Assignment2

- src.c contains 4 functions for optimized Bcast, Gather, Reduce, Alltoallv and 4 functions for their default form.
- For Bcast, Gather and Reduce, optimization of type 2 is used for generating datafile. Optimization type 1 and 3 are also present but commented.
- For Alltoallv, optimization type 4 is used.
- Datafile and Plots are formed using hostfile of category 3(default in script.py).
- Plots of all tried optimizations are mentioned in report but the plots of best optimizations for every collective call in provided in git repo.
- Number of groups is taken as 4 explicitly so as to get nodes from different groups. As we are aware of topology, we tried this so that we do not get all nodes from same group. To test default and optimized versions of collective calls in better way and for better observation, we took number of groups as 4.
- Process per node(ppn) is passed as second argument from cmd. First argument is data size D(in KB) as mentioned in assignment pdf.
- run.sh is main job script. So, running only run.sh file will end up creating data.txt,hostfile (on the fly) and all 4 plots.

Optimizations

- Different optimizations are performed based on different types of hostfiles. For different placements of processes, three different categories of hostfiles are generated using script.py. These 3 categories are as follows:
  - **Category 1** : It contains the nodes from the groups in a sequence along with the cores mentioned (i.e. cores after semicolon). For example, for 4 groups and 2 nodes per group, this hostfile will have data in below format,

    csews2:8 {from Group1}
    csews4:8 {from Group1}
    csews17:8 {from Group2}
    csews18:8 {from Group2}
    csews34:8 {from Group3}
    csews35:8 {from Group3}
    csews50:8 {from Group4}
    csews52:8 {from Group4}

  - **Category 2** : It contains the nodes in similar fashion as category 1 but cores are not mentioned. In this category, processes will be placed in round robin manner. Example,

    csews2 {from Group1}
    csews4 {from Group1}
    csews17 {from Group2}
    csews18 {from Group2}
    csews34 {from Group3}
    csews35 {from Group3}
    csews50 {from Group4}
    csews52 {from Group4}

  - **Category 3** : It contains the nodes one from each group and then, repeating the groups till required no. of nodes complete.

csews2 {from Group1}
csews17 {from Group2}
csews34 {from Group3}
csews50 {from Group4}
csews4 {from Group1}
csews18 {from Group2}
csews35 {from Group3}
csews52 {from Group4}

In script.py file, different sections are there for different categories. One can uncomment the section to generate specific category hostfile.

- We performed different types of optimization for different Collectives. These optimizations are as follows :
  - **Type I** : Using 3 different sub-communicators. First sub-communicator, containing all processes of same node. Second sub-communicator, containing lower ranks processes of nodes(node leaders) belonging to same group. And third sub-communicator, containing lowest rank processes(group leaders) from all groups. This type of optimization was performed for the hostfile of category 1 and mainly for Bcast, Gather and Reduce. For example, taking 4 processes per node in hostfile category 1,

    csews2:4 {Rank 0,1,2,3}
    csews4:4 {Rank 4,5,6,7}
    csews17:4 {Rank 8,9,10,11}
    csews18:4 {Rank 12,13,14,15}
    csews34:4 {Rank 16,17,18,19}
    csews35:4 {Rank 20,21,22,23}
    csews50:4 {Rank 24,25,26,27}
    csews52:4 {Rank 28,29,30,31}

    First sub-communicator : Rank 0-3(one group), 4-7(one group), 8-11(one group), 12-15(one group), 16-19(one group), 20-23(one group), 24-27(one group), 28-31(one group)
    Second sub-communicator : Rank 0 and 4(one group, node leaders), Rank 8 and 12(one group, node leaders), Rank 16 and 20(one group, node leaders), Rank 24 and 28(one group, node leaders)
    Third sub-communicator : Rank 0, 8, 16, 24 (in single group, can say group leaders)
    Reason : We tried this optimization to reduce the multiple calls from same node. First collective calls between the group leaders, Second collective calls between node leaders within groups simultaneously in all groups and third between the processes within same node simultaneously in all nodes.
  - **Type II** : Using 2 sub-communicators. First sub-communicator, containing all processes of same node. Second sub-communicator, containing lower rank processes of all nodes(node leaders). This type of optimization was performed for the hostfile of category 2 and 3 and mainly for Bcast, Gather and Reduce. For example, taking 4 groups, 1 node per group and 4 processes per node,

    csews2 {Rank 0,4,8,12}
    csews17 {Rank 1,5,9,13}
    csews34 {Rank 2,6,10,14}
    csews50 {Rank 3,7,11,15}

