Name: Divya Gupta
Roll No.: 210353
Name : ShabadPreet
Roll No.: 210954

Assignment-1

CS633: Parallel Computing
Group-1
Course Instructor: Prof. Preeti Malakar

---

1. Execution of code

Group-1.zip contains following files:

- src.c : Source code for performing Halo exchanges for the given domain sizes - N over Stencils- 5 and 9
- Datafile.txt : Contains 3*4 datapoints/time(sec) given as input to boxplot.py, where src.c executes three times, for $N = 512^2, 2048^2$, where for each N, it runs for 5-stencil and 9-stencils.
- boxplot.py: Generates box plot(Fig 1), for time(sec) vs (N,Stencils)
- readme.pdf: Contains information about source code, observation and results.

2. Code Explanation

1. **Flags Initialization**:
   - `lflag`, `rflag`, `uflag`, and `dflag` are initialized based on the rank of the current process (`myrank`) and the grid topology.
   - These flags indicate whether the current process has neighboring processes on its left, right, up, and down directions, respectively.
2. **Sending Data**:
   - If there is a neighboring process on the left (`lflag` is true), the data from the left boundary of the grid (`data[i][0]`) is packed into the `send_left` buffer using `MPI_Pack`.
   - Similarly, data from the right, up, and down boundaries are packed into `send_right`, `send_up`, and `send_down` buffers, respectively.
3. **Asynchronous Sending**:
   - After packing the data, the processes initiate non-blocking sends (`MPI_Isend`) to their neighboring processes using `MPI_Send`.
4. **Receiving Data**:
   - If there is a neighboring process on the left (`lflag` is true), the process receives data from its left neighbor (`myrank-1`) into the `recv_left` buffer using `MPI_Recv`.
   - Similarly, data is received from the right, up, and down neighbors into `recv_right`, `recv_up`, and `recv_down` buffers, respectively.
5. **Unpacking Received Data**:
   - After receiving the data, it is unpacked from the receive buffers using `MPI_Unpack`.
   - The unpacked data is stored in arrays (`left`, `right`, `up`, `down`) representing the boundary values from neighboring processes.
6. **Computations with Halo Data**:
   - After exchanging boundary data with neighboring processes, the local computation function (`computeSmol` or `computeBig`) is called, passing the boundary data arrays (`left`, `right`, `up`, `down`) along with the local data array (`data`).
   - This function performs computations involving both the local and boundary data.
7. **Looping and Timing**:
   - The process repeats these steps for a specified number of `steps`.

- Timing information is recorded to measure the execution time of the computation.

Overall, this part of the code segment demonstrates the implementation of halo exchange, where boundary data is exchanged between neighboring processes using MPI communication functions (`MPI_Send` and `MPI_Recv`), and the exchanged data is utilized in local computations.

Function– computeSmol:

- Performs stencil computation for 5-point stencil for next time step.
- double* up, double* down, double* left, double* right - are inputs for computeSmol, contains recieved and unpacked data from other processes.
- Breakdown of code:

– Corner Cases:If the current point is at the top-left corner (i=0, j=0), the stencil value is computed using the available neighboring points and the point itself. The denominator (denom_corner) is adjusted based on the availability of neighboring points (3 if both up and left are NULL, 4 if either up or left is NULL, and 5 otherwise).Similarl for the topright (i=0, j=N-1), bottomleft (i=N-1, j=0), and bottom-right (i=N-1, j=N-1).

– Boundary Rows and Columns:For points along the top row (i=0, j=1 to N-2), the stencil value is computed using the available neighboring points (left, right, bottom, and the point itself). The denominator (denom_b) is set to 4 if any of the neighboring points (up, down, left, or right) is NULL, and 5 otherwise.Similarly, the bottom row (i=N-1, j=1 to N-1), left column (i=1 to N-2, j=0), and right column (i=1 to N-1, j=N-1) are handled in the same way, with the appropriate adjustments to the denominator.

– Interior Points:For interior points (i=1 to N-1, j=1 to N-1), the stencil value is computed using the standard 5-point stencil formula, which takes the average of the point itself and its four immediate neighbors.

After computing the stencil values in the temp_matrix, the function copies the values back to the original data array for further computations or communication.

