CSCI 6360: Parallel Computing Assignment 3 Report

Anirban Das (dasa2@rpi.edu)

March 23, 2018

0.1 Graphs

Here, we can plot runtimes of both the `MPI_Reduce` and `MPI_P2P_reduce` collective operations as a function of the number of ranks. In both cases we assume the runtime equal to the sum of time required to do the local sum and the collective operations.

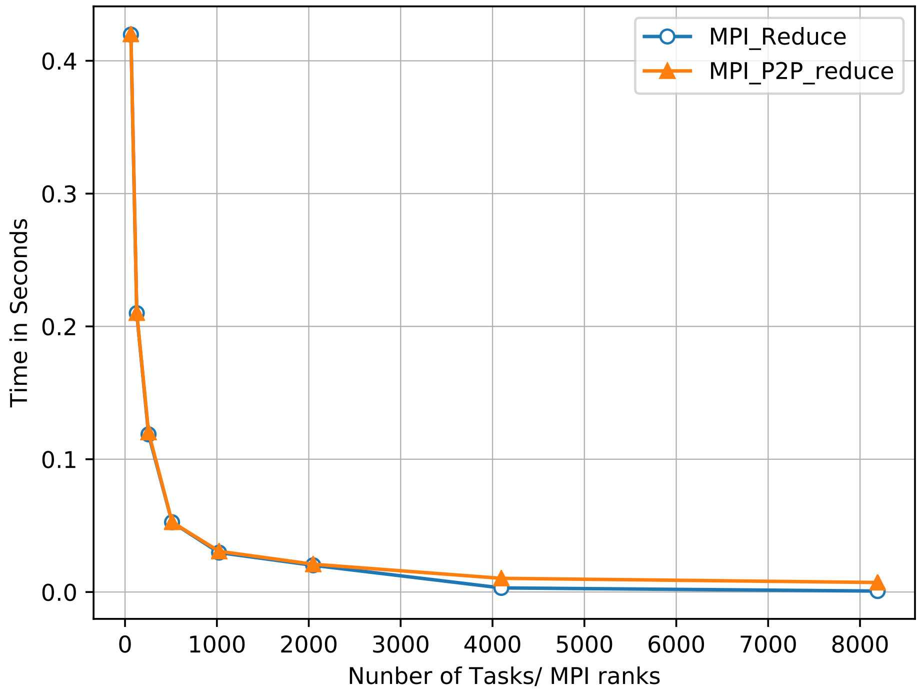

Figure 1: Plot the execution time of both reduction approaches as a function of the number of MPI ranks (Y axis in seconds)

The following is the same graph, but with a $\boldsymbol{\log_{10}(seconds)}$ Y axis, so that the difference between performance of `MPI_Reduce` and `MPI_P2P_reduce` is more prominent:

