CS633: Assignment 1
Group 14

Chitwan Goel
chitwang21@iitk.ac.in
210295

Aniket Suhas Borkar
aniketsb21@iitk.ac.in
210135

Shrey Bansal
shreyb21@iitk.ac.in
210997

1 Data Structures

1.1 Data Arrays

We have used two $2-D$ data arrays `data` and `data_avg` to store the data for each process. At each time step, the processes use `data` array as the current data and store the average values after stencil computation in the array `data_avg`. The arrays are swapped after each iteration so the processes can use the new data.

1.2 Buffers

We have used 4 buffer send arrays `sendtop`, `sendbot`, `sendleft` and `sendright`, and 4 buffer receive arrays `recvbot`, `recvtop`, `recvleft`, `recvright` for the data exchange in interprocess communications.

1.3 Variables

There are various arguments parsed from the command line which is as follows:

1. `P_x`: The number of rows in the process placement matrix.
2. `N`: The matrix size for each process, i.e., $N = n^2$ where n is the number of rows and columns in the matrix.
3. `timesteps`: The number of times we have to run the stencil.
4. `seed`: Used for random number generation purposes.
5. `stencil`: For 5 or 9 point stencil.

Besides this, we have used some other variables, which are explained as follows:

1. `P_y`: The number of columns in the process placement matrix
2. `n`: The number of rows and columns in the data matrix of each process
3. `send_recv_size`: The size of the send and receive buffers. It equals n or $2n$ for 5 and 9-point stencils, respectively.
4. `size_pack`: The size of the buffer used in `MPI_Pack`.

2 Communication

We iterate over the `num_time_steps`. In each iteration of the loop, there are 4 `if-else` blocks, which are explained below:

1. `if (rank` $\geq$ `P_y)` - In this block, we are packing the topmost 1 row in case of 5-point-stencil and the top 2 rows in case of 9-point-stencil to send to the upper process. After sending, we have an `MPI_Recv` call, which receives the bottom-most row(s) from the upper process. This condition ensures that this code is not executed for the processes in the top-most row of the process placement matrix.

2. `if (rank + P_y <` $<$ `size)` - In this block, we are packing the bottom-most 1 row in case of 5-point-stencil and the bottom 2 rows in case of 9-point-stencil to send to the lower process. We have an `MPI_Recv` call, which receives the bottom-most row(s) from the upper process. After the receive call, we have an `MPI_Send` call, which sends the packed data to the lower process. This condition ensures that this code is not executed for the processes in the bottom-most row of the process placement matrix.

3. `if (rank % P_y` $\neq$ `0)` - In this block, we are packing the leftmost 1 row in case of 5 point-stencil and left 2 rows in case of 9 point-stencil to send to the left process. After sending, we have an `MPI_Recv` call, which receives the left-most row(s) from the right process. This condition ensures that this code is not executed for the processes in the left-most row of the process placement matrix.

4. `if (rank % P_y` $\neq$ `P_y - 1)` - In this block, we are packing the right-most 1 row in case of 5 point-stencil and right 2 rows in case of 9 point-stencil to send to the right process. We have an `MPI_Recv` call, which receives the right-most row(s) from the left process. After the receive call, we have an `MPI_Send` call, which sends the packed data to the right process. This condition ensures that this code is not executed for the processes in the bottom-most row of the process placement matrix.

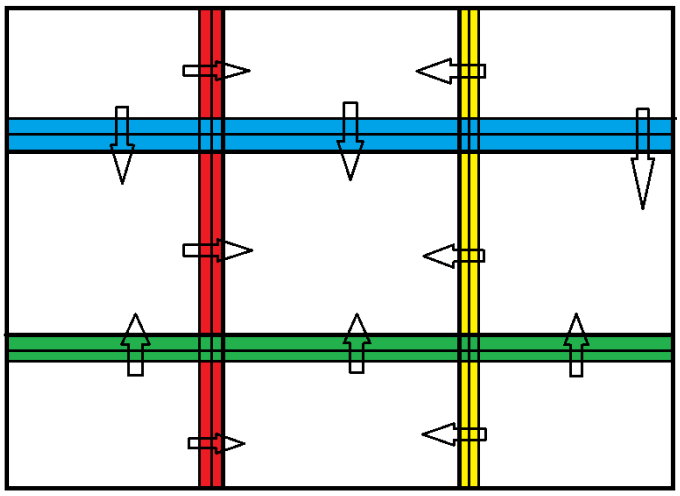


Figure 1: Arrows show communications between the processes and the coloured regions show the data to be exchanged. Communications also occur in the reverse direction of those shown in figure.