ComputeBig :

- Performs matrix computation for a 9-point stencil for each time step.
- It takes a temporay matrix to store the new values for next time step along with original data matrix, *up* matrix containing last 2 rows from above processor if there exist a processor above it, similarly *left, right* and *down* matrix from left, right and below processor respectively.
- Breakdown of Code:
  - Corner cases: There will be 4 matrices of 2x2 containing corner cases at each corner, computation of each corner matrix is done seperately in code.
  - Boundary rows and columns were also calculated seperately.
  - Interior points: For interior points (i=2 to N-2, j=2 to N-2), the stencil value is computed using the standard 5-point stencil formula, which takes the average of the point itself and its four immediate neighbors

3. Observations and Results:

3.1. Observation

- The timing values for the larger data size (4194304.0) are significantly higher than those for the smaller data size (262144.0), as expected, since the computation and communication costs increase with the problem size.
- For both data sizes, the 9-point stencil configuration takes longer to execute compared to the 5-point stencil configuration. This is also expected since the 9-point stencil involves more neighboring points in the computation, leading to increased computational complexity.
- The spread (the height of the boxes) of the timing values is relatively small for the smaller data size (262144.0), indicating that the variation in execution times across different runs is minimal.
- For the larger data size (4194304.0), the spread of the timing values is more prominent, especially for the 9-point stencil configuration, suggesting a higher degree of variation in execution times across different runs.
- The median timing values (represented by the horizontal lines within the boxes) align well with the overall trend, being lower for the smaller data size and the 5-point stencil configuration.

Name: Divya Gupta
Roll No.: 210353
Name : ShabadPreet
Roll No.: 210954

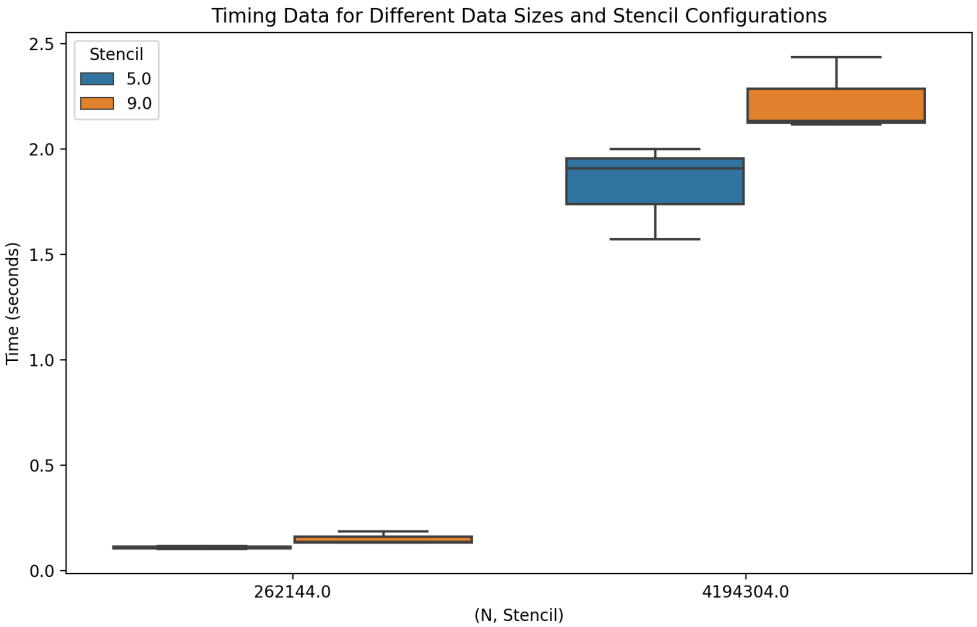

Figure 1 the timing plot

3.2. Results

As expected, larger data sizes and more complex stencil configurations lead to higher execution times. The 9-point stencil consistently takes longer than the 5-point stencil due to its increased computational complexity. Additionally, the variation in execution times is more pronounced for larger data sizes, especially with the 9-point stencil, suggesting the influence of runtime factors such as system load and network conditions. These insights highlight the trade-offs between data size, stencil complexity, and performance, providing valuable information for optimizing the algorithm and allocating resources effectively in parallel computing applications involving stencil computations and halo exchange operations.