Assignment-4

Mandar Deshpande
210010014

Explanation of program:

Init function : I have allocated memory to 3 arrays (each of size logP). 2 arrays for MPI_Request and one to receive values after every phase.

Allreduce function :
Number of phases = log2(P)
For each process, in every phase, the message will come from the process with rank **(size + rank - (1 << phase)) % size** and this process will send a message to the process with rank **(rank + (1 << phase)) % size .** I used (1<<phase) to reduce load on the CPU due to the pow() function. I have used the same tag for every message, because I am not broadcasting the message. Before the start of the next phase, each process must **Wait** for its **Irecv** to complete. After Wait, I increment the value that process will send in the next phase by the value received in the current phase. In the end, the value in the sendbuf is our reduced value.

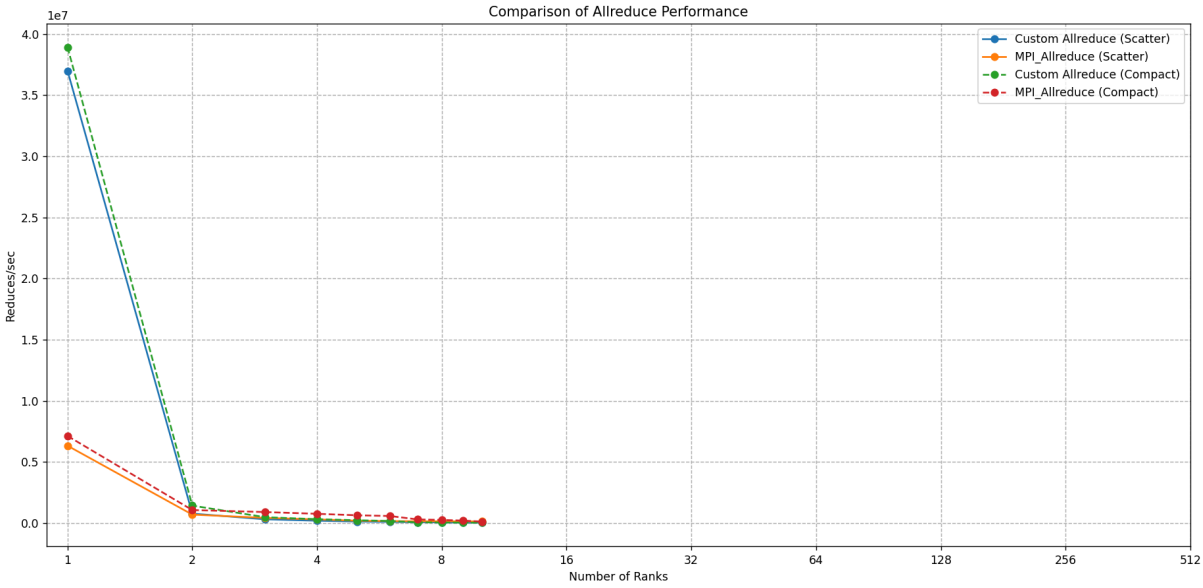
Significance of Non-blocking APIs:

By using blocking send and recv apis, there is a possibility of deadlock.

Difference between compact vs. scatter MPI rank placement:

In Scatter assignment, tasks are distributed across nodes as evenly as possible. In Compact assignment, tasks are placed contiguously on each node before moving on to the next node. In scatter assignment, inside the loop, it calculates the number of tasks per node (-ppn) dynamically based on the current number of tasks and the number of nodes available. However, in Compact, the number of tasks per node (-ppn) is fixed to SLURM_NTASKS_PER_NODE.

Graph showing comparison of Custom allreduce performance:



Observations from the Graph:

For custom implementation, reduces/sec for RANK =1 and 2 are more than MPI_Allreduce for both custom and scatter rank placement. The performance for custom implementation reduces as we increase the number of tasks. Program didn't scale well. Scaling behavior for Scatter is much better than Compact.

How given code tests for correctness:

Given code tests correctness based on **recvbuf** value of only the first process (RANK 0) with expected value (which is sum of an Arithmetic Progression).

New test cases added:

I added the following condition in the existing if loop:
((rank == size - 1) && (recvbuf != expect))
If the previous condition and this condition both are true, that means process with Rank 0 and process with Rank size-1, both have correct recvbuf. Adding this condition makes sure that all processes have the correct value of recvbuf.