    First sub-communicator : Rank 0,4,8,12(one group), 1,5,9,13(one group), 2,6,10,14(one group), 3,7,11,15(one group)
    Second sub-communicator : Rank 0, 1, 2, 3, (in single group, node leaders)

    Reason : We tried this optimization for hostfile of category 2 and 3 as Default Collective Calls algorithm were experiencing more inter group hops and our optimization was having less inter group hops. Also, we need to create only 2 sub-communicators. First collective calls between the node leaders, second collective calls between the processes within same node simultaneously in all nodes.

– **Type III** : Using 1 sub-communicator and Isend/Recv calls . First sub-communicator, containing all processes of same node. Isend/Recv between lower rank processes of all nodes. This type of optimization was performed for the hostfile of category 2 and 3 and mainly for Bcast and Gather. For example, taking 4 groups, 1 node per group and 4 processes per node,

csews2 {Rank 0,4,8,12}
csews17 {Rank 1,5,9,13}
csews34 {Rank 2,6,10,14}
csews50 {Rank 3,7,11,15}

First sub-communicator : Rank 0,4,8,12(one group), 1,5,9,13(one group), 2,6,10,14(one group), 3,7,11,15(one group)
Multiple Isends from Rank 0 to 1, 2, and 3.

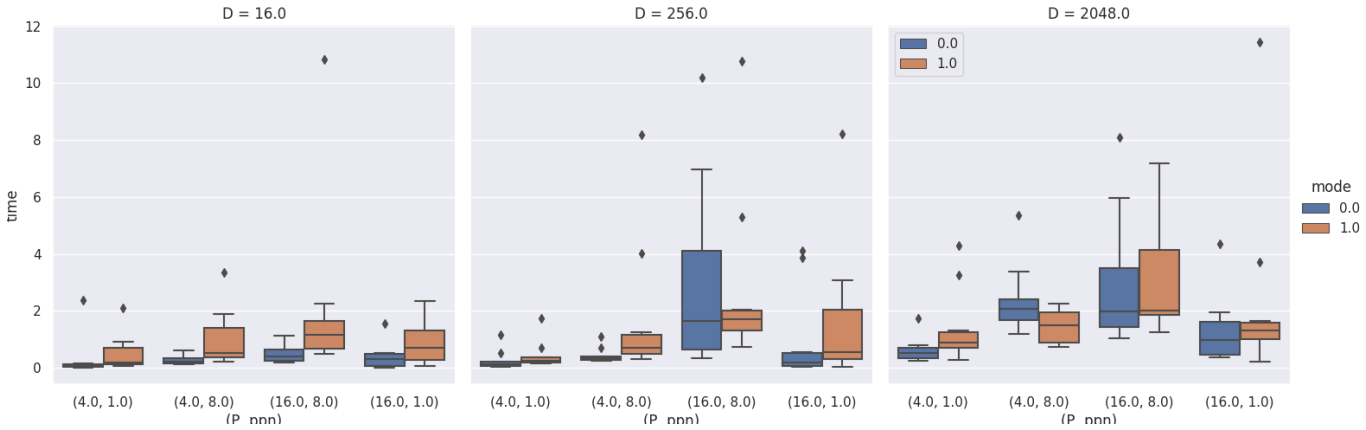
Reason : We tried this optimization for hostfile of category 2 and 3 as Default Collective Calls algorithm were experiencing more inter group hops and our optimization was having less inter group hops. First, nodes number of Isends between the node leaders, Second collective calls between the processes within same node simultaneously in all nodes.

