CS633:Assignment2

Performance Analysis of Halo Exchange Function(with and without leaders)

GROUP 25

April 15, 2024

.

1 Code Explanation

This code implements a parallel stencil computation using MPI (Message Passing Interface). The goal of the code is to perform stencil computations efficiently by reducing the number of inter-node communications.

- **Main Function (`main`):**
  - Initializes MPI environment.
  - Retrieves the rank (`myrank`) and size (`size`) of the MPI communicator.
  - Parses command-line arguments such as the number of processes per row (`Px`), grid size (`N`), number of time steps (`num_time_steps`), and random seed (`seed`).
  - Calls two functions: `leader` and `non_leader`, which represent implementations of the stencil computation with and without a leader process, respectively.
  - Finalizes MPI environment.
- **Leader Function (`leader`):**
  - Decomposes the MPI communicator into sub-communicators based on common leaders (each leader represents a set of processes on the same node).
  - Defines custom MPI data types for efficient intranode communication.
  - Initializes the grid data with random values.
  - Performs stencil computation for the specified number of time steps.
  - Implements hierarchical communication strategy to minimize inter-node communication:
    - Gathers top and bottom boundary values within each node.
    - Communicates boundary values among leaders of different nodes using 1D nearest-neighbor communication.
    - Scatters received boundary values to all processes within each node.
  - Calculates the execution time and prints the maximum time across all processes.
- **Non-Leader Function (`non_leader`):**
  - Initializes the grid data with random values.
  - Performs stencil computation for the specified number of time steps.
  - Implements inter-node communication for boundary exchange using point-to-point communication among neighboring processes.
  - Calculates the execution time and prints the maximum time across all processes.

2 Code Structure and Optimizations

1. **Hierarchical Communication** The code organizes the MPI processes into groups based on their "leaders". Each group is associated with a node in the computational cluster. Instead of directly communicating with all neighbors, processes within the same node first communicate with each other, and then a single representative from each node communicates with representatives from other nodes. This reduces the total number of inter-node communications.

2. **Leader-Based Communication** Within each node, a single process (the "leader") is responsible for gathering data from neighboring nodes and scattering data to other processes within its node. This ensures that communication between nodes is minimized, as each node only communicates with the leaders of other nodes.

3. **Custom Datatypes** Custom MPI datatypes (`top_vector` and `bottom_vector`) are defined to facilitate efficient communication of top and bottom boundary data between processes within the same node. These datatypes allow for more compact communication of contiguous blocks of data, reducing communication overhead.

4. **Asynchronous Communication** The code uses non-blocking communication operations (`MPI_Isend` and `MPI_Irecv`) to overlap multiple communications. This helps to reduce communication latency and improve overall performance.

5. **Optimized Communication Patterns** The code optimizes communication patterns by selectively communicating only with relevant neighbors, based on the stencil computation requirements. This avoids unnecessary communication overhead and minimizes the total amount of data exchanged between processes.

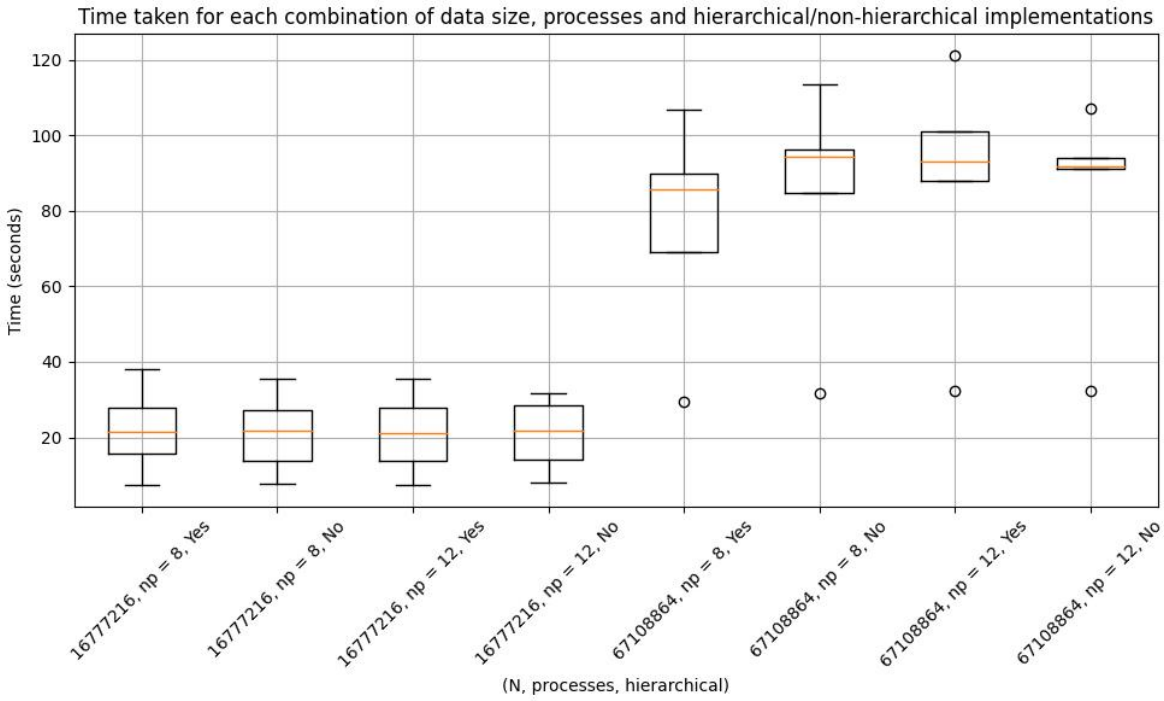
3 Performance Observations

From the boxplot of timings, the following observations can be made regarding the performance of the halo exchange function:

- As expected, the times for 8192^2 data size are greater the times for 4096^2 data size.
- We see that for 4096^2 data size, the variation between times is quite low for all configurations compared to 8192^2 data size.
- For the 8192^2 data size, the time trend for hierarchical implementation is less than the non-hierarchical implementation.
- In the case of 4096^2 data size, the difference in times of both implementations is not significant enough to be visible over the errors due to congestion.

4 Group Information

- **Group Number:** 25
- **Group Members:**
  - Lakshvant Balachandran (Roll Number:210557, Email:lakshvant21@iitk.ac.in)
  - Om Shivam verma (Roll Number: 210684, Email: omsv21@iitk.ac.in)
  - Piyush Bhatia (Roll Number: 210720, Email: piyushb21@iitk.ac.in)
- **Contributions:**
  - Lakshvant Balachandran: creating the plot code, collecting data
  - Om Shivam Verma: collecting data, putting data to create plot
  - Piyush Bhatia: optimisation by replacing Iscatter and Igather with Gather and Scatter
  - The main code was written with the inputs and efforts of all members.


Time taken for each combination of data size, processes and hierarchical/non-hierarchical implementations
Time (seconds)
120
100
80
60
40
20
16777216, np = 8, Yes
16777216, np = 8, No
16777216, np = 12, Yes
16777216, np = 12, No
67108864, np = 8, Yes
67108864, np = 8, No
67108864, np = 12, Yes
67108864, np = 12, No
(N, processes, hierarchical)