Assignment 2

Aniket Sanghi (170110)
Sarthak Singhal (170635)

28th March, 2021

We pledge on our honor that we have not given or received any unauthorized assistance.

1 How to Run

```
>>> ./run.sh
```

2 How to make plots

```
>>> python3 plot.py data <construct_comms>
```

where *construct_comms* = 1 if time for constructing communicators is to be taken into account (for the optimized case) else 0.

3 Algorithmic Explanation

Network Topology
Exploited the *Simple Tree* topology of the CSE lab servers network. Avoided "4 hops" where ever possible.

Sub-communicators setup

1. *intra_node_comm* A sub-communicator for each node for communication between processes within a node.
2. *intra_grp_node_leaders_comm* A sub-communicator for each network group for communication between leaders of every node. Processes with rank=0 in intra-node-comm are the leaders here.
3. *grp_leaders_comm* A sub-communicator for communication between leaders of every network group. The process with rank 0 in "intra-grp-node-leaders-comm" is made leader in every network group.
4. IMPORTANT NOTE: Communicators are such that the **root** (if present) will always be the process with *rank = 0* in all the above communicators.

MPI_Bcast

Algorithm

1. Broadcast to all Network-group-leaders from the root. It will require 4 hops for each p2p communication.
2. In each network group, the leader broadcasts the data to all its nodes, to the node-leader. Here 2 hops will be required for each p2p.
3. In each node, the node-leader broadcasts the data to all its processes. Here since all communication is intra-node, hence much efficient as MPI uses shared memory to optimise.

MPI_Reduce

Algorithm

1. Perform reduction for every process within each node and deliver the result to the node-leader.
2. Reduce the data stored with all node-leaders within each network group and deliver it to the network-group-leader.
3. Finally reduce the data stored with each network-group-leader and deliver it to the root.

MPI_Gather

Algorithm

1. Every process sends "given_size + 1" data instead of "given_size" as it sends the rank of the process in MPI_COMM_WORLD as the last element of the array. This is used in order to rearrange data in rank-order once data is collected in root.
2. Firstly, every node-leader gathers data from each of the process of the node in a contiguous array.
3. Then, every network-group leader gathers data from each of the node-leaders.
4. Then finally the root gathers data from each of the network-group-leader

MPI_Alltoallv

Algorithm 1

1. Firstly, collect everyone's send-count and displs array at the root i.e. two 2D array collected at root
2. Secondly, collect data to be sent for each process at the root using optimised gatherv which is similar to the gather algorithm above that uses network topology, thus giving us a 2D data matrix
3. Then the root scatters the data using the above information (roughly each column of 2D matrix) to all the processes using the default Scatterv offered by MPI

Algorithm 2

1. For every process call optimised gatherv in a loop (iterations = number of processes) i.e. each process becomes root once. Works better with larger data-size.

Optimisation Explanation

1. Firstly the optimisation reduces the number of network communication with 4 hops p2p which saves us from the network congestion in top links of tree.
2. With better intra-code communication in the optimised algorithm we better utilise the use of intra-node-communication-optimisation of shared memory done by MPI

Issues faced in practical testing

1. The formation of communicators take extremely high time as compared to just the run of broadcast.
2. We got appreciable improvement in time if the time for creation of communicators is not included.
3. The best solution is to avoid using communicators and instead write the algorithm using send/recv from scratch avoiding the use of communicators.

