Indian Institute of Technology, Kanpur
CS633 - Parallel Computing

Assignment 1

by

Yash Phirke (231010090)
Prashant Kumar (22201264)
Vishw Patel (231010087)

README

1 Code Explanation

The purpose of this code is to demonstrate parallel communication using MPI for a 2D stencil computation. It uses the **MPI_Pack / MPI_Unpack** and **MPI_Send/MPI_Recv** for sending the halo regions to its neighbouring processors. It calculates the average value of each element in a 2D grid based on its neighboring elements, iteratively over multiple time steps and it prints the **maximum** of time taken by each processes.

Upon initialization, the code initializes MPI, establishing communication channels among different processes. Each process is assigned a rank within the MPI communicator MPI_COMM_WORLD, facilitating coordination and data exchange. Command-line arguments provide essential parameters such as the number of processes in x-direction (Px), the total grid size for each processors (N^2), the number of time steps (time_step), a random seed (seed), and the stencil size (stencil). These parameters define the problem's scope and execution settings.

The local grid size is determined based on the square root of the (N^2) based on that each processors get 2D grid of N x N points. The 2D array used for computation is initialized as $[N + 2 * overlap]^2$. This additional overlap region acts as ghost cells to copy-paste the halo region grid points of the neighbouring processes. The overlap is computed based on formula $overlap = (stencil-1)/4$ which give us the overlap cells to be added in each direction. Additionally, temporary arrays and buffers are initialized to facilitate this data exchange between neighboring processes. The main 2D array is randomly initialized based on the given formula and the overlap regions are set to 0.0. The code uses boolean to determine the existence of neighboring processes which are true/false based on the the grid topology which is of Px x Py for P processors. Where Px is an input argument and Py = P / Px.

The main computational loop iterates over the specified number of time steps. Within each iteration, the code orchestrates several critical operations.

Firstly using the boolean, code checks for the existence of the processor in left/right/top/bottom. Based on that, it makes a temporary array **(temp_r, temp_l, temp_b, temp_t)** and populates it with the data from the local halo regions which is packed into buffer **(buff_r, buff_l, buff_t, buff_b)** using MPI_Pack . MPI communication functions, such as MPI_Send and MPI_Recv, facilitate this packed data exchange among neighboring processes. This Send/Recv operations are done based on the Nearest Neighbour (NN) algorithm. The code switches the buffer names while this Send/Recv operations. For example, a processor sends its right buffer [buff_r] to the right processor

which will be received and stored into the left buffer [recvbuff_l] of that processor and vice versa is done for left-right and top-bottom communications. The received buffers **(recvbuff_r, recvbuff_l, recvbuff_t, recvbuff_b)** are unpacked using MPI_Unpack library into the temporary recv-arrays **(recvtemp_r, recvtemp_l, recvtemp_b, recvtemp_t)** and copied into the respective overlap regions. This completes the communication part.

Following communications, each process performs the stencil computation on its local grid portion. This computation involves updating each grid point based on the average value of its neighboring points, as defined by the stencil pattern. The computed values are stored in temporary arrays. Then, temporary 2D array values are copied back into our main 2D array. This completes the loop.

The time is measured using **MPI_Wtime()** function, which includes the communication and computation time for the given number of time steps. The maximum execution time across all processes is determined using the **MPI_Reduce()** function, enabling efficient performance analysis.

2 Plots

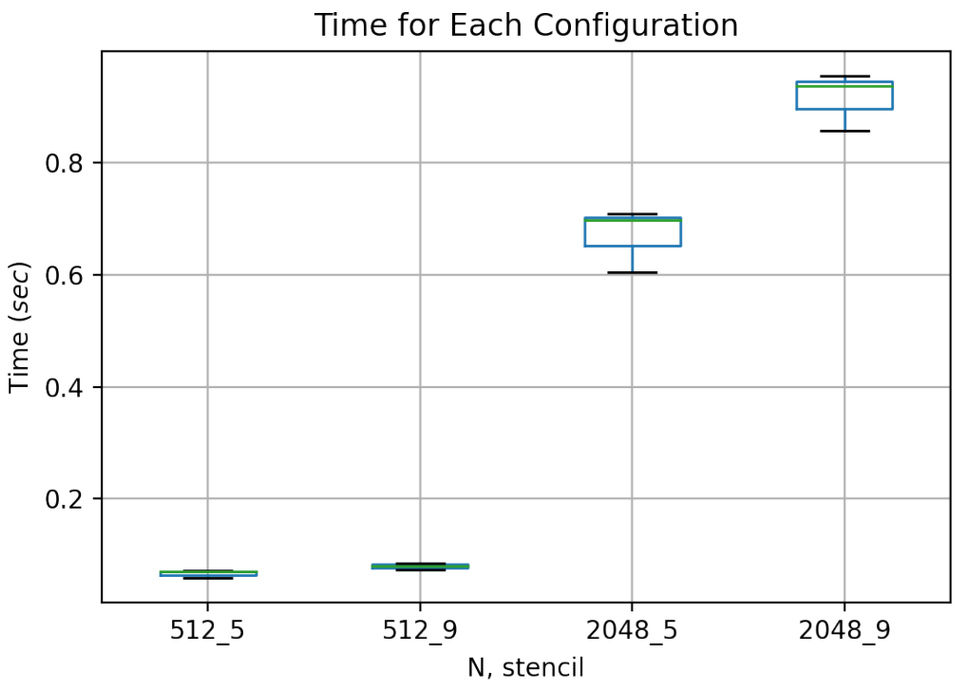

Figure 1: Execution time for each given cases

Observations:

- With increase in data size from 512^2 to 2048^2 we can see the increase in time. This could be mostly due to increase in computation as well as the send/recv buffer size for communication.
- We can see that for increase in stencil for each data size the time is also increased. This is due to increase in communication buffer size. For 2048 case this increase is significant because the relative difference in send/recv buffer size for its 5 and 9 stencil is larger than the relative difference of 5 and 9 stencil of 512 case.