CS633: PARALLEL COMPUTING

ASSIGNMENT 2

HALO EXCHANGE WITH NEIGHBORING PROCESSES WITH LEADER on HPC2010

Group Members - Group 28

Name	Roll Number	Email ID
Mehta Shrey Kartik	200580	mehtask20@iitk.ac.in
Priyanka Jalan	190649	prianka@iitk.ac.in
Arpit Kumar Rai	200190	arpitkr20@iitk.ac.in

Problem Statement

Write a program to perform halo exchange incorporating topology-aware hierarchical communication (Lecture 14).

The goal is to reduce the number of inter-node communications. For example, consider a 9-process decomposition of the 12x12 domain on the right. In the default case, there will be 0->3, 1->4, 2->5 communications from node 1 to node 2 (i.e. 3 simultaneous communications) in case of 3 nodes with ppn=3. You may select any rank as the leader rank on each node and reduce the number of inter-node communications (as discussed in class).

Code Explanation

1) The code initializes a 2D array data buffer using the random initialization method provided in the question.

   srand(seed*(myrank+10)); data[i][j] = abs(rand()+(i*rand()+j*myrank));

   We use two arrays *data* and *data_without_leader* with the same values to perform the 9-point stencil exchange with and without a leader rank, respectively. The code for the case without leader rank remains the same as submitted in the previous assignment.

2) In case of performing communication with the leader ranks, to accommodate the P processes in continuous groups of order Px, we form separate sub-communicators for the processes with ranks (0,...Px-1), (Px,...2*Px-1).. , i.e. for the processes in the same node, so that we can efficiently use the Gather and Scatter function of MPI to send and receive the data from all the process in that node to its rank leader.

```
MPI_Comm newcomm;
int color = my_rank / Px;
MPI_Comm_split (MPI_COMM_WORLD, color, my_rank, &newcomm);
```

3) For each and every process, we gather the data of the first two and the last two rows of its matrix and send it to the rank leader (we have taken the rank 0 in that sub-communicator to be the leader) in the buffers *intra_node_left_recv_up* and *intra_node_left_recv_down*, respectively using the function *MPI_Gather*.

```
MPI_Gather(&data[0][0], 2 * side_len, MPI_DOUBLE, intra_node_left_recv_up, 2 * side_len, MPI_DOUBLE, 0, newcomm);  // Gathers
MPI_Gather(&data[side_len-2][0], 2 * side_len, MPI_DOUBLE, intra_node_left_recv_down, 2 * side_len, MPI_DOUBLE, 0, newcomm);
```

4) Exchanging data across leaders of different sub-communicators: We then exchange the data in the buffer *intra_node_left_recv_up* with the leader of the sub-communicator above the current one *(Rank = my_rank - Px)* in the buffer *inter_node_down_recv* and the data in *intra_node_left_recv_down* with the leader of the sub-communicator below the current one *(Rank = my_rank + Px)* in the buffer *inter_node_up_recv* (for whichever processes applicable) using the Odd-Even NN-2 algorithm for communication across the leaders (inter-node communications).

```
// Intra-Columns NN-2
if(my_rank % Px == 0) {
    int col_rank = my_rank / Px;
    if (col_rank % 2 == 0 && col_rank < Py - 1)
    { // Down send/recv
        MPI_Send(intra_node_left_recv_down, 2 * Px * side_len, MPI_DOUBLE, my_rank + Px, my_rank + Px, MPI_COMM_WORLD);
        MPI_Recv(inter_node_down_recv, 2 * Px * side_len, MPI_DOUBLE, my_rank + Px, my_rank, MPI_COMM_WORLD, &status);
    }
    else if (col_rank % 2 != 0 && col_rank > 0)// Up send/recv…
    if (col_rank % 2 != 0 && col_rank < Py - 1) // Down send/recv…
    else if (col_rank % 2 == 0 && col_rank > 0) // Up send/recv…
}
```

5) These values received are then scattered to the processes in the sub-communicator using the function *MPI_Scatter*, so that all the processes get the data they require for the computation, i.e. the data of the two relevant rows of the above and below process.

```
MPI_Scatter(inter_node_up_recv, 2 * side_len, MPI_DOUBLE, up_recv, 2 * side_len, MPI_DOUBLE, 0, newcomm);
MPI_Scatter(inter_node_down_recv, 2 * side_len, MPI_DOUBLE, down_recv, 2 * side_len, MPI_DOUBLE, 0, newcomm);
```

6) The remaining communications of the data of the required columns for the computation are done in the same way as in Assignment 1, packing them and using the Odd-Even NN-2 algorithm to communicate across the processes in the same node (intra-node communications). The computation is then done using the data received and the value is updated for each time step. Also, the denominator calculation for the edge cases is done exactly the same way as in Assignment 1.

