Report (Assignment 2 – CS633)

-Yuvraj (180898), Vishwas Choudhary (180876)

1. Steps to run the code

- Run the job script using python3 run.py to generate all the data_{X}.txt files
- Run the plot script using python3 plot.py to generate all the plot_{X}.jpg files
- Requirements : Pandas, Seaborn, Numpy, Matplotlib
- If you wish to run the code on a customized use-case, follow the below steps :
  1. mpicc src.c -o collectives
  2. mpirun -np 64 -f hostfile ./collectives 1024 8 3 3 2

     Notes :

     a) The arguments pattern of the executable is :

     ./collectives D (data size in KB) ppn (processes per node) number_of_nodes_in_group_1 number_of_nodes_in_group_2 ...

     The above command, when run, the code will assume that D=1024KB, ppn=8 and the hostfile contains first 3 nodes from one group, the next 3 nodes from another group, the next 2 nodes from another group (3+3+2=8 total nodes, ppn=8 => total processes=64). Careful generation of the hostfile (keeping the groups in mind) and passing of correct arguments is the responsibility of the user if he/she decides to pass customized arguments

     b) Each node in the hostfile must be followed by ":ppn", the same number passed as one of the arguments, since the code assumes "hostwise" placement of ranks

2. Code explanation

- **run.py** : Runs the make clean command, followed by make command. The algorithm for hostfile generation is "greedy" in the sense that it uses "nodefile.txt" to find the first P available nodes and runs the final executable keeping in mind the group of every node
- **src.c** : The major functions involved are as follows :
  - standard_bcast, optimized_bcast
  - standard_reduce, optimized_reduce
  - standard_gather, optimized_gather
  - standard_alltoallv, optimized_alltoallv
  - binomial_gather_rank_reorder : reorders existing ranks to incorporate topological awareness in the gather call (assuming that gather uses binomial tree algorithm)
  - Rest of the explanation is present within the code itself in the form of comments

3. Optimizations and reasoning

- **Bcast** : The concept of master ranks (hierarchical broadcasting) has been used, which has 3 levels (inter_group, inter_node, intra_node).

  Inter group communicator : The root node broadcasts the message to the "group level" master ranks within every group (one per group). This step happens only when there are multiple groups

Inter node communicator : The “group level” master ranks now broadcast the message to “node level” master ranks (one process of every node within their group). This step happens only when nodes in the concerned group > 1
Intra node communicator : The “node level” master ranks now broadcast the message to every process within the node. This step happens only when ppn > 1

Other minor optimizations include using non-blocking bcast calls and using MPI_Comm_create_group to create multi-level communicators instead of MPI_Comm_split, which proved to be a very expensive call
This optimization has been implemented since a topologically unaware bcast would not try to minimize high latency communications but our algorithm implements the optimal solution in the sense that it does minimal and only required inter group and inter node communications thus minimizing the latency

- **Reduce** : The algorithm implemented is the same as the one implemented to optimize Bcast, but in reverse order (since Reduce essentially collects the data to root node along with some processing). Again, non blocking calls and MPI_Comm_create_group have been used. The reason for implementing this particular optimization technique is similar to that of Bcast

