CS633-Assignment1, Group 3

Deven Gangwani (210327, devenag21@iitk.ac.in)
Geetika (210392, geetika21@iitk.ac.in)
Talin Gupta (211095, taling21@iitk.ac.in)

March 21, 2024

1 Code explanation

The code first takes command-line arguments. Py is calculated from P and Px. Then, random initialization is performed by each process, for the data in its domain, using the given seed. Tags used for all communications are process ranks.

1.1 Data arrays

data_old and data_new are 2D matrices, which represent the data a process has. These 2 matrices ensure that at every time step, we are using values from previous time step for the calculations. At the end of every time step, data_old is updated with the values from data_new. The count matrix stores for each entry, what are the number of neighbours it has interacted with(through communication and computation in same process) + 1 (it's own old value is also seen when calculating new average).

1.2 Different send and receive buffers

We have used different buffers for sending and receiving to and from the processes which are to the left, right, above, and below in virtual topology. The names of the buffers indicate which communication they are meant for(recvbuf_left is for the data coming from process on the left in virtual topology)

1.3 Communication: sending to process to the right and receiving from the process to the right

We first send data(from the last column/last two columns) to process in the right(if it exists). More specifically we check if the process is not the last process in the row ($myrank\%Py! = Py - 1$), the last process in the "row" won't have anyone to send and receive data from . Afterward, we pack the column data points, for the 9-point stencil we pack the last column first, and then the second last column. Then we send this packed data to the process on the right and then receive the first column or first two columns from the process on the right.

1.4 Communication: receiving from the process to the left and sending to the process to the left

We receive the last column/last two columns from the process on the left(if it exists). More specifically we check if the process is not the first process in the row ($myrank\%Py > 0$) first process can't be received and sent to the left. Afterward, we pack the column data points, for the 9-point stencil we pack the column first, and then the second column. Then we receive data last column/last two columns from the process on the left and send the first column or first two columns to the process on the left.

1.5 Communication: sending to and receiving from the process below

Once communications between processes in the same row in virtual topology have been done, the bottom 2 rows(9 point) or 1 row (5 point) are sent to the process below current process. We have used the condition $(myrank/Py) < Px - 1$ to ensure that there are processes present below in the virtual topology. We first send to and then receive from the process below.

1.6 Communication: sending to and receiving from the process above

Next, we communicate with the process above, using the condition $(myrank/Py) > 0$ to ensure that it is not the first row. Here, we receive first and then send, so our code is a safe one.

1.7 Computations within the same process

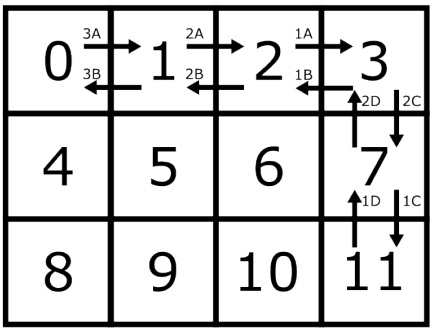
We next perform computations that need data points within the same process. That is, to a cell, we add values from its neighboring data points in the same process. We have implemented this using if conditions. Note that the condition for checking stencil as 9 is outside for loop to avoid executing that instruction everytime the loop runs.

1.8 Computations involving data received from other processes

These computations are performed after relevant receive buffers from them are filled, that is, when the receive call gets completed. This has been done in the corresponding if blocks for the communications. 9 point and 5 point stencil have been dealt with separately wherever required.

