**Name**: Saket Jain **Roll No**: 210905 **Email**: saketj21@iitk.ac.in
**Name**: Shwetank Anand **Roll No**: 211025 **Email**: shwetank21@iitk.ac.in
**Name**: Shri Krishna Sahu **Roll No**: 211011 **Email**: sksahu21@iitk.ac.in

**Group No. 34**

# 1 Explanation for the Code

`src.c` is the source code which performs halo exchange, 5-point, and 9-point stencil computation operations using ISend and IRecv. It takes arguments `N` (number of data points per process), `time steps`, and `mode` (communication using MPI Pack and Unpack).

The program begins by initializing the MPI environment, obtaining the rank and size of the communicator. The size of the matrix ($N$) and the number of steps (`steps`) are then obtained from the command-line arguments. The program proceeds to allocate and fill the matrix with random values as dictated in the question. The position of the current process in the grid of processes is determined, and flags are set to indicate the presence of neighboring processes above, below, to the left, and to the right.

The program then enters a loop, which performs the iteration for the specified number of steps. Within the loop, each process sends its boundary data to its neighboring processes and then receives boundary data from its neighbors. This data exchange is accomplished using MPI's non-blocking send (`MPI_Isend`) and receive (`MPI_Ireceive`) functions. The program then performs the update step, using the received boundary data to update the values of the matrix.

Finally, the program calculates the maximum time taken by any process and prints the result. This allows for an analysis of the program's performance, taking into account the communication overhead from the message passing.

## 1.1 At the Sender's Side

The program uses `MPI_Pack` to pack the data to be sent to the neighboring processes. This data is packed into a buffer, which is then sent to the neighboring process using `MPI_Isend`. The arguments to `MPI_Pack` are the data to be packed (`N_matrix`), the number of elements to be packed (`N`), the datatype of the elements (`MPI_DOUBLE`), the buffer to pack the data into (`send_up`, `send_down`, `send_left`, `send_right`), the size of the buffer, and the position in the buffer where the next pack operation will start. `MPI_Isend` is used to initiate a non-blocking send operation.

## 1.2 At the Receiver's Side

The program uses `MPI_Ireceive` to initiate a non-blocking receive operation. The arguments to `MPI_Ireceive` are the buffer to receive the data into (`recv_up`, `recv_down`, `recv_left`, `recv_right`), the number of elements to be received, the datatype of the elements (`MPI_PACKED`), the rank of the sending process, and the request handle. The request handle is used to check the status of the receive operation later.

`MPI_Unpack` is used to unpack the received data from the buffer. The arguments to `MPI_Unpack` are the received buffer, the number of elements to be unpacked, the datatype of the elements (`MPI_DOUBLE`), the buffer to unpack the data into (`up`, `down`, `left`, `right`), the size of the buffer, and the position in the received buffer where the next unpack operation will start.

# 2 Optimizations and Observations in Stencil Operations

In our computational code, we've implemented several optimizations to enhance efficiency, particularly in stencil operations. These optimizations are detailed as follows:

- **Utilization of Integer Array for Denominator**: Instead of using a double array, we have opted for an integer array to store denominators. This choice allows us to optimize the utilization of memory space within our code and exploit the use of implicit type casting. Use of this extra space allowed us to cut on non-redundancy.
- **Optimized Communication for 9-Point Stencil**: To streamline the 9-point stencil operation, we've devised a method where we combine two rows or two columns together during data transmission. By doing so, we achieve a significant reduction in communication overhead, enabling us to perform the entire 9-point stencil operation with just a single send and receive operation.
- **Observations on Execution Times**: Through performance analysis, we've observed that the *execution times for both the 9-point stencil and the 5-point stencil operations are comparable.* Furthermore, we've noted that these execution times exhibit a *proportional and significant increase as the size of the dataset grows which is prominent for larger datasets when compared between* $512^2$ *and* $2048^2$.

These optimizations collectively contribute to the overall efficiency and scalability of our computational framework. The same can be illustrated through the box plots in the next section illustrated extensively.

