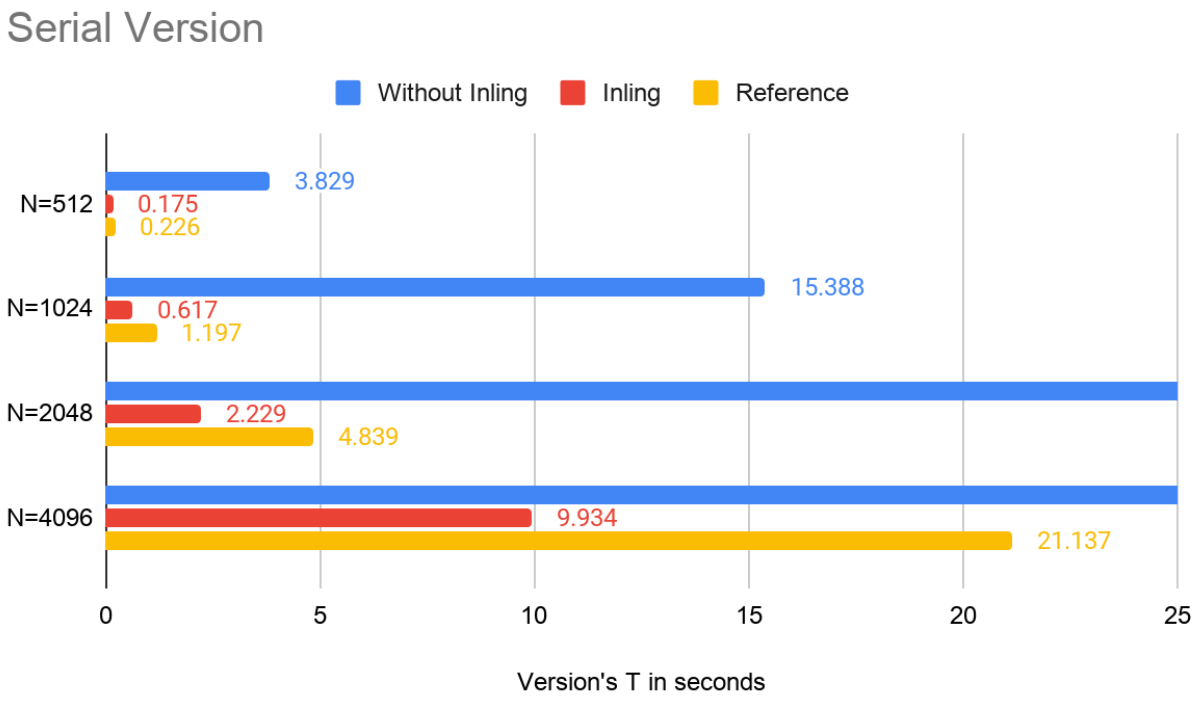
Zijian Feng, SID: 916024691
Yuxuan Huang, SID: 916509132

Part1: Serial Version

The basic structure of the serial implementation falls into 3 steps: 1.Retrieving necessary data from command line, 2. Steps in to the order to initialize a mandelbrot calculation(i.e. Zn+1 = Zn^2 +c),3. Setting the final brightness number (0~ n), and 4. Writing the final pgm image.

Mainly, the optimization happened in step 2, mandelbrot calculation, where switching from using library functions to manually calculating with primitive signs resulted in a significant speed up in calculation, which is basically equivalent to inlining small functions.

Here are the average time for both versions with the same argument N-wise comparing to the reference program(Note here data points exceed the right limit will not be fully represented.):



There seems to be a trend that the inlined version is 3 times faster than the one without inlining and 2 times faster than the reference program. In essence, as this is also mentioned on piazza, library functions that serve as general function calls for different types include checking mechanism and other operations. By manually using a branch of the function, it will improve the performance, especially when it is called in an enormous for loop.

A side testing was also conducted on using default malloc instead of aligned_alloc(cache_size,size), yet there was not much improvement on speed. It may be the case that this program is not as reading heavy as Project one was. The cache only needs to store some local variables that are going to be modified until some conditions are not met so that they are written back to a memory location.

Part 2: MPI Version

Since the MPI version does not change the base logic of implementation of the serial version, this part will be mainly focused on explaining how we used OpenMPI and why we used it in such ways.

Part 2.1 Using MPI

We used the suggested manager-workers approach. Besides it being suggested, it also is the more optimal approach for this program. Since every workers' tasks will never be the same as another worker, and workers do not need to communicate with each other.

Specifically, we have the first process (rank#0) being the manager, and its task is to distribute an "equal" amount of workload to each worker (including itself). The manager divides the whole task into N equal portions (N being the number of processes we have in total). Since the only difference between each portion is the range of index it is in, the manager only needs to send the beginning and ending index to each worker via function MPI_Scatter(). Sending only the two index largely reduces the amount of data being sent every time, which becomes especially beneficial when dealing with a workload that is relatively heavy.

Each worker will allocate a local block memory that is the right size for its task. After finishing the task, the block of data will be sent back to the manager via function MPI_Gather(). It achieves the same goal of minimizing the amount of data being sent.