3 Computation

1. `count` - We have iterated over the grid points in two loops, one for columns (`j`) and one for rows (`i`). To account for the number of grid points used to compute the average, we have initialized the `count` variable to `stencil` and subtracted the number of points that cannot be accessed for that particular row and column. For the (left/top/right/bottom) most grid points of the processes in the (left/top/right/bottom) most columns, we need to decrease the count by 1 and 2 in the case of 5 and 9-point stencils.

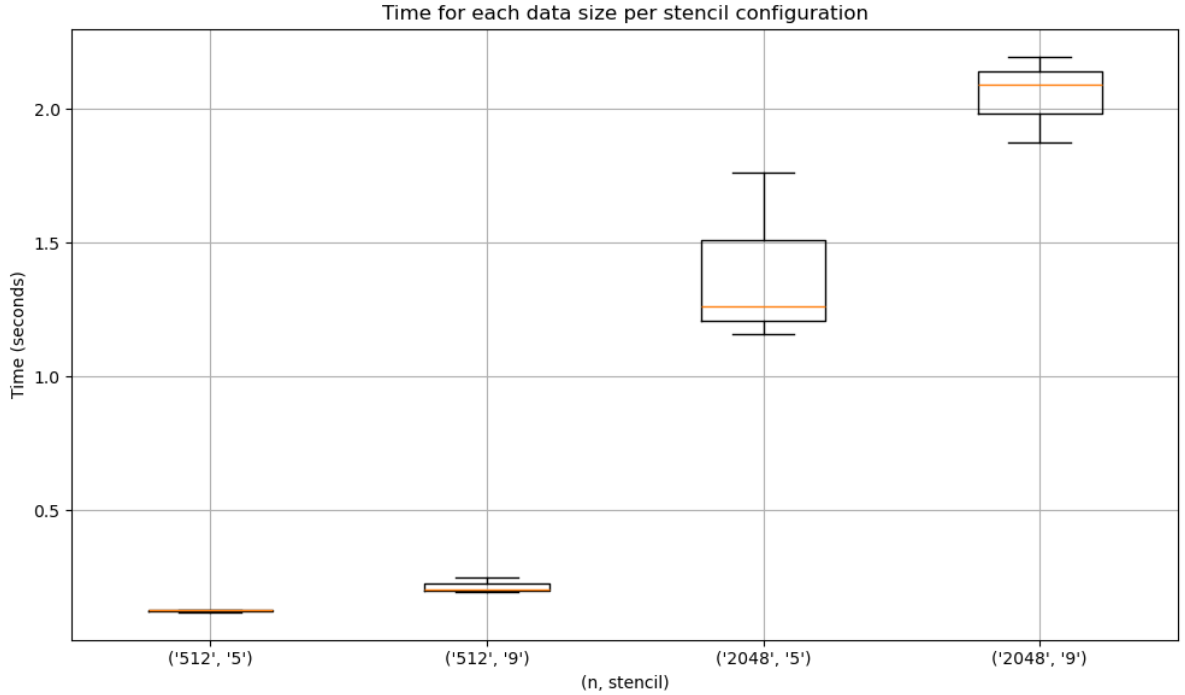
2. `getval()` - Given `i,j` and `data`, it returns `data[i][j]`. If i or j are out of index, it returns the value from the suitable buffer. For example, for an element in the topmost row of the matrix, there is no upper neighbor. So when we do `getval(i-1, j)`, it gives the jth element from the data received from the upper process (if it exists, else 0 - as the receive buffer is initialised to 0, and never changed if no data is received).

3. `sum` - This stores the sum of values for each grid point `data[i][j]`.

4. `data_avg` - This array is updated by averaging `sum` over `count` i.e., `data_avg[i][j] = sum / count`.

4 Results

The data obtained is tabulated in `data.csv` file. We have used `python` to get the box-plot, the code for which is in `plot.py`.



4.1 Observations from the above Boxplot

1. The time increases with an increase in data size, which indeed should be the case as larger data size implies larger communication and computation.

2. For the same value of $N(\text{or } n)$, the time taken for 9-point stencil is more than that for 5-point stencil. This is perhaps because in 9-point stencil, the size of data for each communication is double of that for 5-point stencil.