# 3 Timing Plots (Box Plots)

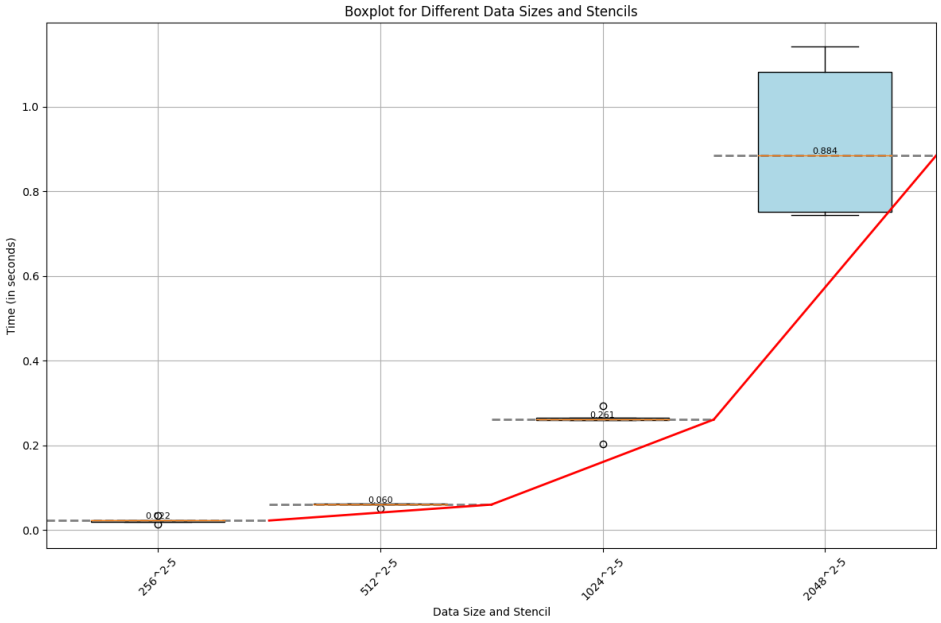


Figure 1: 5-point stencil compared for $256^2$, $512^2$, $1024^2$, $2048^2$ sizes

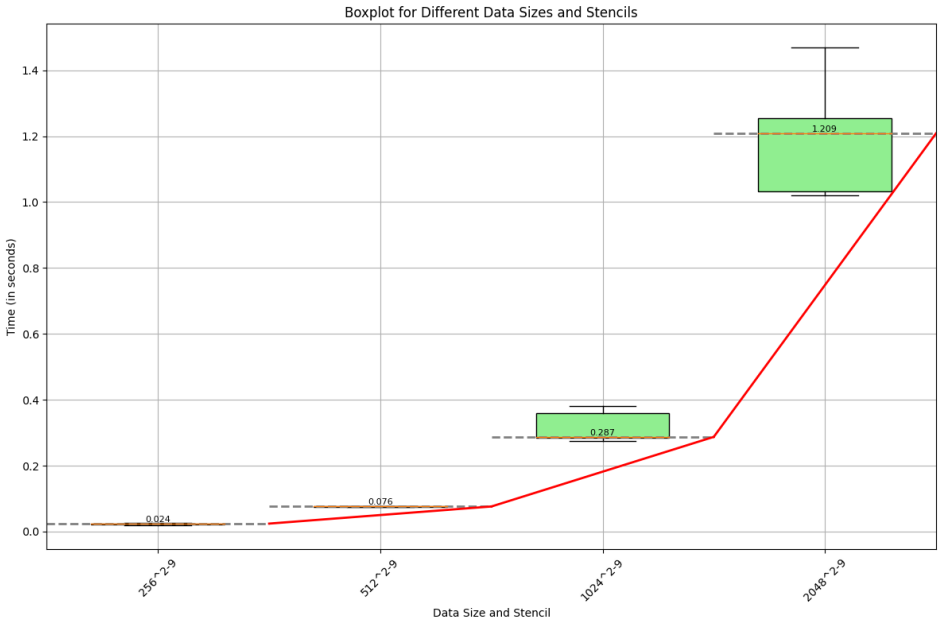


Figure 2: 9-point stencil compared for $256^2$, $512^2$, $1024^2$, $2048^2$ sizes

