CS633 Assignment 2

Group 19
Rishabh Arijeet-210842(rishabh21@iitk.ac.in)
Utkarsh Mishra-211131(utkarshm21@iitk.ac.in
Narendra Singh - 210649(narendras21@iitk.ac.in)

September 2, 2024

MPI Parallel Computation: Readme

This repository contains an MPI-based parallel computation code for performing grid averaging with halo exchange. The code is designed to distribute the computation across multiple processes, utilizing MPI for communication between processes.

Code Components and Functionalities

1. **Initialization and MPI Setup:**
   - The code includes necessary libraries such as `mpi.h` for MPI functions and other standard libraries.
   - The `main()` function initializes MPI, retrieves the rank of the current process, and obtains the total number of processes (`P`) in the MPI communicator (`MPI_COMM_WORLD`).
   - Input arguments (`Px`, `N`, `steps`, `seed`, `stencil`) are read from the command line.
2. **Halo Exchange and Averaging:**
   - The main computation loop runs for `steps` iterations, representing the number of times the averaging process will be performed.
   - Each process initializes a local grid (`data`) with random values. The grid size for each process is `n` by `n`, where `n` is calculated as the square root of `N`.
   - Processes are arranged in a 2D grid of `Px` rows and `Py` columns based on the total number of processes `P`.
   - Halo exchange is performed for the boundary points of each process:
     - Processes determine which sides need communication (left, right, up, down) based on their position in the process grid.
     - Data is packed from the boundary points (`temp_left`, `temp_right`, `temp_up`, `temp_bot`) into buffer arrays (`send_left`, `send_right`, `send_up`, `send_bot`) using MPI Pack functions.
     - `MPI_Isend` is used to send the packed data to neighboring processes.
     - `MPI_Recv` receives the packed data from neighboring processes into receive buffers (`recv_left`, `recv_right`, `recv_up`, `recv_bot`).
     - Unpacking is done to extract the received data back into arrays.
   - Once the halo exchange is complete, the averaging computation is performed on the local grid of each process.
   - The average value is computed for each point in the grid using the neighboring points according to the chosen stencil.
   - A temporary array (`temp`) is used to store the updated values.
   - The updated values are then copied to the original grid (`data`).

3. **Timing and Output:**

   - Time taken for the computation is measured using `MPI_Wtime()` before and after the main computation loop.
   - The maximum time across all processes is then found using `MPI_Reduce()` with the `MPI_MAX` operation.
   - If the current process is ranked 0, it prints the maximum time taken for the computation.

4. **Stencil Computation Explanation:**

   Without Leader :

   - For a 9-point stencil, the average at each point is computed using the point's value and its 8 neighboring points (including diagonal points). Also, for edge cases like corner points, denominator is changed using flags.

