README

How to execute the Code

The Assignment2 directory contains the following files:

1. src.c
2. run.sh
3. Makefile
4. plot.py
5. script.py
6. nodefile.txt
7. src_opt1.c (one additional optimization we attempted)

The whole assignment can be executed by the script file 'run.sh'. For executing the code, make sure that the path to 'mpich' is pointing to user's installation directory. Use the following commands to execute:

$ chmod 777 run.sh

$ ./run.sh

The 'run.sh' script will create output file named 'data.txt' which will be used by the 'plot.py' file to generate the plots named 'Plot_{X}.jpg' for X in {Bcast, Reduce, Gather, Alltoallv}.

To generate the plots, run the following command.

$ python3 plot.py

Note: as there was some seaborn version mismatch on csews machines, so we generated plots on the local system.

If it is required to execute the code again, use following command before running again.

$ make clean

Explanation of the code

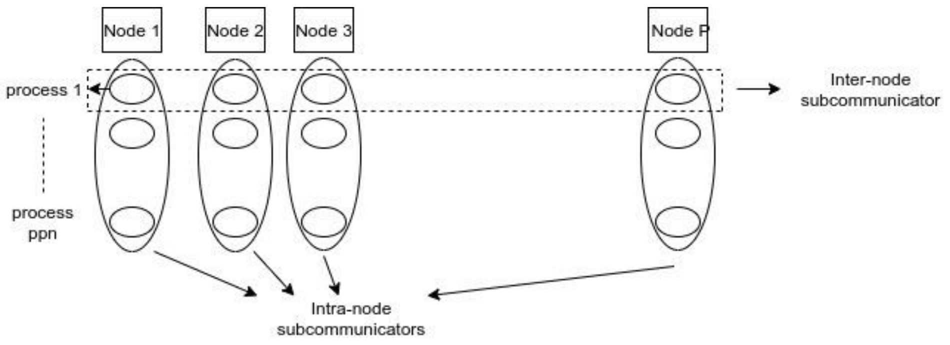
- **'src.c'** is the main code file which implements the optimizations performed for the given four collectives. It accepts **D** and **P** as arguments, where **D** is the data size and **P** is the number of processes. For each collective first we have executed the default (standard) call and took the average of execution time over five calls. Then we performed the optimization and executed the optimized collective five times to obtain the average execution time. Then we find the maximum time for both default and optimized versions over all the processes using MPI_Reduce function, and output it.
- There is a **'Makefile'** in the Assignment2 directory, which compiles the **src.c**.
- To generate the hostfile on the fly, we used 'script.py' given in the 'helper_scripts' folder on piazza.
- To generate the output plots, We used 'plot.py' provided in the 'helper_scripts' folder on piazza with some modifications.

Optimization Performed

We performed two different optimizations.

1. **Topology Unaware optimization (opt_1)**

- First we didn't made use of the physical topology of the 'csews' cluster and the information that the nodes are divided in the groups. Intra-group communication is more efficient compared to inter-group communications. (This information is given in the assignment2 text)
- We thought of grouping all the collective calls which are being made from the same node. We used the concept of sub-communicators for this. We created two sub-communicators for the first optimization. One sub-communicator (intra-node) is used for all the processes at each node and other sub-communicator (inter-node) is used for all the node leaders. In each of the node, the process with lowest rank is treated as node-leader, i.e., it will be the only process to handle all the communication for this node with other nodes.



a. **Optimization for MPI_Bcast**

In the default MPI_Bcast, one process (root, generally rank 0) sends same data to all the other processes. This requires lot of different inter-node communication. Inter-node communication is expensive as compared to intra-node communication, so we can try to reduce the inter-node communication for optimization.

