CS633A

Parallel Computing

Assignment 2

Rohit Raj
20111051

Mani Kant Kumar
20111030

Instructor:
Dr. Preeti Malakar

1 Overview

In this assignment, we tried to optimise the following collectives namely **MPI_Bcast**, **MPI_Reduce**, **MPI_Gather** and **MPI_Alltoallv**.

- MPI_Bcast is used to broadcast the elements to all processes including itself from root.
- MPI_Reduce is used by processes to apply a reduction calculation. The values sent by different processes will be combined by the reduction operation given and the final result will be stored on the process that is considered as root.
- MPI_Gather takes elements from all other processes and gather them to a single process named as root.
- MPI_Alltoallv is used to send and receive different size of data from each process. It is generalised collective operation.

We performed the assignment for the following configurations:

- P (number of nodes) = 4, 16
- ppn(number of processes per node) = 1, 8
- N (data points per process) = $16KB$, $256KB$, $2048KB$

2 Experimental Setup

- The aim is to optimize the collectives so that we get efficient time than the standard collectives. So we created sub-communicators to exchange data which will reduce the bottleneck at root process. We first need to find the number of sub-communicators so that we can reduce the overload at root process for each configuration. The main part was to decide whether we create sub-communicators on group level or node level. As communicators for node level took more time so we chose to create them on group level i.e. the nodes in one group will be in one communication world and all their calls and information will be present at their group leader. And finally, all leaders of their respective groups will communicate with each other by passing their information to the root.

- To generate boxplot we used python and its libraries like numpy, matplotlib.pyplot, pandas, seaborn, catplot etc.

3 Instructions to run the code

- chmod +x run.sh (in case root access is required)
- ./run.sh

4 Explanation of code

As the motive was to optimise the following functions namely **MPI_Bcast**, **MPI_Reduce**, **MPI_Gather** and **MPI_Alltoallv**, we thought of creating **sub-communicators** to optimise them.

As per the assignment it is given that combine the collective calls from the MPI ranks of the same node into a single collective call instead of multiple calls from the same node to optimize collective calls. So we came up with idea that we should create sub-communicators on the basis of group level from the **nodefile.txt**. We came to know that **Intra** group distance of nodes in a cluster is 2 and **Inter** group distance is 4.

We created sub-communicators as per our need in order to reduce time. For instance if there are 4 groups from which we have taken nodes then the number of sub-communicator that we need will be **5** i.e **4** for the groups and **1** group which will consist of all leaders from their respective groups.

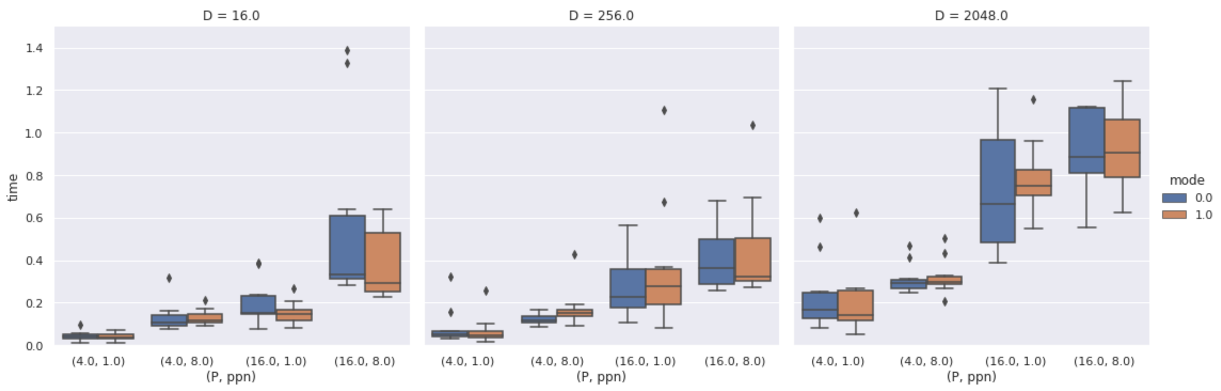
- **MPI_Bcast** : In Bcast, we did the communication between group leaders so that they can get all the information then all the group leaders will distribute that information to their respective groups.
- **MPI_Reduce** : In Reduce, we used the **MPI_SUM** operator. We summed up all the elements within a particular process, then we reduced it to partial_sum with root as leader of that group. After that all the group leaders having partial_sum of their respective groups communicate among them to get the total_sum i.e. equal to the sum we got from the default case.