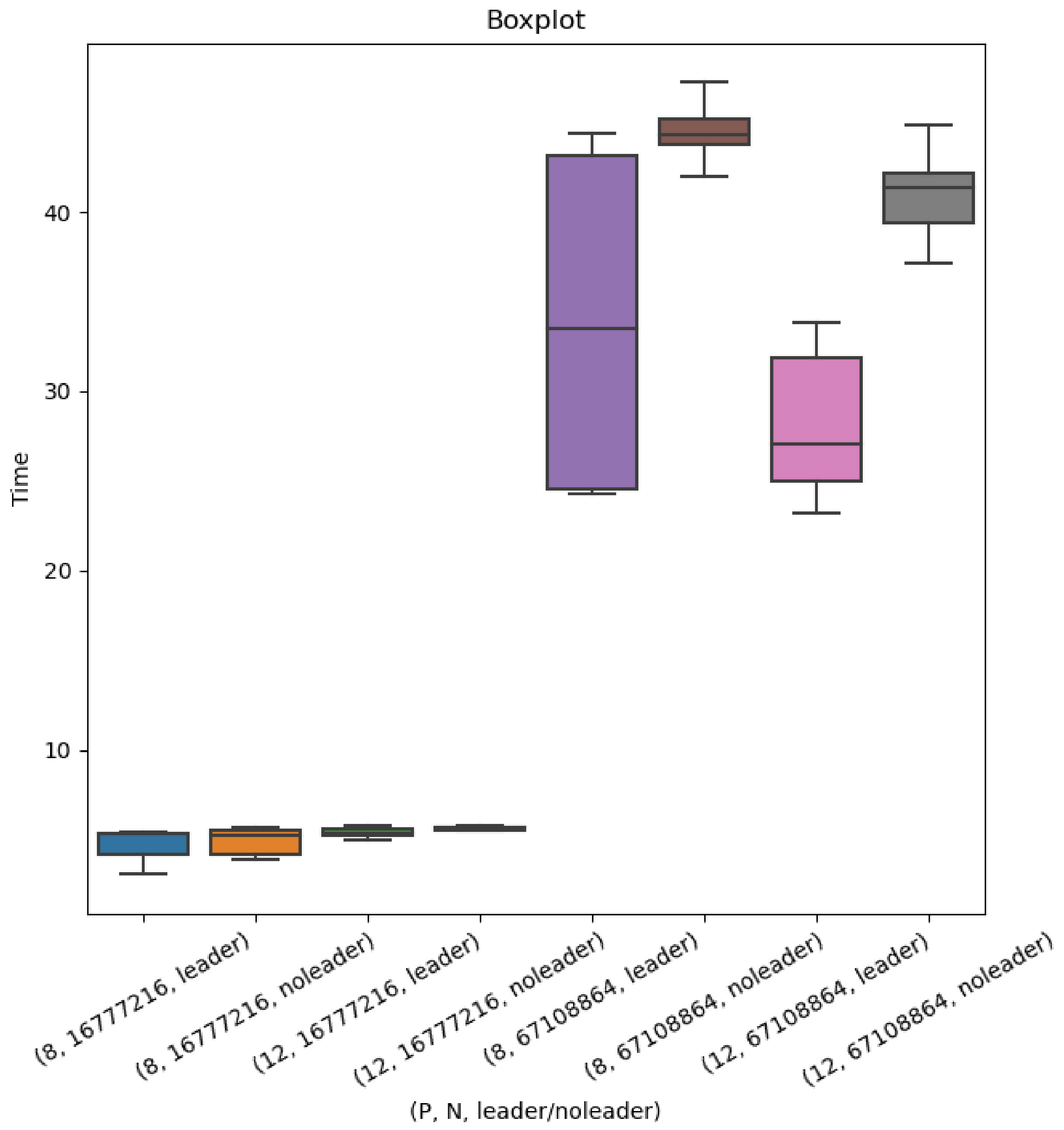
   With Leader :

   - The side-wise communications and the computation remains the same as before.
   - For the top and bottom communications, we make use of the topology of the system. We make separate communicators for each node. Instead of using a large number of inter-node communications, we use **MPI Gather** to send the required data to the leader process of each communicator, which shares the data with the leader of the receiver process via the global comm, and the receiving leader process then communicates the data with the intra comm using **MPI Scatter**

5. **Optimizations:**

   - We have used MPI ISend instead of MPI Send, which allows for asynchronous communication and overlapping communication with computation, potentially improving overall performance by enabling the sending process to perform other tasks while the message is being sent.
   - For the 9-point stencil computation, we have packed two arrays and sent them as one, reducing the number of communications between the processes, thereby reducing process execution time.
   - In the With Leader segment, we have used topology aware hierarchical communication, reducing the number of inter-node communications and the overall time taken by the algorithm

6. **Plot:**


Boxplot
Time
10
20
30
40
(8, 16777216, leader)
(8, 16777216, noleader)
(12, 16777216, leader)
(12, 16777216, noleader)
(8, 67108864, leader)
(8, 67108864, noleader)
(12, 67108864, leader)
(12, 67108864, noleader)
(P, N, leader/noleader)

7. **Plot Observations**
   The observations from the plot above are:
   - The time taken for N=16777216 and N=67108864 differ hugely, i.e. as N increases, time also increases. So, we can conclude that the time taken is largely dependent on the input size.
   - When N and P are held constant, it's observable that employing leader ranks results in shorter times for data exchange compared to scenarios without them. Although this contrast is less noticeable for N=16777216, it becomes more pronounced when N=67108864. Thus, the utilization of leader ranks reduces data exchange time by diminishing the frequency of inter-node communications. As the dataset size expands, this reduction in inter-node communications progressively becomes a more influential determinant of data exchange time.

Contributions :

- Rishabh Arijeet : 33.33 %
- Utkarsh Mishra: 33.34%
- Narendra Singh: 33.33%