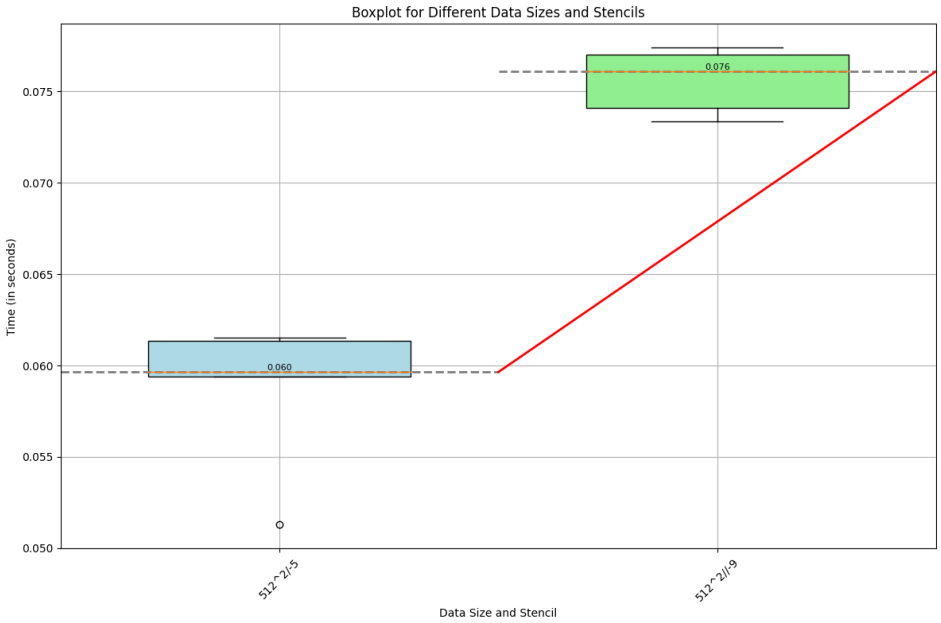


Figure 3: 5-point and 9-point stencil compared for $512^2$ size

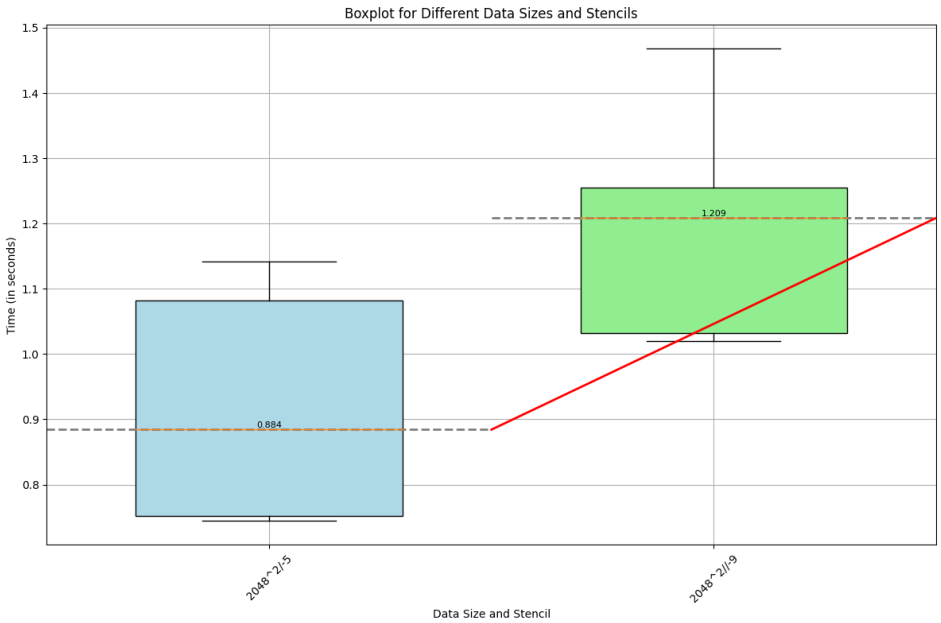


Figure 4: 5-point and 9-point stencil compared for $2048^2$ size