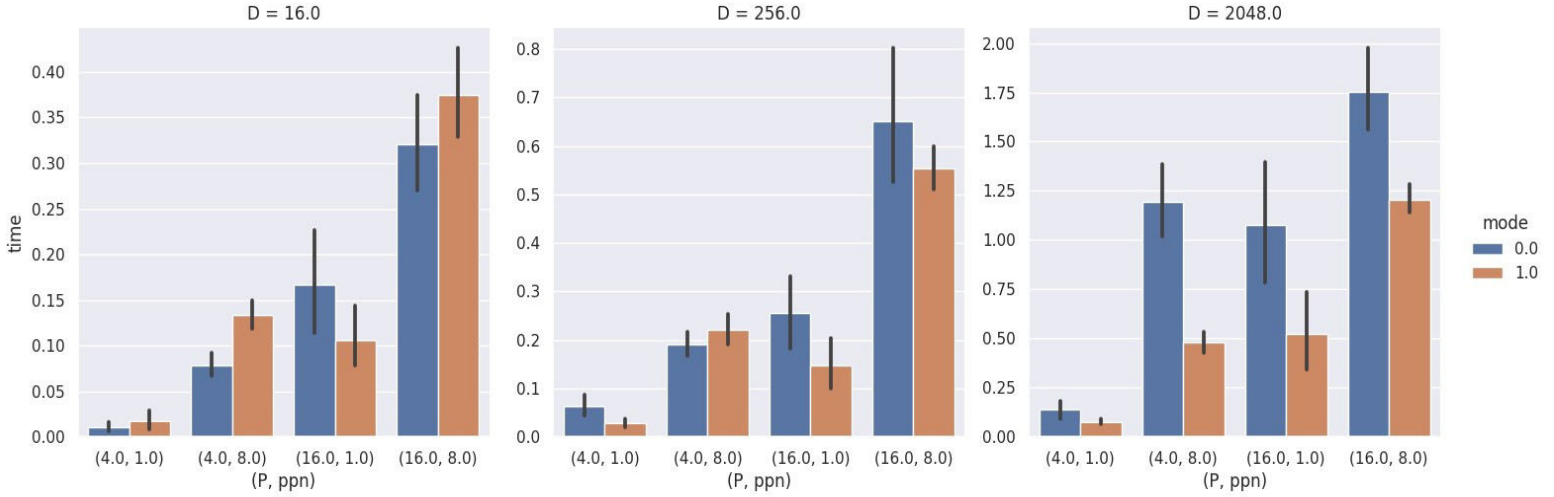
- **Gather** : The technique used for previous two calls was tried, however very poor results showed up. This was expected since MPICH uses binomial tree algorithm to implement gather and as we go up the tree, data size keeps on increasing by a factor of 2 (unlike bcast and reduce, where data size remains constant throughout). Hence using the same technique we would get suboptimal results as the highest amount of data would then be communicated at inter group level, which is not desirable

  Thus, we used a topologically aware rank reordering heuristic (assuming binomial tree implementation for Gather) which essentially reorders existing ranks based on the communication weights between different ranks.

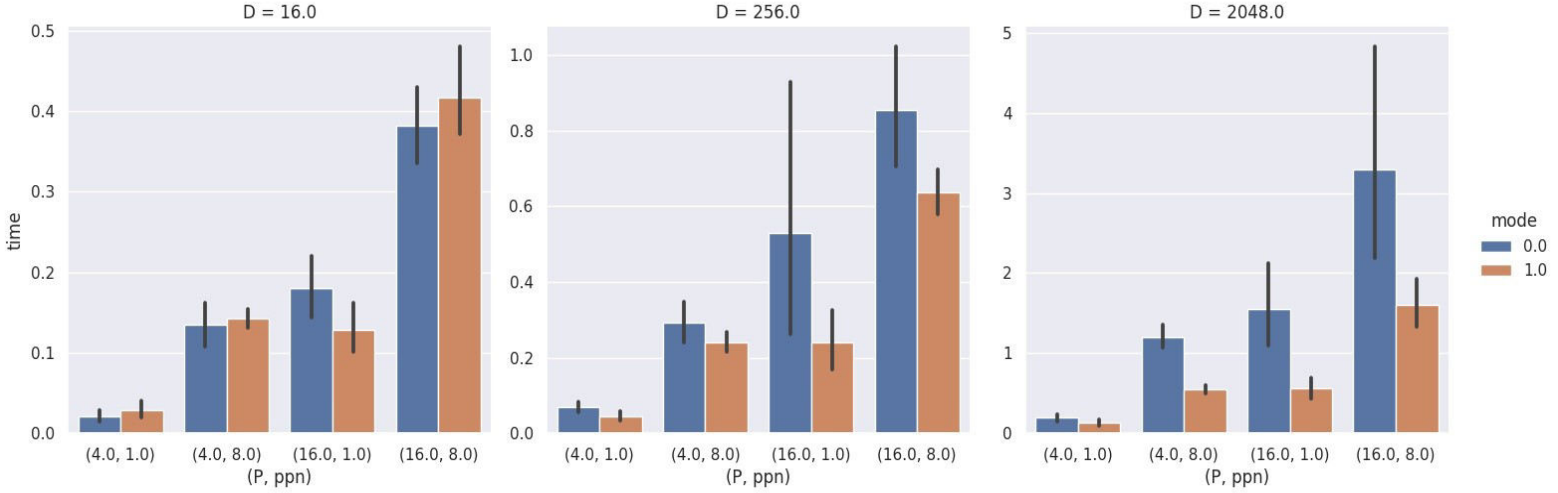
- **Alltoallv** : This algorithm was the hardest to improve since the underlying logic is simply composed of isends and irecvs and rightly so, since every process sends unique data to every other process, so there must be a communication between them either directly or indirectly (but this would cause more latency as data increases, as we shall see in observations)
  Thus we tried to improve the algorithm by issuing multiple iscatterv calls (total being equal to the number of processes). Non blocking call has been used to ensure there is no delay due to any one call blocking. However for large data, standard alltoallv is used because scatterv showed slight improvements that too only with small data.
  An optimized version of scatterv was written (using rank reordering heuristic), but it worsened the results, so was neglected midway and standard iscatterv is used now