4 Plots

4.1 Plots including time of constructing communicators(for optimized case)

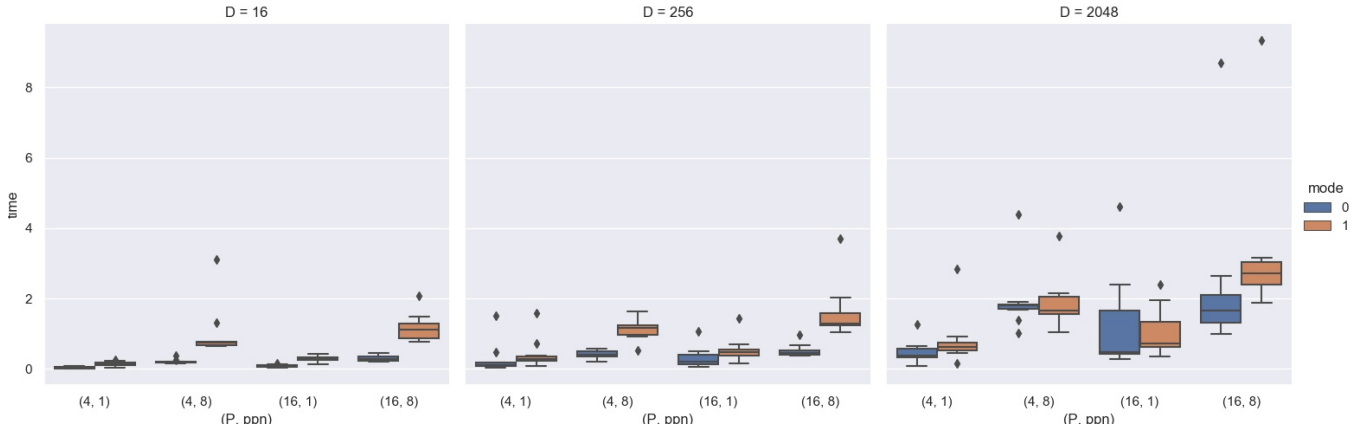

Figure 1: Bcast

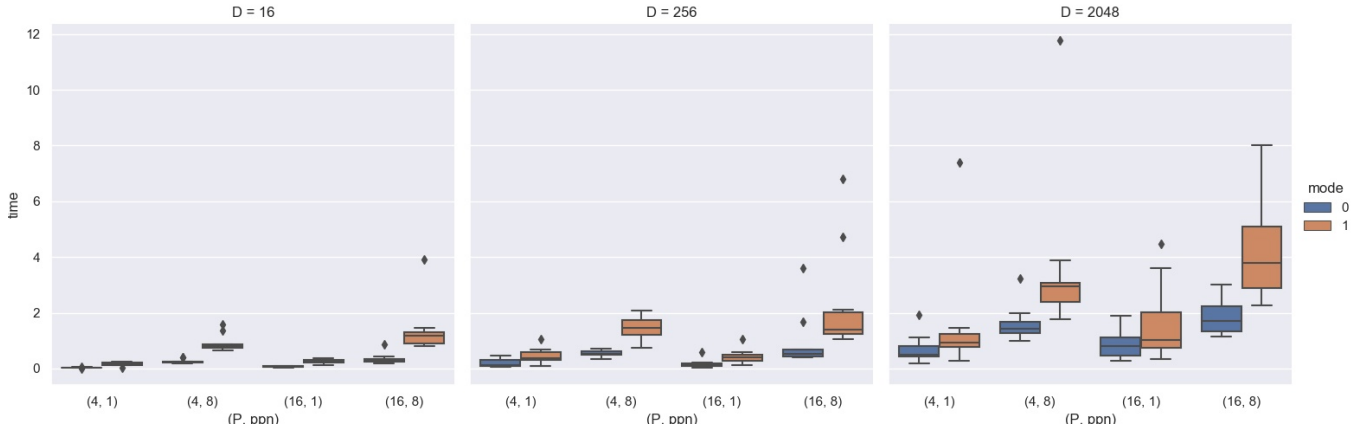

Figure 2: Reduce

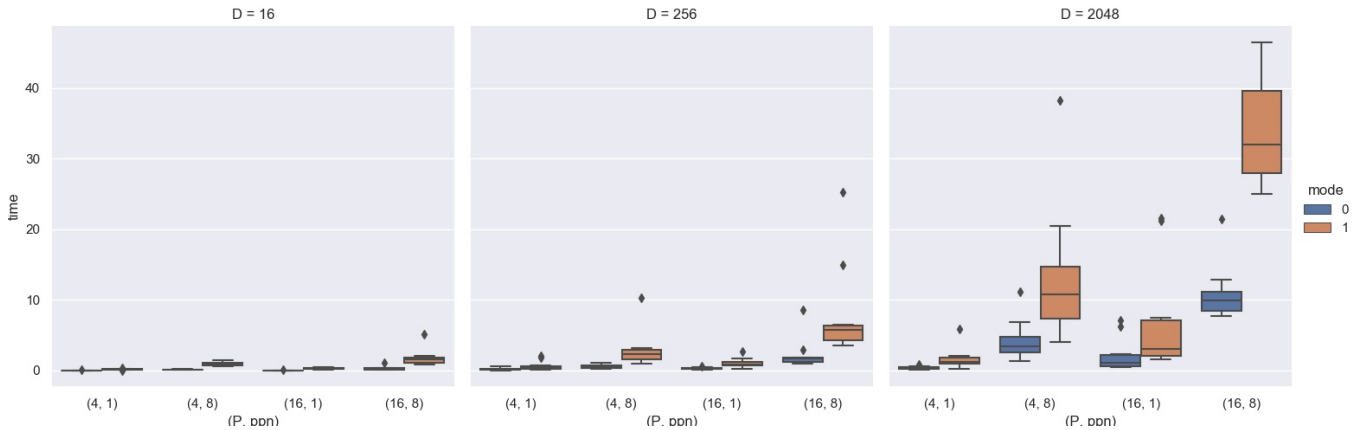

Figure 3: Gather

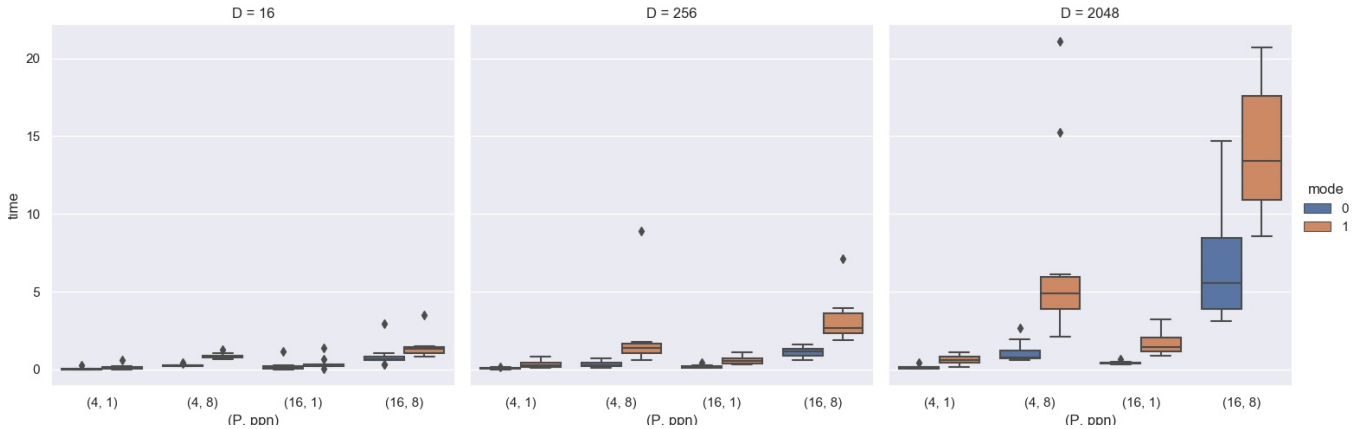

Figure 4: Alltoallv

4.2 Plots excluding time of constructing communicators(for optimized case)