– **Type IV** : This is used for Alltoallv collective call. In this optimization, Scatterv is used from all processes. We used hostfile of category 3 for this.
Reason : We just tried this variant to test whether multiple scatterv performs better or not.

MPI_Bcast

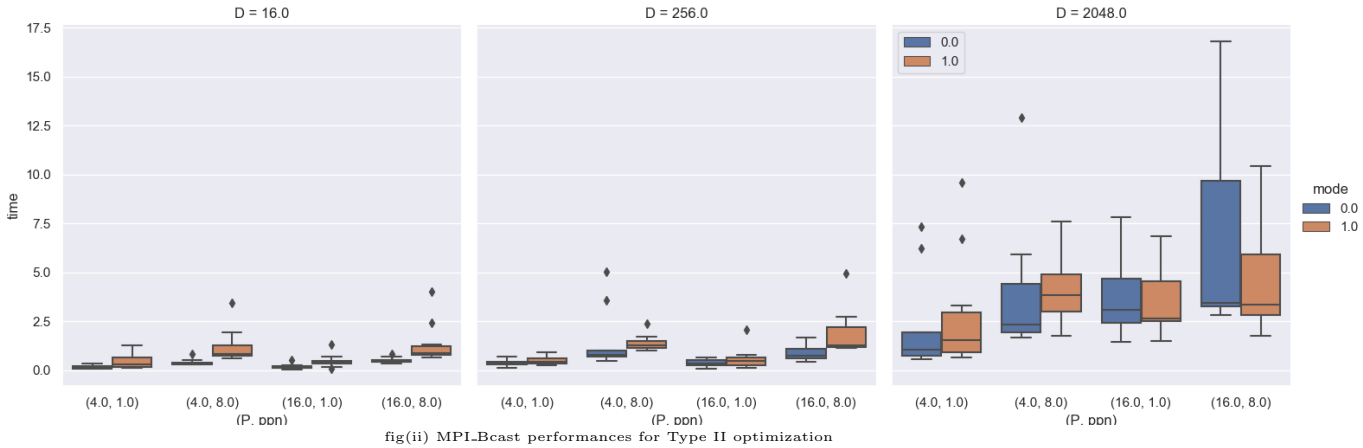
- For MPI_Bcast, Type I, II and III optimizations were performed.
- It was observed that all three types of optimizations performed better for data size 2048 KB.
- For less data size i.e. 16 KB and 256 KB, default Bcast performed better, because in three optimizations creation of sub-communicators were affecting performance.
- From all types of optimization, Type III optimization was best.

On type I optimization



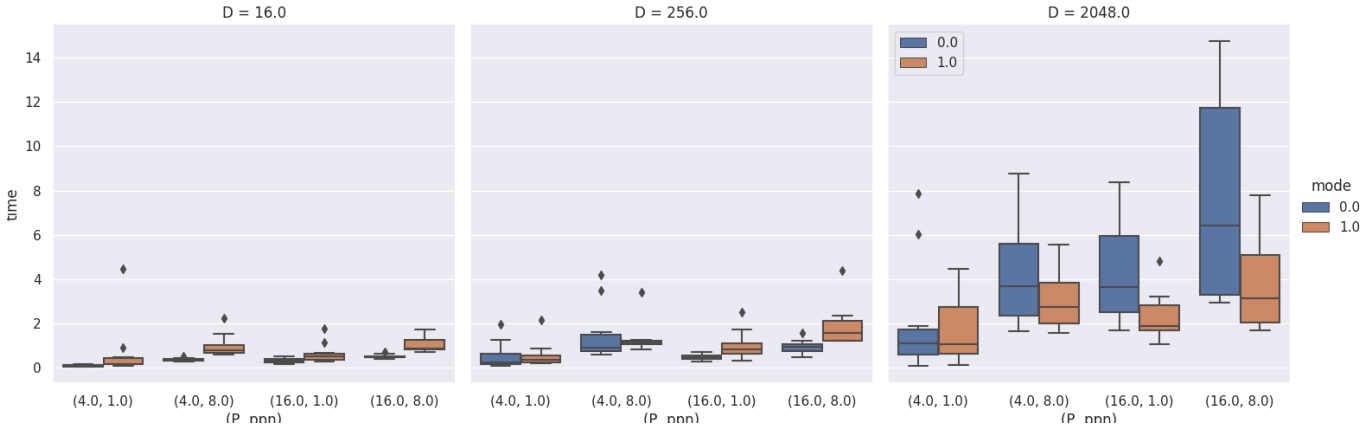
fig(i) MPI_Bcast performances for Type I optimization