To optimize MPI_Bcast, we first used inter-node sub communicator. In this one root process (rank 0) will broadcast its data to each node-leader. Then all the node leaders will broadcast this data to all the 'ppn' processes residing in their node using intra-node sub communicator.

b. **Optimization for MPI_Reduce**

To optimize MPI_Reduce, we first performed MPI_Reduce within each node using intra-node sub communicator. After this each node-leader will have reduced data of all the 'ppn' processes from their node. Then we used inter-node sub communicator to perform MPI_Reduce across all node-leaders. After this root process (rank 0, globally) will have the reduced data from all the processes.

c. **Optimization for MPI_Gather**

To optimize MPI_Gather, we first performed MPI_Gather within each node using intra-node sub communicator. After this each node-leader will have data of all the 'ppn' processes from their node. Then we used inter-node sub communicator to perform MPI_Gather across all node-leaders. After this root process (rank 0) will have all the data from all the processes.

d. **Optimization for MPI_Alltoallv**

To optimize MPI_Alltoallv, we first performed MPI_Gather within each node using intra-node subcommunicator. After this each node-leader will have data of all the 'ppn' processes from their node. Then we performed MPI_Allgather across node leaders using inter-node sub communicator.

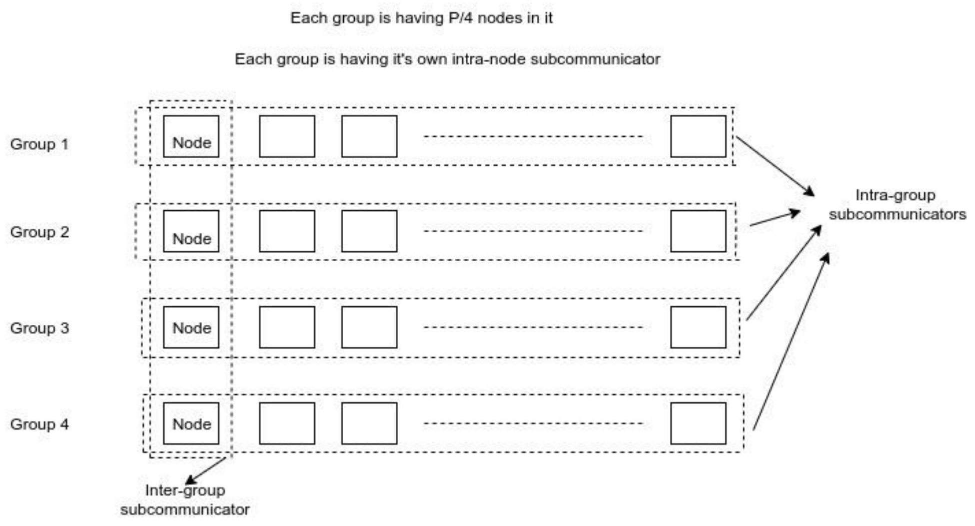
After this each node-leader will have the data from all the processes. Then we performed MPI_Bcast in each node using the intra-node sub communicator. After this each process will have data from each of the other processes.

2. Topology aware optimization

For the next optimization, we also considered the physical topology of the 'csews' cluster and considered the group-wise placement of the nodes. It is given that intra-group communication is faster than the inter-group communication.

For the hostfile generation, we are using 'script.py' file. In that we are passing argument "no_of_groups" as 4, I.e., we will always consider 4 groups and from each group we will select P/4 nodes.

This time we are creating 3 sub-communicators. First, we created intra-node sub communicator at each node, comprising all the process at each node. Then we created intra-group sub-communicator comprising node-leaders for each node present in the group (I.e., P/4 node leaders in a group). Then we created inter-group sub communicator comprising one group-leader from each of the group. All leaders are selected as the lowest ranked process in that sub communicator.


Each group is having P/4 nodes in it
Each group is having it's own intra-node subcommunicator
Group 1
Group 2
Group 3
Group 4
Node
Node
Node
Node
Intra-group subcommunicators
Inter-group subcommunicator

e. Optimization for MPI_Bcast

To optimize MPI_Bcast, we first used inter-group sub communicator. In this one root process (rank 0) will broadcast its data to each group-leader. Then all the group leaders will broadcast this data to all the node-leaders in their group using intra-group sub communicator. Then each node-leader will use intra-node sub communicator to broadcast this data to each of the process in their respective node.

f. Optimization for MPI_Reduce

To optimize MPI_Reduce, we first performed MPI_Reduce within each node using intra-node sub communicator. After this each node-leader will have reduced data of all the 'ppn' processes from their node. Then we used intra-group sub communicator to perform MPI_Reduce across all node-leaders in each of the group. Then we used inter-group sub communicator to perform MPI_Reduce across all the group leaders. After this root process (rank 0, globally) will have the reduced data from all the processes.

