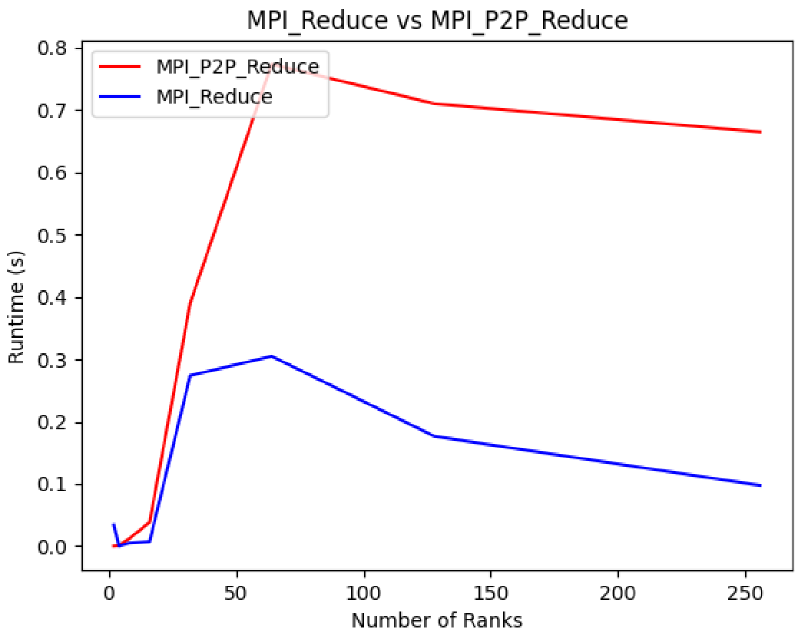
CSCI 4320 Assignment 3 Report

Zhiqi Wang wangz56@rpi.edu

March 3, 2023

After implementing the `MPI_P2P_Reduce` function, I ran it with different number of ranks on AiMOS and compared the results with the `MPI_Reduce` function.



Generally, while as the number of ranks increases from 2 ranks to 32 ranks, both function slows down. This could be due to the the over head of sending/receiving takes longer for more ranks. Then there's a slight speed up for both function starting at 64 ranks, which we starting to use more than one node. This is suprising since we are using more than one node, which should create more overhead which would slow down the program (under the assumption that communication across nodes is slower than within nodes). However, this is not the case. This could be due to the fact that as the number of ranks increases, the size of the data we are sending is decreasing. But this is not convincing since the size of the data don't change much since the data type is fixed(`LONG_LONG_INT`). In conclusion, I can't explain why there's a slight speed up for both function starting at 64 ranks. But what makes sense is that the `MPI_P2P_Reduce` 's speed up is not as significant as the `MPI_Reduce` 's speed up, since it requires more rounds of sending/receiving.

The `MPI_P2P_Reduce` function is slower than the `MPI_Reduce` function for all number of rank configurations. This is because the `MPI_P2P_Reduce` function need to receive/send messages for $\log_2$(number of ranks) times while the

`MPI_Reduce` function only need to send/receive messages for one time. We can see that as the number of ranks increases, the `MPI_P2P_Reduce` function takes more and more time since there's more rounds of sending/receiving messages and the overhead of each round is not negligible.