4. Plots (mode 0 : standard call, mode 1 : optimized call)

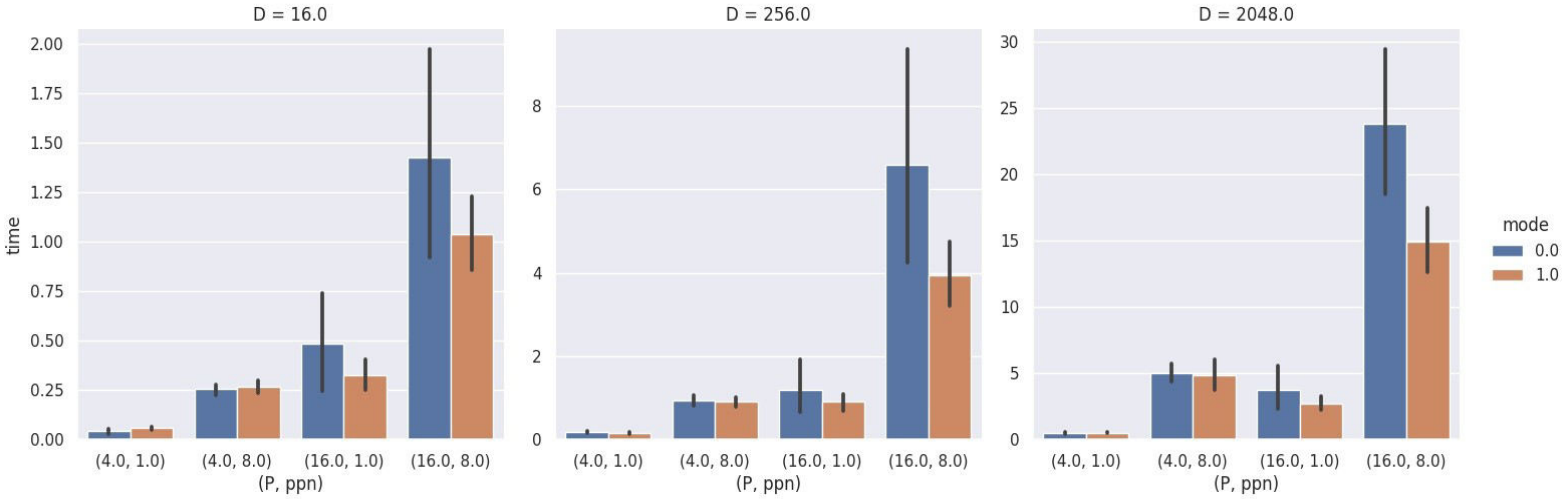
Bcast



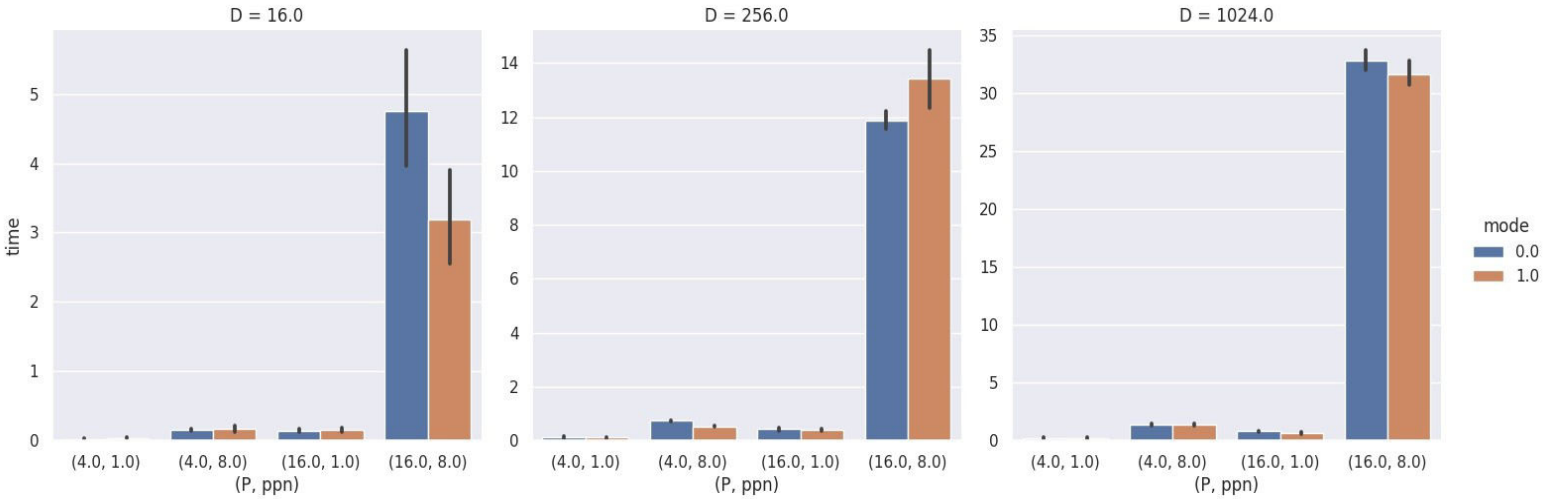
Reduce



Gather



Alltoallv



5. Observations

- **Bcast** : Optimized bcast performs considerably better than standard bcast as the data size goes up, thus minimizing high latency communications is clearly helping significantly. Worse performance is observed when data size is low AND ppn is high. This can be explained by mentioning the fact that process mapping is done hostwise, which is already close to optimal for some of the algorithms employed by bcast in this data range, thus reducing the time gap in the case of high ppn. For very low data size, the overhead of multiple bcasts overtakes the latency caused by topological unawareness
- **Reduce** : Similar effect can be seen in reduce, but the optimization is higher in magnitude than bcast, owing to the fact that reduce uses recursive distance doubling algorithm which makes our multi level communication optimize the time (as opposed to the binomial tree algorithm used by bcast in some cases, which is already close to optimal for existing rank ordering). Again, the times are comparable in case of low data due to the reasons same as bcast
- **Gather** : Optimization can be observed in almost all the cases. However, it is more pronounced in the case of higher number of nodes and processes. This is because of the fact that for same D, with higher number of processes the size of data transfer increases “more” in the later stages of binomial gathering. In the case of optimized gather, such communication-heavy ranks are placed nearby, thus the speed gain. However, for lower number of processes, the overhead due to reordering (and communicator generation) appears more
