Assignment 2, CS633

Saketh Maddamsetty 170612,
Ayush Tharwani, 170201

Directory - Assignment 2
Files

- src.c - source code
- Makefile
- run.py - helper code for execution and plotting
- Readme.pdf - this file
- plot.py

Non standard Libraries : *matplotlib,searborn,numpy,pandas* from python3

Steps to run the code:

1. *python3 run.py*
   make clean, followed by make to compile source, then generates host file using Node Allocator, runs the executable for 10 times for P=4,16 nodes for ppn = 1,8 cores per node for D (D=16,256,2048) size of arrays, outputs the (standard collective call average time, optimized collective call average time) for each (execution_number,P,ppn,D) type configuration in “output.{collective name}” and plots the barplots corresponding to each collective in ‘plot_{collective name}.jgp’.

Code Explanation

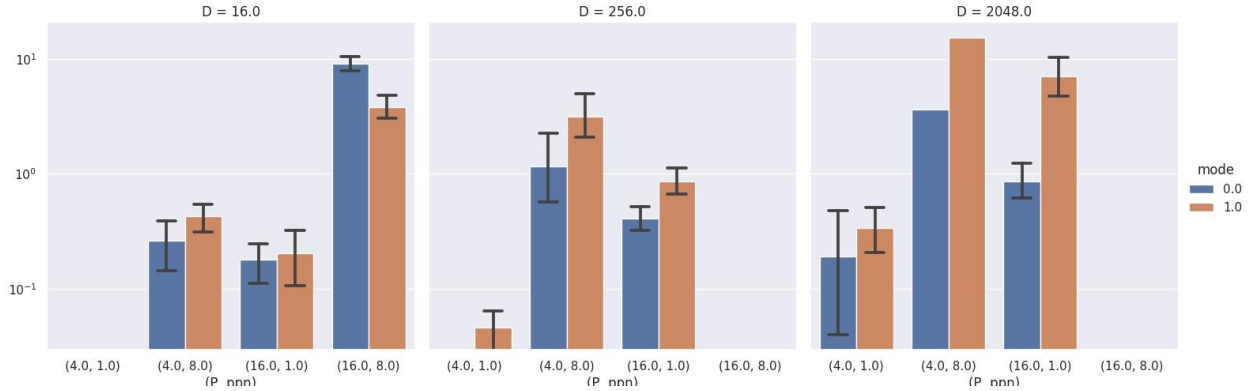
src.c

- **find_my_topology** function is central function that categorizes each process according to their group no in physical topology and assign it **global_id** so that topology aware communicators can be formed.
- Data initialisation is performed before collectives starts.
- **new_bcast** is collective bcast call where new communicator among all processes is formed such that processes running in same physical groups have contiguous ranks allotted to them.
- **new_bcast_alpha** is collective bcast call where processes among each group form a communicator using MPI_Comm_split and find_my_topology function, each group is assigned process with intra_comm_rank =0 as the leader process. All leader processes also form an communicator, it is ensured that root rank in the original communicator is a group leader rank. Firstly, message is broadcasted among group leaders. Then group leaders broadcast message among their group members.