On type II optimization



fig(ii) MPI_Bcast performances for Type II optimization

On type III optimization

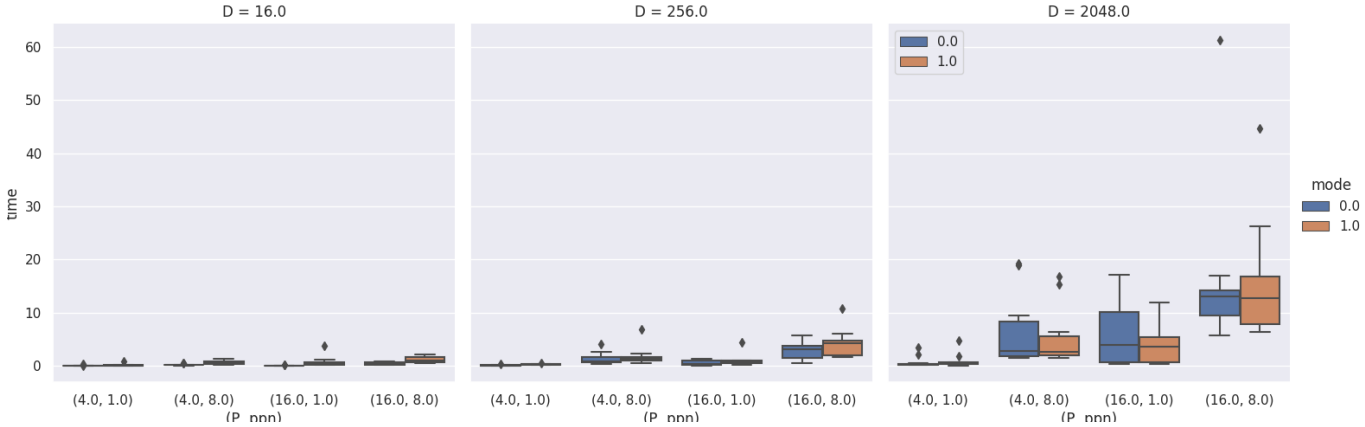


fig(iii) MPI_Bcast performances for Type III optimization

MPI_Gather

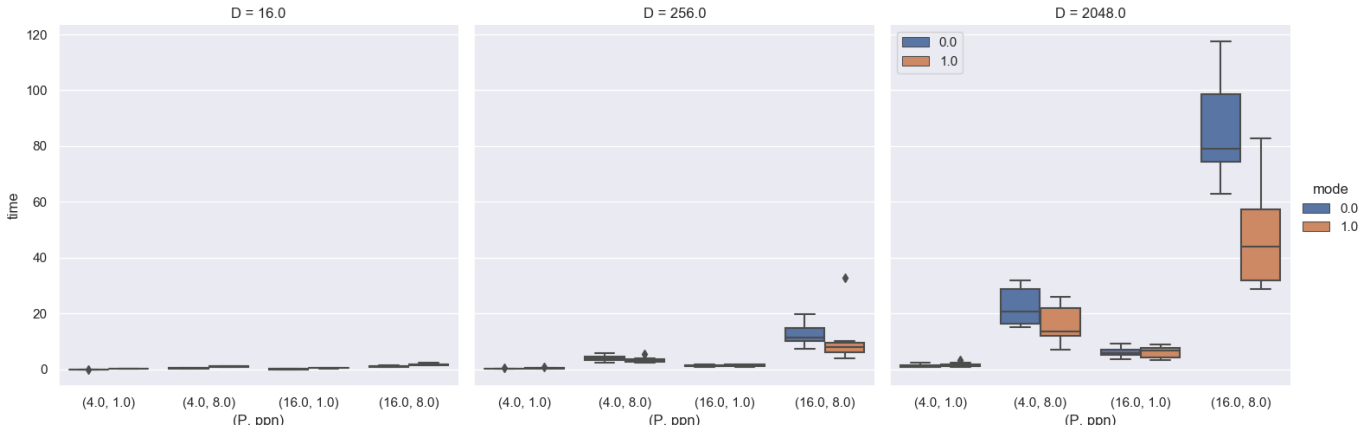
- For MPI_Gather, Type I and II optimizations were performed.
- For large data size D=2048 KB, both optimizations of Type I and II performed better. But for small size 16 KB, default was better because of sub-communicator creation time in optimizations.
- Type II optimization also performed better on data size 256 KB.
- As Type II optimization involves time of creating only two sub-communicators. Hence, overall it performs better than Type I.