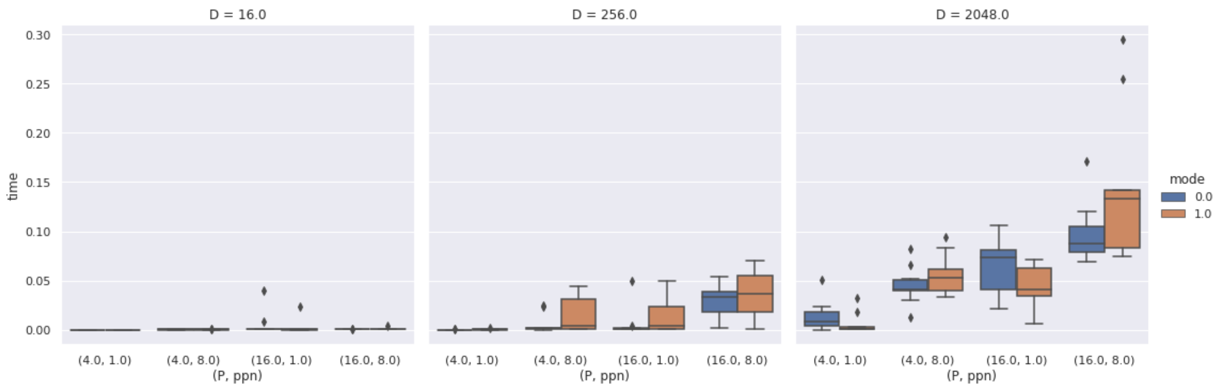
- **MPI_Gather** : In this collective call we divided process into sub-groups. Each process of sub-group communicate with their group leader and gather data at root process of that group. In other way we can say that we did intra group communication followed by inter-group communication. We gathered all the elements from different processes to root.

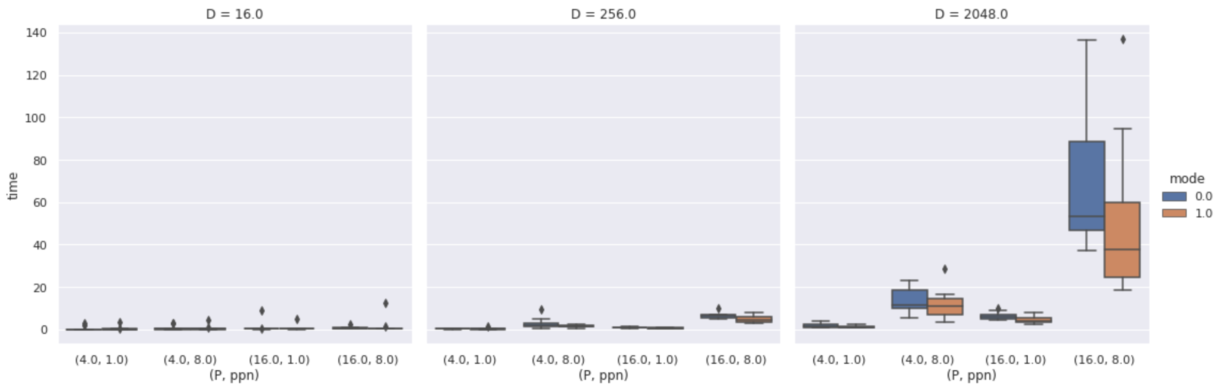
The functionality of individual files submitted:

- **run.sh** : This job script runs all the configurations mentioned in the assignment. It first compiles the code using the “make” command and then generates machinefile / hostfile on-the-fly based on the node status so that jobs never fail. After generation of hostfile, the script performs every configuration as mentioned. The generated data* files are used to generate boxplot for each P*ppn (number of processes).
- **Makefile** : compile the code named ”src.c” using the “make” command
- **script.py** : search out all the online nodes(from nodefile.txt) from the allhost files and store it in the hosts file.
- **src.c** : contains the code in C to perform the given experiment and generate 4 data files.
- **plot.py** : generates 3 boxplot (named as plotP.jpg) using the generated data files.

5 Plots


D = 16.0
D = 256.0
D = 2048.0
time
(P, ppn)
(4.0, 1.0)
(4.0, 8.0)
(16.0, 1.0)
(16.0, 8.0)
mode
0.0
1.0


D = 16.0
D = 256.0
D = 2048.0
time
(P, ppn)
(4.0, 1.0)
(4.0, 8.0)
(16.0, 1.0)
(16.0, 8.0)
mode
0.0
1.0


D = 16.0
D = 256.0
D = 2048.0
time
(P, ppn)
(4.0, 1.0)
(4.0, 8.0)
(16.0, 1.0)
(16.0, 8.0)
mode
0.0
1.0

6 Observations

The time taken for optimised **MPI_Bcast** and **MPI_Gather** reduces significantly in most of the cases especially when data is large. So, we can say our optimised **MPI_Bcast** works well.
The time taken for optimised **MPI_Reduce** reduces in few cases as we can see in the 2nd plot above. We tried to generate all the configurations of hostfile. For instance, P=16 and ppn=8 we generted two hostfiles of (4,4,8) and (2,8,8) (no. of groups, nodes per group, cores per node) and we got better result for (4,4,8). We observed that we didn't get the expected result in case of (P=4 and ppn=8), (P=16 and ppn=8).
At the last, we were able to implement the default **MPI_Alltoallv** but failed to implement the optimised version. In general the order of execution time for 3 configurations especially for large data is :

- **MPI_Reduce $<$ MPI_Bcast $<$ MPI_Gather**