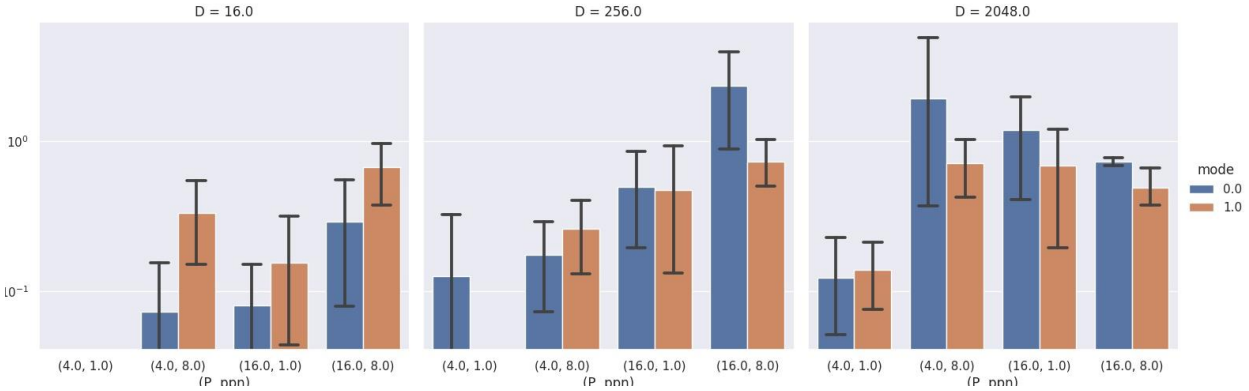
- **new_bcast_beta** is collective bcast call where processes among each group form an intra communicator using MPI_Comm_split and find_my_topology function, each group is assigned process with intra_comm_rank =0 as the leader process. All leader processes also form an inter communicator, in order to do so ranks of group leaders in original communicator are gathered at every process using Allgatherv, then using MPI_Intercomm_create, inter communicators are created between group with root and other groups. It is ensured that root rank in the original communicator is a group leader rank. The group with root as group leader broadcasts message in its group using intra communicator and broadcasts message in other groups using intercommunicator.
- Our main code will run **new_bcast_alpha** for construction of graphs as it was having consistent and low times. After understanding that alpha call for bcast performed better, we dropped the idea of inter communicators for other calls.
- **new_gather** is collective gather call where processes among each group form a communicator using MPI_Comm_split and find_my_topology function, each group is assigned process with intra_comm_rank =0 as the leader process. All leader processes also form an communicator, it is ensured that root rank in the original communicator is a group leader rank. All group leaders gather messages from their respective groups and then group leader who was also the root will gather these messages at root process. Finally, at root process, the combined array is reordered according to what original communicator would have received.
- **new_reduce** is collective reduce call where processes among each group form a communicator using MPI_Comm_split and find_my_topology function, each group is assigned process with intra_comm_rank =0 as the leader process. All leader processes also form an communicator, it is ensured that root rank in the original communicator is a group leader rank. All group leaders collect reduced messages from their respective groups and then group leader who was also the root will collect reduced message at root process.
- **new_alltoallv** is also optimised as all the above calls. It is done is three main steps. First data is gathered at leaders of their respective groups. Then the leaders perform an alltoallv call. Then data is scattered to the respective processes inside the groups. To achieve this metadata like sendcounts, recvcounts and new rank , old rank pairs are shared with the leaders.

