Name: Brijesh Mavani
CWID: A20406960
University: Illinois Institute of Technology
Course: Parallel and Distributed Processing
Assignment: 3 - Design Report

There are two communication methods to implement MPI code.

1) Point to point communication

2) Collective communication

I have implemented the code in both point to point communication and Collective communication. While doing the analysis of the executions, I found that Collective communication provides more speed up and the efficiency. **Hence, code implemented with Collective communication (MPICollective.c) is the final. Just for a reference, I have provided code for point to point communication (MPIP2P.c).**

I have provided details for both implementations below:

1. Point to Point communication: **The Array size should be multiple of number of processors and less than 100000000.** Once the array is initialized by the main processors (rank =0), input array will be divided into the equal number of elements (array size/number of the processors) and scattered to all the processors. So, each processor will work on its individual array chunk and calculate the partial prefix sum. The code will loop to pass the partial sum computed to other processors. For this index will be calculated and play the major part in the point to point communication. Instead of passing partial sums in a sequential manner i.e process 0 will pass to processor 1 and processor 1 pass to processor 2, so on, the interleaved logic is built which with each pass processor passes its partial sum to processor away from it by array index. In the 1st pass, each processor will pass the partial sum computed to immediate next processor. The receiving processor will revise its partial sum by adding the sum received from previous processor. In 2nd pass, all processors which already sent its partial sum will do nothing and processor with revised partial sum, send it to processor apart from it by 2. This process will continue and at the end last processor will have the final prefix sum computed.

   Once, the prefix sum is computed it will be printed as output.

   For the correctness of the algorithm, I have used the MPI barrier to sync all the processor to prior to prefix sum communications & updates and after the communications. Also, I have computed the total sum of the array during initialization itself. So, correct result of the prefix sum should match with the total sum computed during initialization. I have printed both as the output and compared them.

2. Collective communication: **The Array size should be multiple of number of processors and less than 100000000.** Once the array is initialized by the main processor (rank =0), input array will be

divided into the equal number of elements (array size/number of processes) and scattered to all the processors. So, each processor will work on its individual array chunk and calculate the partial prefix sum.

Once the partial sum is computed, MPI_Barrier is invoked to sync all the processors. Then MPI_Scan function to computes the scan (partial reductions) of data on each process. After computing the partial reductions with MPI_Scan function, all the results are collected at processor 0 with MPI_Gather function. The result of MPI_Gather function is then displayed as output.

For the correctness of the algorithm, I have used the MPI barrier to sync all the processors to prior to prefix sum communications & updates and after the communications. Also, I have computed the total sum of the array during initialization itself. So, correct result of the prefix sum should match with the total sum computed during initialization. I have printed both as the output and compared them.

This approach reduces the inter-process communication drastically compared to point to point communication. Hence, it provides more speed up and the efficiency. This can be seen in the result document. Though the result can vary with different number of processors and input array size and computation time heavily dependent on those two factors.

In theory, the sequential code will have computation time of O(n) where n is array length and parallel implementation will have computation time of O(n/p) where n is n is array length, p is number of processors. With this speed up (Ts/Tp) will be p. This will give us the efficiency (Sp/p) of 1. Here, we are getting sub-optimal results but better than sequential execution.

- Efficiency at 16 processes:
  - ✓ Collective Communication: 0.38
  - ✓ Point to Point communication: 0.12

- Speed up at 16 processes:
  - ✓ Collective Communication: 6.159852
  - ✓ Point to Point communication: 1.97209