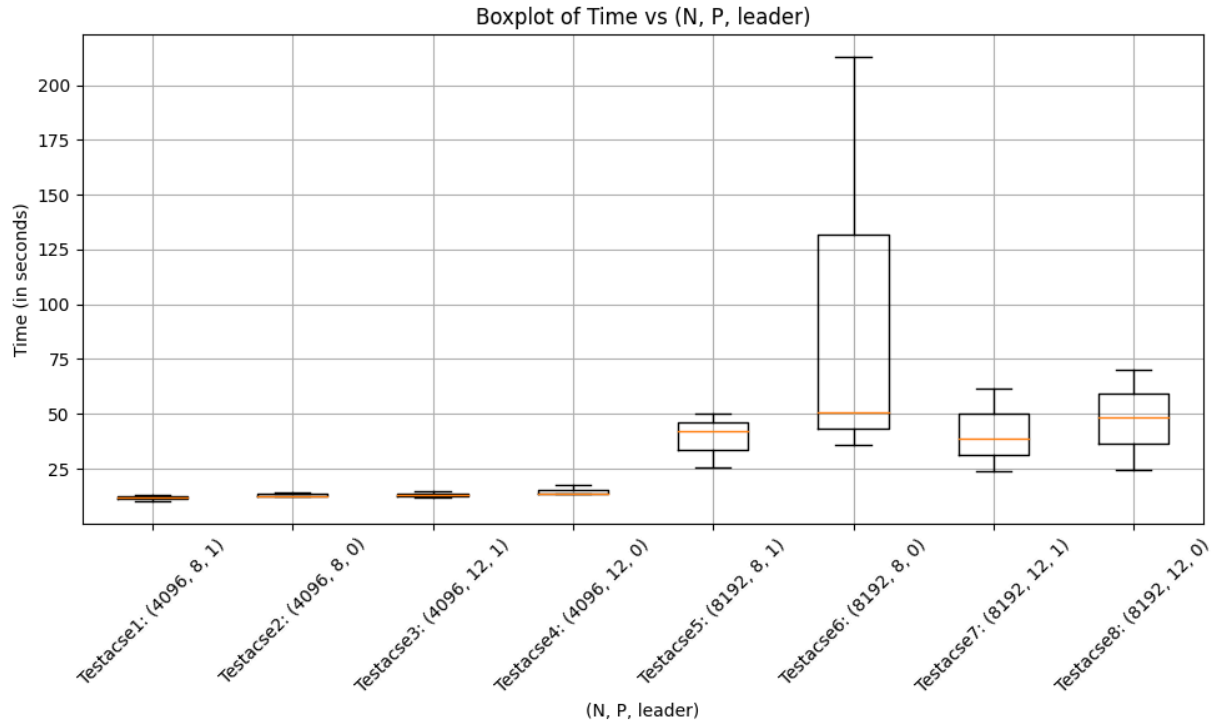
Optimizations

1) **Inter-Node Communication only between Rank Leaders:** The major cost in time is incurred for the inter-node communication between the processes, so we use the mode of inter-node communication between the rank leader instead of all the processes in the node.
2) **Used collective communication methods**: We used the MPI functions MPI_Gather and MPI_Scatter in place of normal MPI_Send and MPI_Recv to send the data to the rank leader to reduce the latency for the communication.
3) **Employing Odd Even Nearest Neighbour (NN-2) Algorithm**: NN-2 gives better performance as compared to NN-1 for a large number of processes as the blocking Send and Recieves ultimately leads to sequential completion of the communication.

Timing Plot

The timing plot shows the time taken by the main halo exchange on Prutor, including the computations and communications and excluding the initializations. The x-axis represents the test cases (#datapoints, #processs, leader) on which the program was run and the y-axis denotes the time in seconds

Test Case 1 (4096, P = 8, With Leader)	Test Case 2 (4096, P = 8, Without Leader)	Test Case 3 (4096, P = 12, With Leader)	Test Case 4 (4096, P = 12, Without Leader)	Test Case 5 (8192, P = 8, With Leader)	Test Case 6 (8192, P = 8, Without Leader)	Test Case 7 (8192, P = 12, With Leader)	Test Case 8 (8192, P = 12, Without Leader)
10.227533	12.684958	14.735146	17.509435	42.172452	50.845554	61.572438	69.974211
12.910762	14.253065	12.869342	13.575183	50.161943	212.863907	38.470790	48.238835
12.170771	12.548478	11.990742	13.416233	25.321630	36.073100	23.877489	24.532015



Performance Observation

We analyze the parallel performance by observing the effect of **data size** and **leader implementation**.

1. **Data Size** : As the data size per process increases, the time taken by both with and without leader implementation increases This change can be attributed to the

increase in the stencil computation time as well as the increase in the communication time between neighboring processes. Another interesting observation is that the variation of the time plot is greater for higher data size because for higher data size, inter-node communication becomes the bottleneck factor.

2. **LEADER vs NO LEADER implementation**- It can clearly be seen that halo exchange has performance improvement with leader process chosen to be on the same rack node. Total number of inter node communications are reduced significantly from 4 per row to 1 per row by using leader processes. Similarly intra-node communications are increased by 2 per node (1 communication for gathering all data to rank 0 and 1 communication for scattering data from rank 0 to Px-1 processes). As time(inter-node) >>>> time(intra-node), an expected decrease in time taken is observed.