- **Alltoallv** : For large data, results are comparable since the same call is used in this case. The performance of scatterv worsens as data size increases (16 to 256) because scatterv uses binomial tree algorithm which works well with short data size, hence shows comparable or even better performance when D<1024 but as the data size increases, the latency due to binomial transfers overtakes the simple isend-irecv combinations implemented by alltoallv

6. Additional comments

- A few important to mention assumptions :
  a) The code is written and optimized assuming that “hostwise” placement of ranks is done i.e. the format of every line in hostfile is “node_name : ppn”. Anything else could also have been assumed
  b) Without loss of generality, rank 0 process (in comm_world) is assumed to be the root process in the relevant calls
  c) “nodefile.txt” has been edited (reordered and limited to 5 nodes per group) to ensure a good number of groups in the hostfile, which would reinforce the impact of optimized algorithms
  d) Alltoallv has been tested for D=1024KB in place of D=2048KB since some nodes resulted in seg-fault in the case when (P, ppn) = (16, 8) probably because every node in this case has to allocate nearly 4GB of memory which sometimes throws error
- A few issues faced :
  a) During the algorithm design and improvement phase, the testing on nodes gave highly variable results (especially for larger data sizes), which made it very hard to proceed in a particular direction and conclude about the performance of any algorithm
  b) Testing took very long time and to add to the woes, the connection would frequently drop hence resulting in loss of progress
  c) The assignment was more like a mini-research project and consumed too much of time

7. References

- S. H. Mirsadeghi and A. Afsahi, "Topology-Aware Rank Reordering for MPI Collectives," 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Chicago, IL, USA, 2016, pp. 1759-1768, doi: 10.1109/IPDPSW.2016.139.