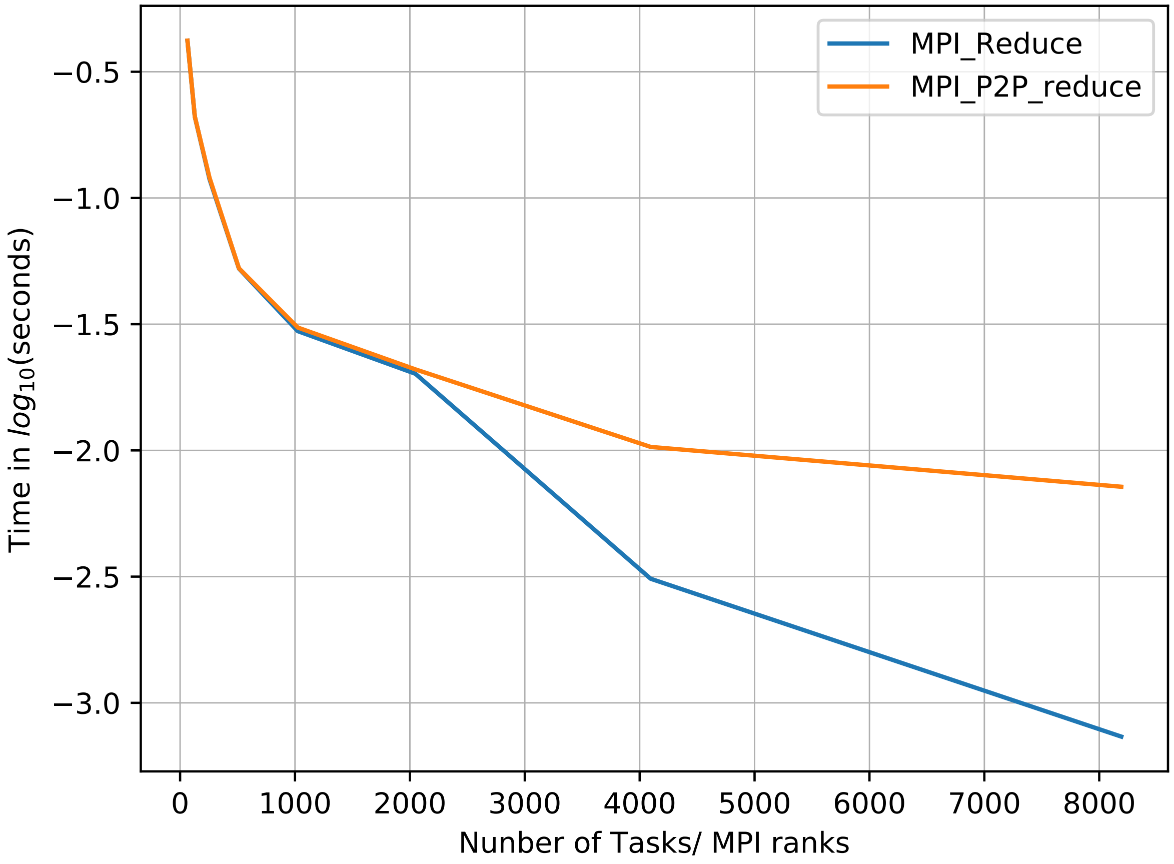

Figure 2: Plot the execution time of both reduction approaches as a function of the number of MPI ranks (Y axis in log10)

0.2 Observations

First of all, `MPI_Reduce` is always faster than `MPI_P2P_reduce` .

1. We observe that with increase in the number of ranks, the runtime goes down for both `MPI_Reduce` and `MPI_P2P_reduce` ,with the lowest for 8192 MPI ranks. This is partly because of two reasons :
   (a) With increase in number of ranks, our 1 billion long array is fragmented up in 8192 small subarrays in each rank. As the size of the array in each individual rank decreases it becomes possible to fit more and more of the array in the cache, resulting in very fast operation.
   (b) With increase in number of the MPI ranks participating, the array gets divided into smaller pieces and each rank has to compute only a fraction of the task now. As a result the computation time per rank decreases, resulting in overall decreasing computation time.

   Now the highest we go is for 8192 ranks which is 128 nodes which is still within one rank.

So the communication overhead over computation overhead is minimum, we are probably in a single midleplane in a single rack.

2. It is also observed that runtime of `MPI_Reduce` is consistently lower than that of `MPI_P2P_reduce` for all the ranks. Though in both `MPI_Reduce` and `MPI_P2P_reduce` uses same number of point to point communications, in the former the communication is mainly between all the other ranks and rank 0. Whereas in `MPI_P2P_reduce` there is a lot of inter communication between several ranks which may be situated in different parts of the network causing little bit more delay.

   Also this disparity in performance is caused by the fact that `MPI_Reduce` is far more optimized for collective operations than our native implementation `MPI_P2P_reduce` using point to point message transfer `MPI_ISend` and `MPI_IRecv`.

3. The difference in runtime of `MPI_Reduce` and `MPI_P2P_reduce` is however is very very small in all cases. But if we look at the log scale, it is noted that as the number of ranks increases, the difference between the performance of `MPI_Reduce` and `MPI_P2P_reduce` increases. This is probably because of our binary tree sort of implementation is not efficient with large ranks maybe because of os jitter or some latency.