Assignment1

Group 1

22 March 2024

1 Introduction

In this assignment we implements halo exchange with neighbouring processes and stencil computation. Halo exchange involves exchanging halo region data points between neighboring processes and stencil point computation involves computing the average of neighbouring data points. The code is designed to run processes in parallel.

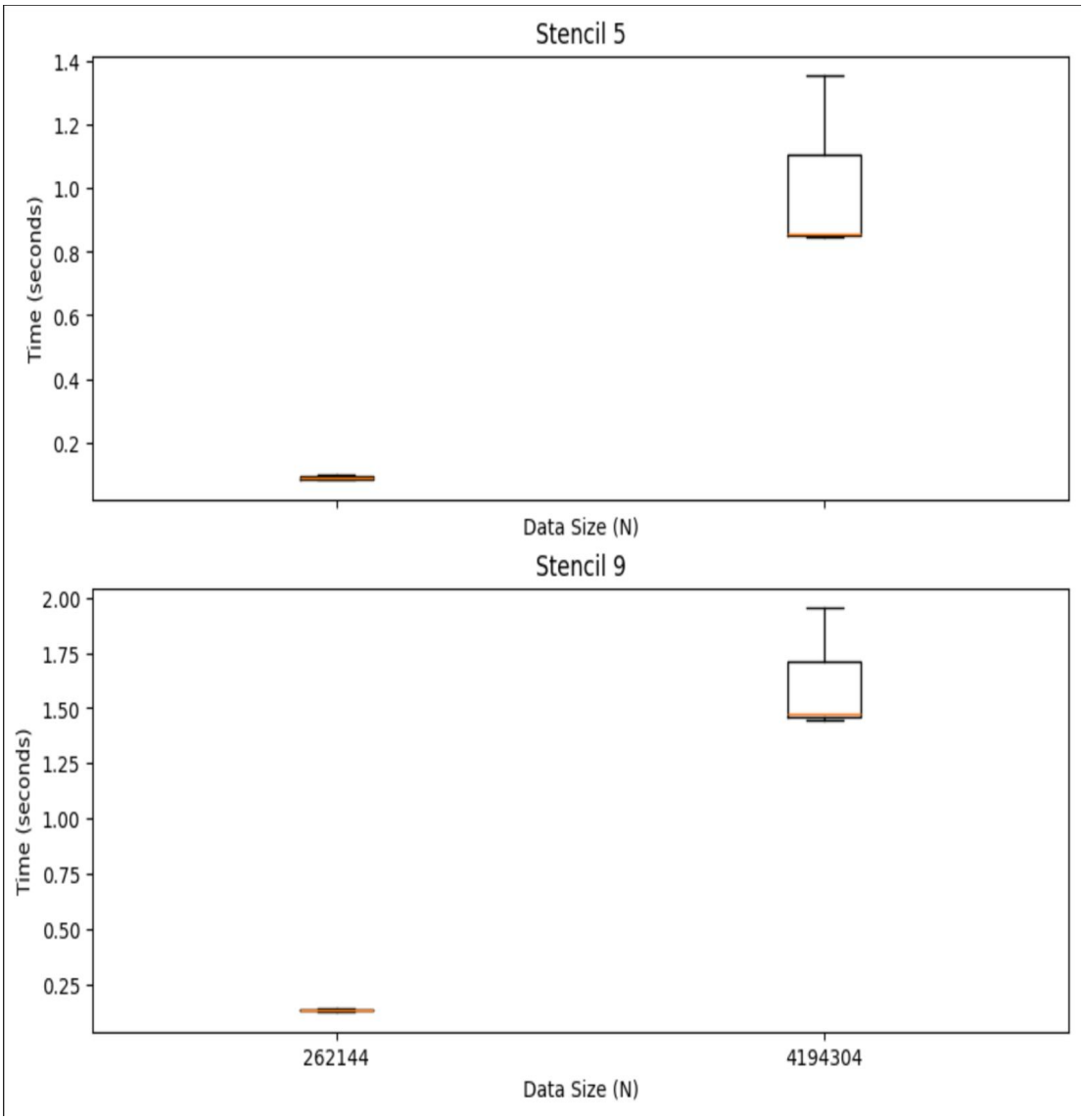
2 Code Explanation

The code is written in C language. MPI library is used for parallelization. First, process initializes its local data array with random values as mentioned in question. Then it iterates through the specified number of time steps. In each step it performs halo exchange and stencil computation for each step.

Code contains implementation of both 5 and 9 point stencil computation. Halo exchange correspond to both is written separately. Here we use MPI Pack, MPI Send and MPI Recv for communication. After receiving boundary data from neighboring process, each process do stencil computation and updates its local data.

3 Time Plot

The time plot is made using "plot.py" script. It contains box plots (from the 3 executions) for every data point in the plot. Time in seconds in y-axis and N in x-axis. Two plots are made, one for 5-point stencil and other for 9-point stencil. These plots shows the distribution of execution times for each configuration across mulitple runs.



4 Observations

From the plot we observed that:

- As the data size increases, the execution time also increases. This shows that larger data sets require more computational time.
- The 9-point stencil takes longer time to execute compared to 5-point stencil due to the additional computation and communication is involved.

- There is some variability in execution times of different configurations. This may be due to some factors like load imbalance and communication overhead.

5 Optimizations

Here are some of the optimizations made to improve performance.

- The code data efficiently packs and sends data using MPI pack and MPI send/recv, it reduce communication overhead.
- Stencil point computation is also optimized which improves overall computational efficiency.
- Each process updates its local data array instead of repeatedly accessing global data arrays. This minimized access time.
- Nearest neighbour communication is optimised by use of odd-even exchange method, which improved the communication efficiency as it avoids serialization.

6 Conclusion

This assignment demonstrated effective parallelization techniques using MPI library. By optimizing communication and computation, the code achieves improved performance for large date sets. This can be even more optimized by using hardware specific techniques.

7 Group members

- Harsh Oza (210411), harsho21@iitk.ac.in
- Kumar Harsh Mohan (210543), harshmohan21@iitk.ac.in
- Ogirala Deeven Kumar (210681), deevenk21@iitk.ac.in