Plots

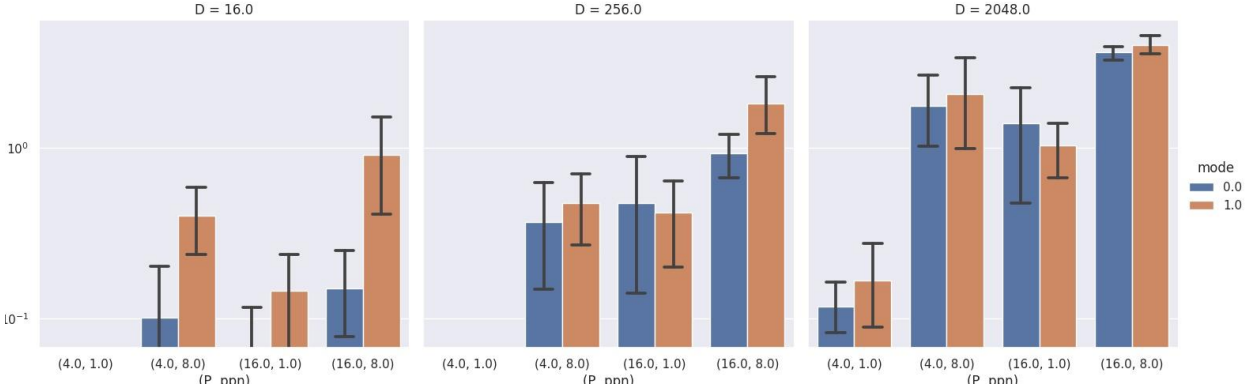
Alltoallv

- 

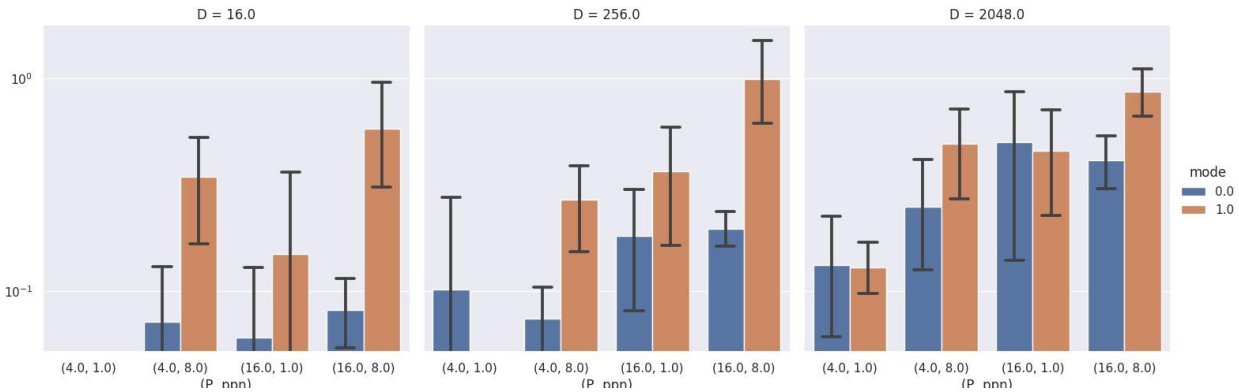
Bcast

- 

Gather

- 

Reduce

- 

Trends observed in the plots

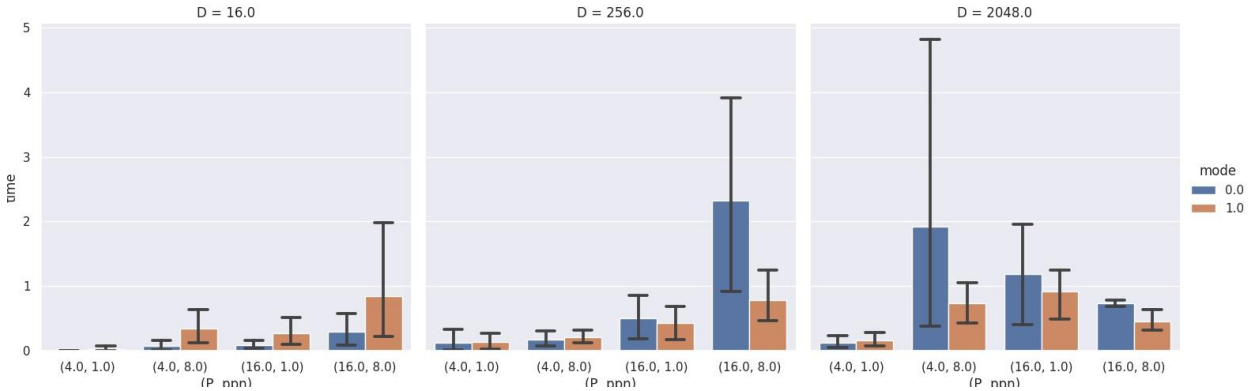
- Orange is for optimised code, Blue is for unoptimized code.
- For **Alltoallv**, the optimised code is performing slightly better for smaller size of data sent. But for larger size of data our optimised call does not give much performance benefit. This may be because of the fact that we internally used MPI collective calls to implement this and MPI collective calls employ different algorithms for different sizes of data which is not yielding performance improvement in some cases.
- For **Bcast**, the optimised code is performing better for medium and large sizes of data sent. But for smaller sizes of data our optimised call does not give much performance benefit. This is because of the fact that the time it takes for performing a comm split call is more than the total time for running unoptimised code. Removing this term and counting it only once might have given some performance improvement.
- For **Gather**, the optimised code is performing slightly better for medium and large sizes of data sent. But for smaller sizes of data our optimised call does not give much performance benefit. This is because of the fact that the time it takes for performing a comm split call is more than the total time for running unoptimised code. Removing this term and counting it only once might have given some performance improvement.
- For **Reduce**, the optimised code is performing slightly better for large sizes of data sent. But for smaller and medium sizes of data our optimised call does not give much performance benefit. This is because of the fact that the time it takes for performing a comm split call is more than the total time for running unoptimised code. Removing this term and counting it only once might have given some performance improvement.
- On average, the time taken to run Alltoallv call is significantly larger than other calls

Issues faced

- make command leading to clock skew, sometimes interrupt with mpiexec call
- Code takes approx 2hr - 4hr from compilation to plotting
- The hostfile generated by script.py was not working hence we used the nodeallocator to generate them.
- Since there are only 3 active groups,
- The version optimised code using intercommunicators calls was not benefiting from performance improvement which is contradicting what we read in theory.
- Debugging the alltoallv took a lot of time since it needed synchronising data(rearranging data inside buffer) at each individual step inside the optimised code.
- The results from timing our codes were very random and were varying drastically depending on network traffic and load on servers.

Other Experimentation

Bcast using only process mapping

- 

Bcast using intercommunicator MPI calls

-