On type I optimization



fig(iv) MPI_Gather performances for Type I optimization

On type II optimization

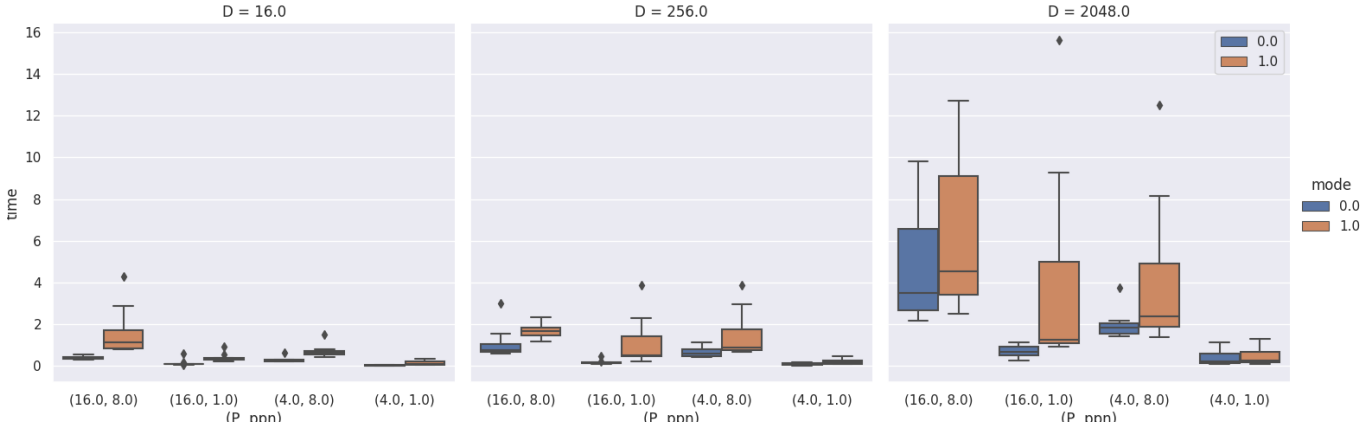


fig(v) MPI_Gather performances for Type II optimization

MPI_Reduce

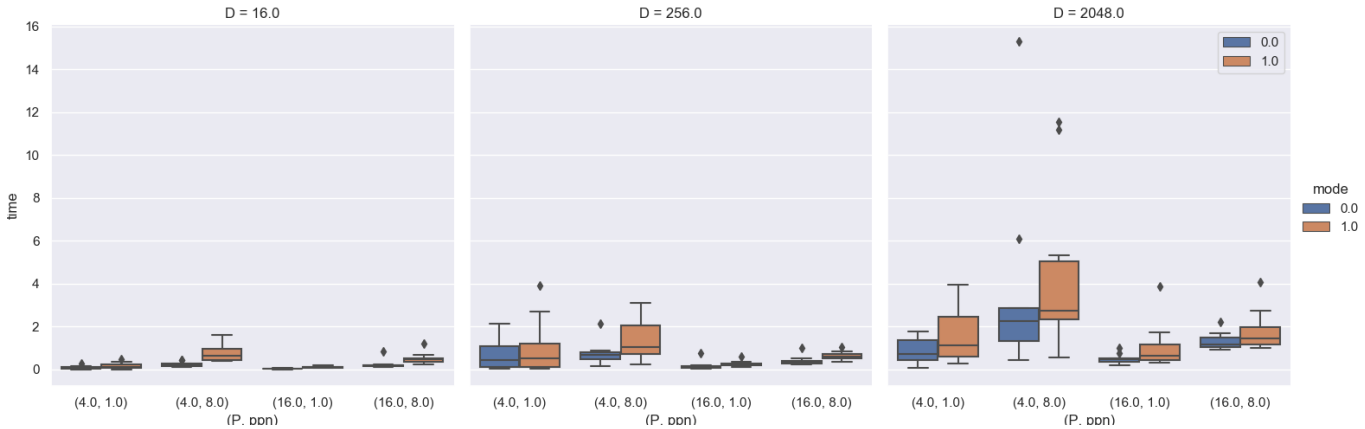
- For MPI_Reduce, we tried Type I and II optimizations.
- Both types of optimizations performed bad for data size 16,256,2048 KB. Default collective call performs better.
- Between these two types of optimization, Type II is better than Type I as it involves creation of only two sub communicators.

On type I optimization