g. Optimization for MPI_Gather

To optimize MPI_Gather, we first performed MPI_Gather within each node using intra-node sub communicator. After this each node-leader will have data of all the 'ppn' processes from their node. Then we used intra-group sub communicator to perform MPI_Gather across all node-leaders in each of the group. Then we used inter-group sub communicator to perform MPI_Gather across all the group leaders. After this root process (rank 0, globally) will have the data from all the processes.

h. **Optimization for MPI_Alltoallv**

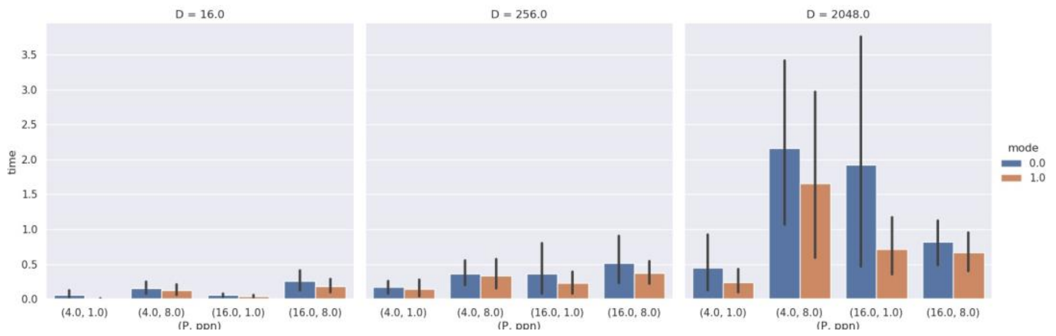
To optimize MPI_Alltoallv, we first performed MPI_Gather within each node using intra-node sub communicator. After this each node-leader will have data of all the 'ppn' processes from their node. Then we performed MPI_Gather across all the node-leaders within a group using intra-group sub communicator. Now each group-leader will have all the data from all processes in the group. Then we performed MPI_Allgather across all the group leaders using inter-group sub communicator. Then we perform MPI_Bcast in each group to broadcast this data to each node-leader in the group. Finally, we performed MPI_Bcast in each node to broadcast this data to all the processes in their node. After this each process will have data from each of the other processes.

Observations regarding Optimization:

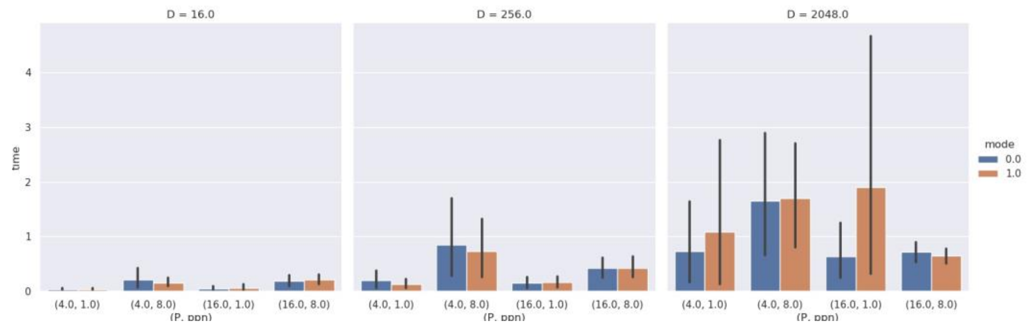
I) Topology unaware optimization:

Note: For all the figures below, mode = 0 implies unoptimized implementation of collectives and mode = 1 implies optimized implementation of collectives

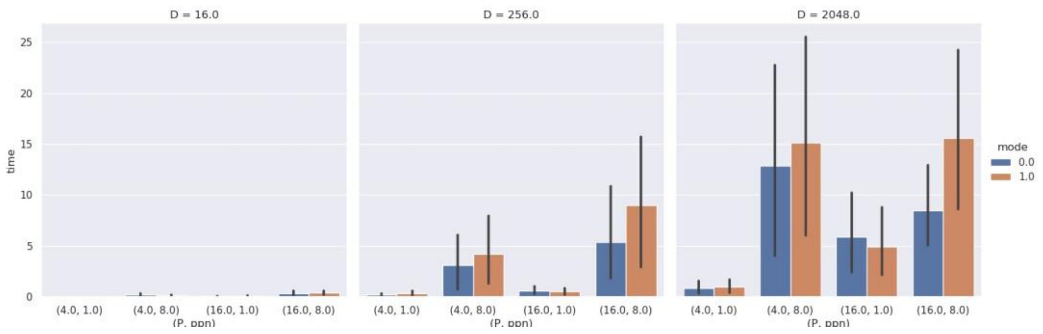
a) Optimization for MPI_Bcast:



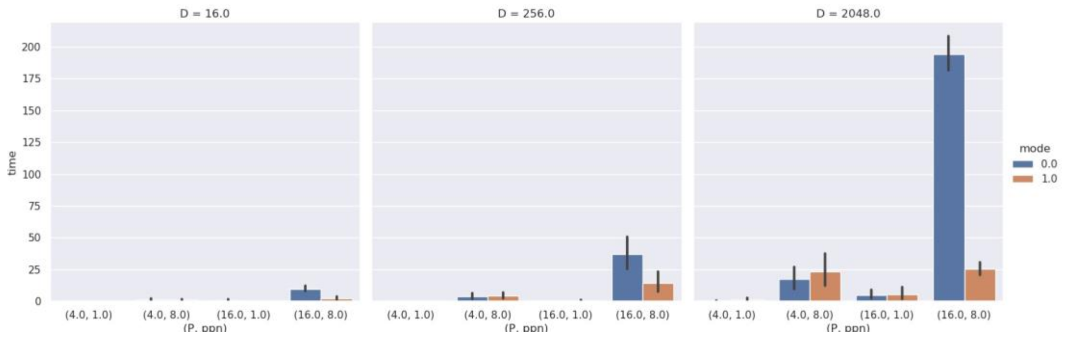
b) Optimization for MPI_Reduce:



c) Optimization for MPI_Gather:



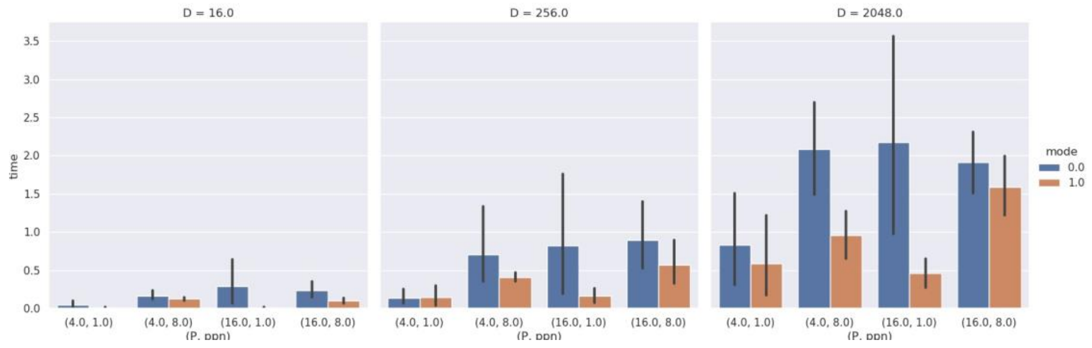
d) Optimization for MPI_Alltoallv:



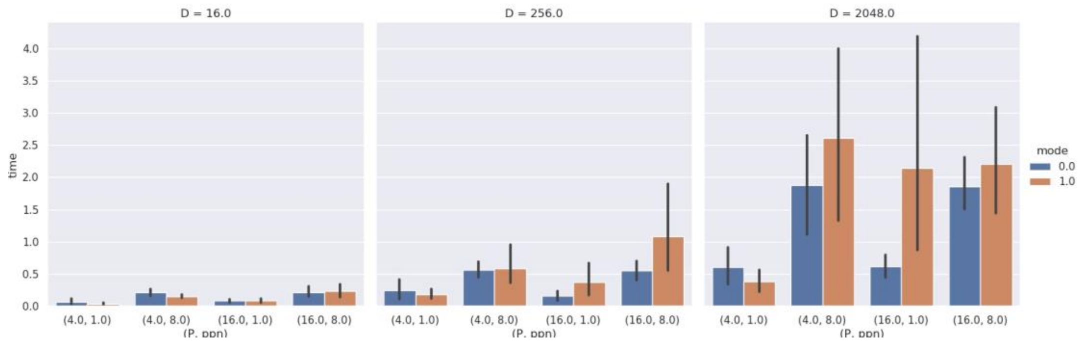
II) Topology aware optimization:

Note: For all the figures below, mode = 0 implies unoptimized implementation of collectives and mode = 1 implies optimized implementation of collectives

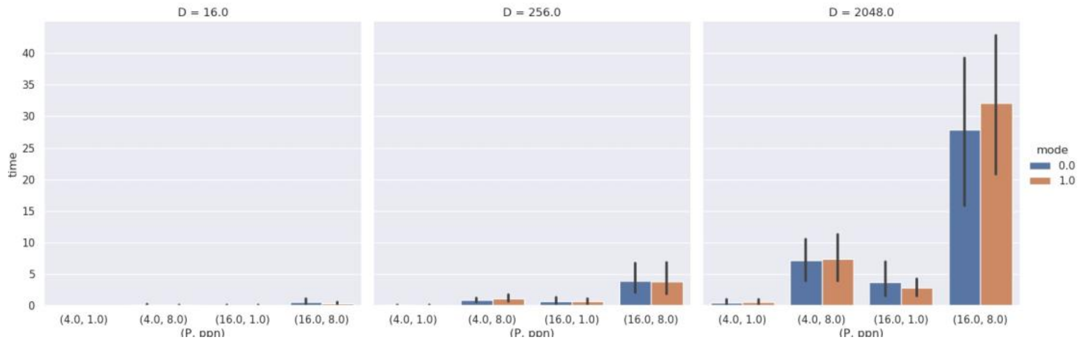
e) Optimization for MPI_Bcast:



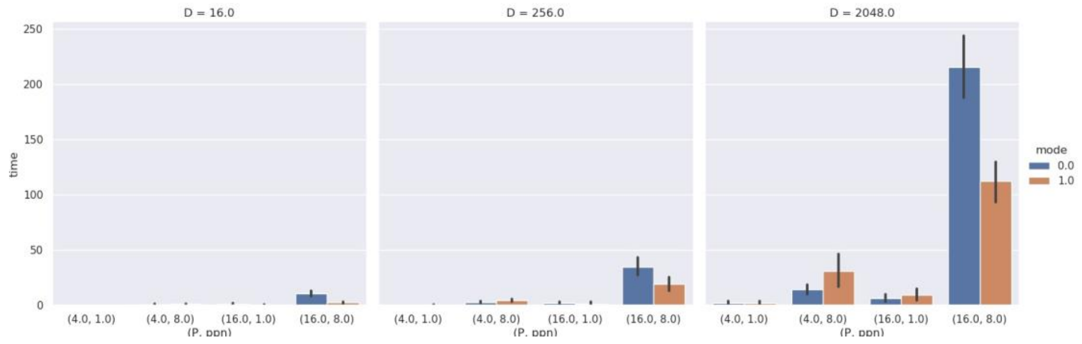
f) Optimization for MPI_Reduce:



g) Optimization for MPI_Gather:



h) Optimization for MPI_Alltoallv:



Our observations:

Both optimizations are performing better than the default implementation. We can also observe that topology aware optimization is performing better than topology unaware optimization. This is as expected because we tried to reduce inter-group communication which is very expensive.

We also observed that for larger data sizes, the execution time difference between default and optimized implementation is more and optimized implementation is working much better.

For MPI_Bcast, we are able to achieve better results in all the cases. For MPI_Reduce the time taken by default and optimized versions are almost similar. For MPI_Gather in some cases we are able to get better results and in some cases default version was performing better. This may be due to congestion in the network. For MPI_Alltoallv we are able to get better results than the default version.

Out of the two optimizations performed in general topology aware optimization is working better, so we submitted code 'src.c' and plots regarding this as final submission. We have included code related to topology unaware optimization also in the file named 'src_opt1.c' which can be used to generate the above results.

Additional optimization

- On observing the given ‘script.py’, we found that it allocates nodes from starting as given in the ‘nodefile.txt’. Generally, starting nodes are overloaded most of the time, so we moved first 2 groups in the ‘nodefile.txt’ to the last to speed up the execution. It worked as previously execution was taking much time.