1.9 Overall view of the communication

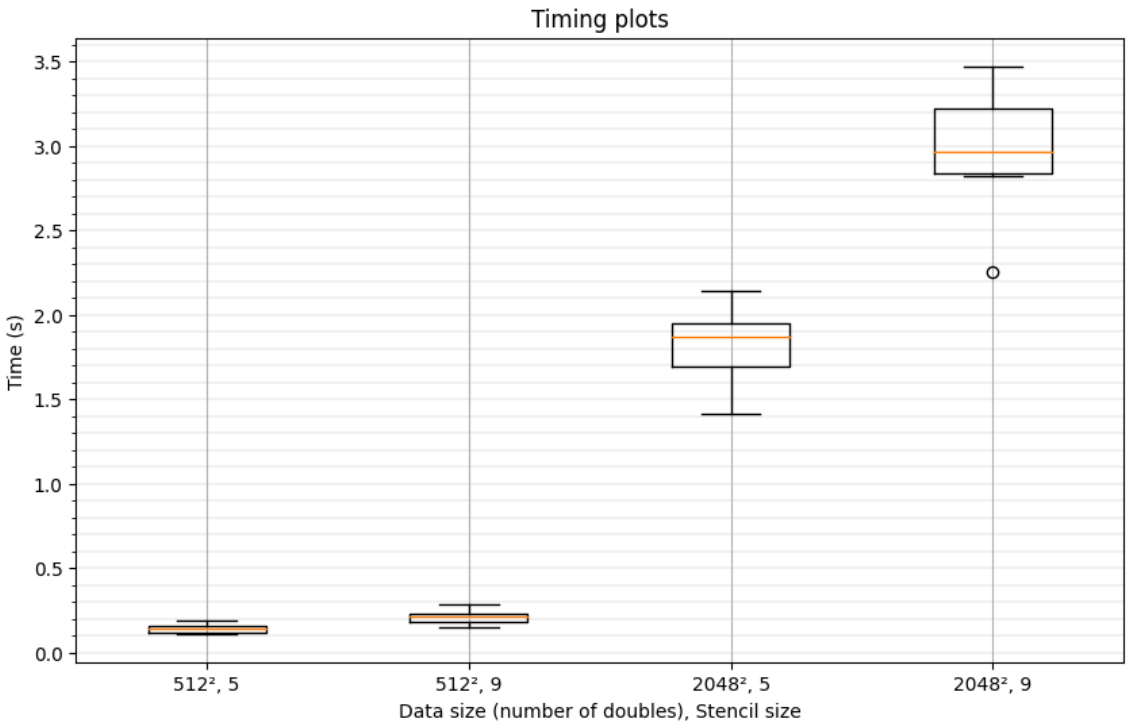
The following diagram shows the order in which communications take place, as per our code.



The diagram above shows how communications happen for P = 12, Px = 3. The first communication is process 2 to 3, occurs when matching sends, receives posted. The communications in order are 1.A, 1.B, 2.A, 2.B , 3.A, 3.B where each arrow shows data sent from sender to receiver. The sends and receives have been posted in such an order that avoids deadlocks. Note that we have shown communications for 1 row and 1 column only. Communications in columns occurs after communication in rows. For communications within a column, the order is 1.C, 1.D, 2.C, 2.D

2 Observations

We used timing data from 9 iterations and got the following timing box plot:



We note that :

1. For larger data size, more time is taken. This is because more data needs to be transferred during communication, and more computations are executed while calculating the stencil average for each point.
2. For the same data size, the 9-point stencil takes more time than the 5-point stencil as, in the 9-point stencil, double the data is communicated (two rows/columns in the place of one). The difference in times is more for larger data, as expected.
3. More variance in the timings is observed in the case of larger data.
4. Between the two stencil sizes, for the smaller data size, a difference of approximately 0.1 seconds is observed. For the larger data size, the difference is 1-1.5 seconds.
5. Data size is a more dominating factor than whether 5 or 9 point stencil is used. This is expected also because the different in data size is very large.

3 Optimizations used

We initially were putting if-conditions inside for-loops (for stencil-specific operations - for example, when computing averaged stencils for each data point). To avoid this redundancy, we took the if condition outside to prevent it from being checked on every iteration. Also, for the 9-point stencil, we used just 1 MPI_Send to communicate data between relevant processes, after packing the data.

4 Data file

In the data file, we have shown the timing data recorded for nine different executions. Our plots were constructed using this data.