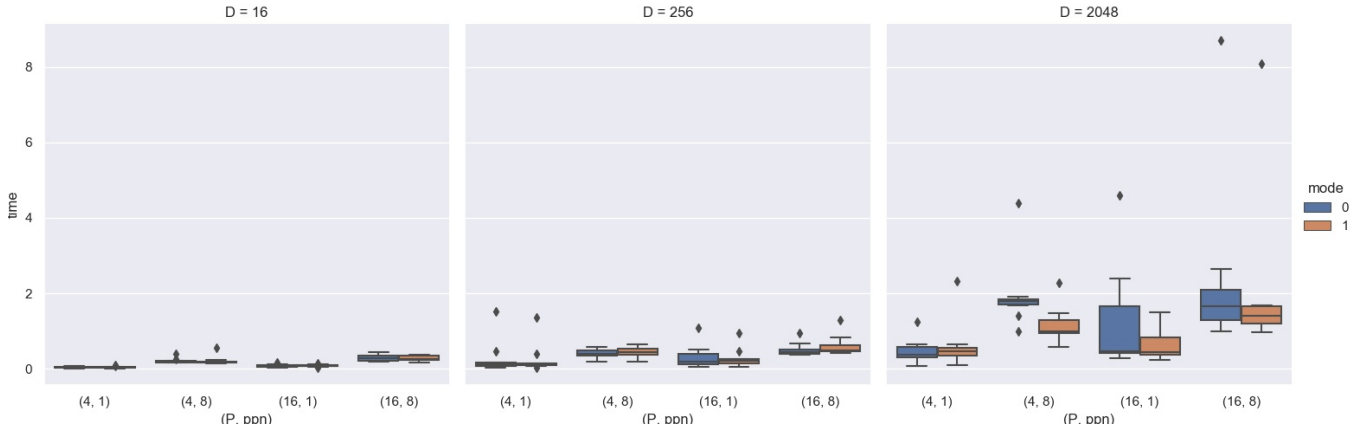

Figure 5: Bcast

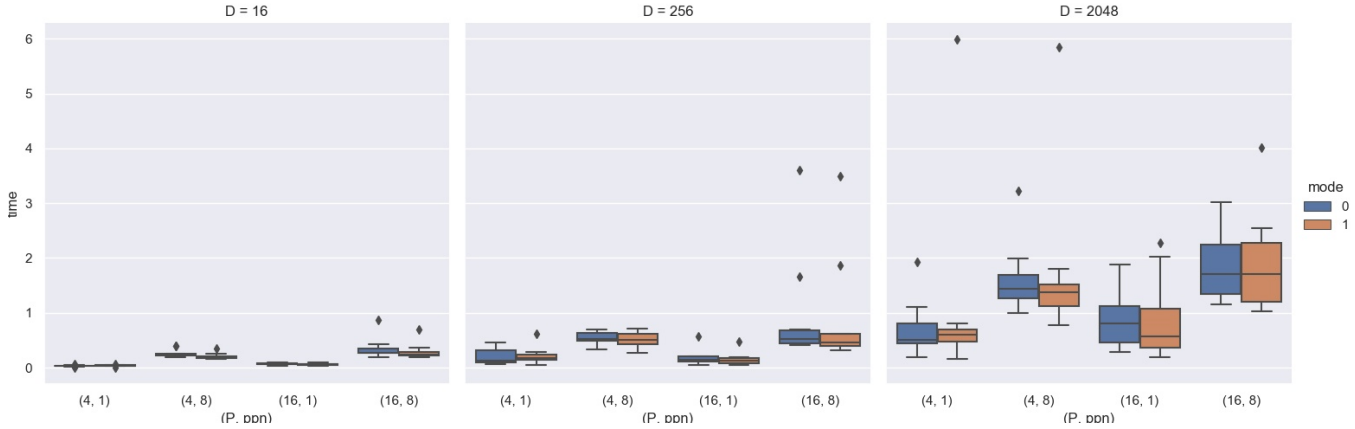

Figure 6: Reduce

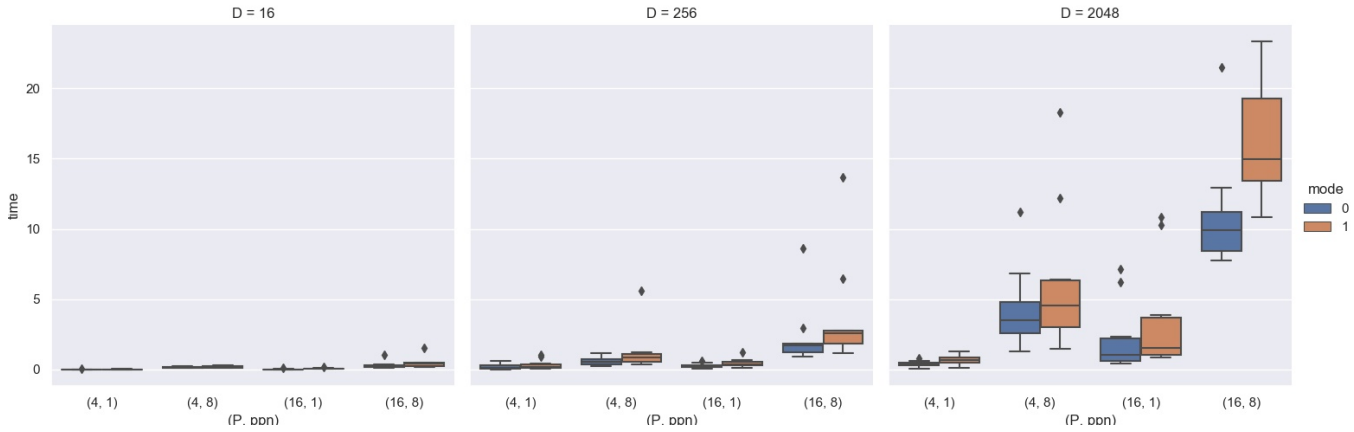

Figure 7: Gather

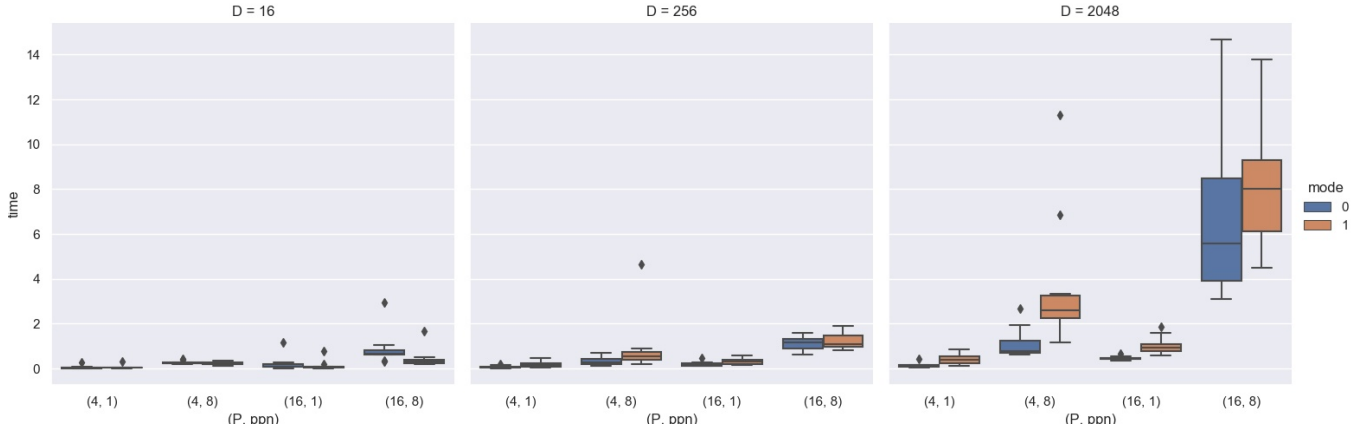

Figure 8: Alltoallv

4.3 Performance from plots

- It can be observed from the above plots that constructing the communicators add significant overhead. Due to which the optimized cases perform worse than default causes. Although many of them were better without considering the time for constructing communicators. This happens because *MPI_Comm_Split* internally uses *MPI_Allgather* which is itself time consuming.
- The optimized versions of Bcast and Reduce perform better than the default version if we ignore the time taken in constructing the communicators.
- Even for the case when we include time for creating the communicators it can be observed the plots for default and optimized cases overlap most of the times. This shows that atleast for a few times our optimized version performs similar to or even better than the default versions.