fig(vi) MPI_Reduce performances for Type I optimization

On type II optimization

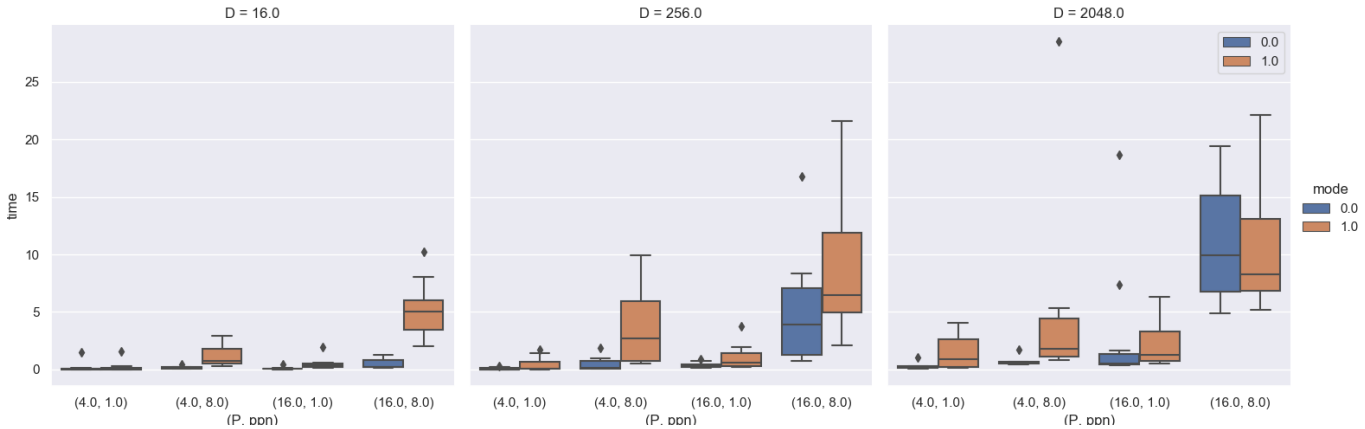


fig(vii) MPI_Reduce performances for Type II optimization

MPI_Alltoallv

- For MPI_Alltoallv, Type IV optimizations were performed.
- We observed that Default Alltoallv performs better in all cases.

On type IV optimization



fig(viii) MPI_Alltoallv performances for Type IV optimization

Experimental setup

We changed the order of groups in nodefile.txt so as to get less loaded nodes. We observed that after shuffling the groups, time taken by the collective calls gradually decreased.