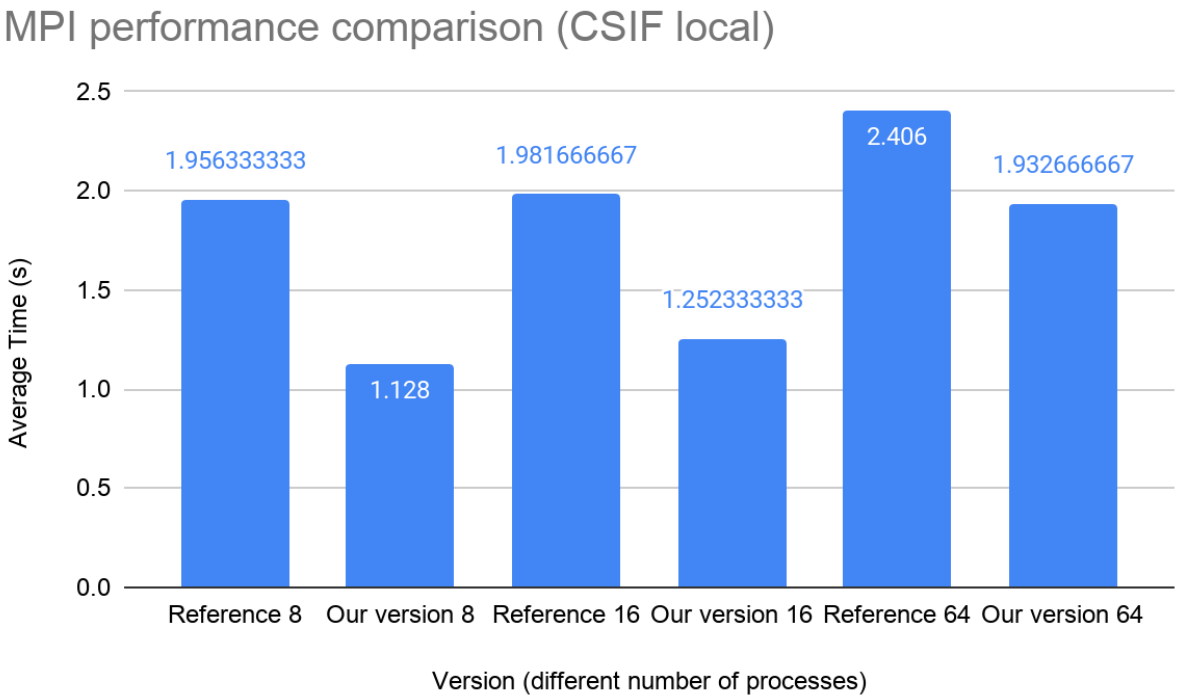
Part 2.2 Other approach for increasing performance

Besides usage of OpenMPI described above, we also practiced methods including but not limited to: increasing locality, avoiding repeated inline functions calls, avoiding repeated calculations.

Part 2.3 Testing results

All tests are ran with the input "2048 -0.722 0.246 15 255".

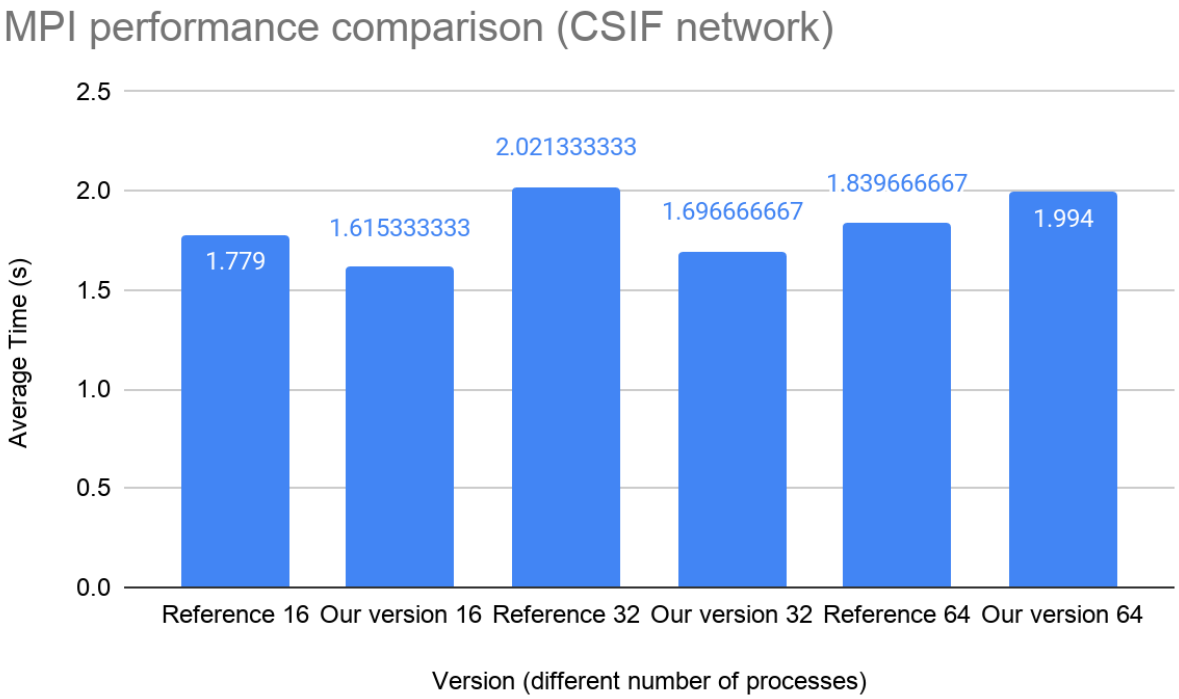
When running locally, we were able to achieve performance that is consistently better thant he reference program. Results are shown in the chart below:



Since we do not know how the reference program is implemented, we suppose that the performance advantage mainly comes from our way of minimizing data transfer. The reason being that we believe that we still are not using the best implementation for this program, which is a fine-grain distribution which the manager assigns smaller tasks to each worker and have them report back to the manager when they are done. We think this might be a better approach because Mandelbrot has uneven computation at each point. Using the the fine-grain method can potentially minimize the time each worker being idle.

We also observed that both our program and the reference program's performance drops as the workload being divided into more portions. One possible cause of it is again, Mandelbrot having uneven computation at each point. Since the manager needs to wait for all the workers to finish their assigned tasks, the difference between time spent by the "fastest" worker and the "slowest worker" will be enlarged.

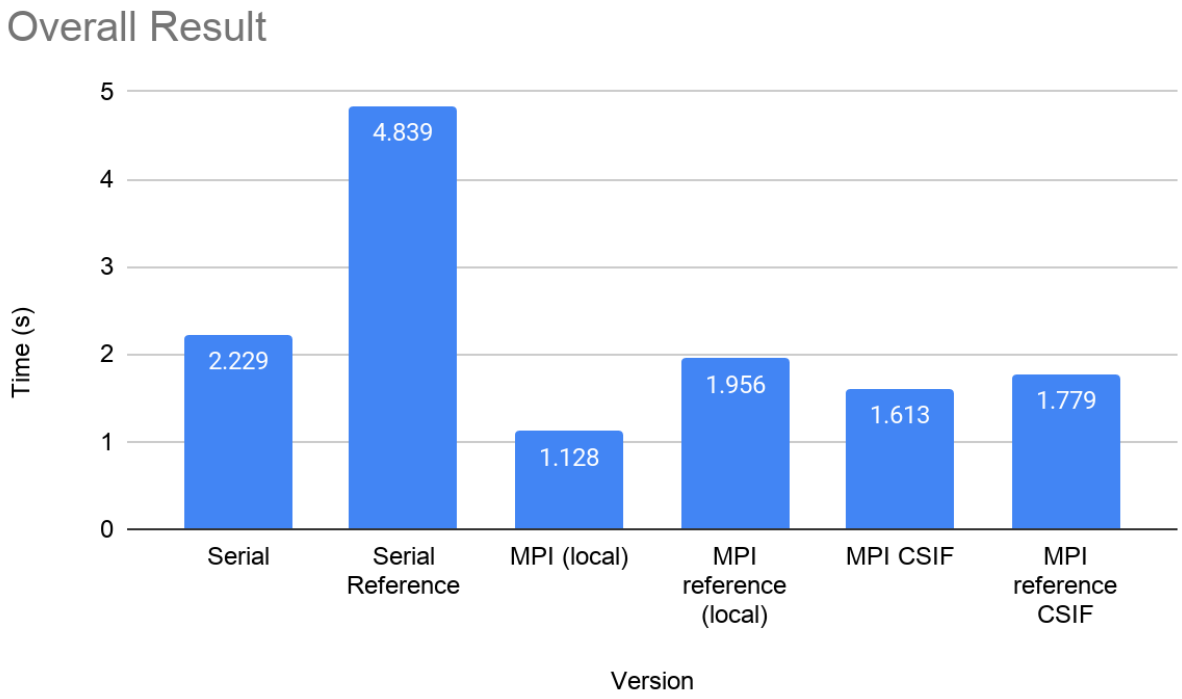
Furthermore, when it comes to the performance on CSIF network, we were able to achieve consistent better performance when assigning less than less cores. The results are shown in the chart below:



However, unlike the reference program which were able to keep similar performance when using local and CSIF networks, our program's performance is significantly worse compared to using local execution. With the case of using 64 processors being an exception. We supposed that the communication speed between CSIF computers might be a factor. As each worker needs to send back less data, less time is spent during data transferring. However, the versions using 16 and 32 processors are still faster than the version using 64. Which suggests that the amount of data each worker sends is not the main factor of this observation.

Part 3: Conclusion

Overall using OpenMPI significantly boosts the performance. Although, when encountering relatively smaller tasks, local multi-core execution is faster than using parallel execution that depends on the network. But both of them have a significant advantage when compared to the serial version. Results are shown as below:


Overall Result
Time (s)
5
4
3
2
1
0
4.839
2.229
1.128
1.956
1.613
1.779
Serial
Serial Reference
MPI (local)
MPI reference (local)
MPI CSIF
MPI reference CSIF
Version