CS633-Assignment2, Group 3

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

April 15, 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.
We first have the code for with leader case, followed by without leader case(same as last assignment). The following explanation is mainly for the communication with leader, as without leader code is the same as before.

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).
For the case with leader, additional buffers have been used. These are:

- leader_scatter_buffer : used by leader for scattering received data to other processes in the new communicator
- leader_gather_buffer : used by leader for gathering data from other processes in the new communicator

1.3 Subcommunicators

The processes on the same node(Or row in virtual topology) are a part of the same subcommunicator. We have used MPI_split for making the subcommunicators. The color is myrank%Px. Hence, the leader is first process in the row In process virtual topology. For example, if Px=4 then processes 0,1,2,3 are placed on the same node.

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

We first send data(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 ($myrank\%Px! = Px - 1$), 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.

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

We receive the 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 ($myrank\%Px > 0$) 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.

1.6 The Gather Calls

Each process in a row in the virtual topology(that is, each process in a node) sends its first 2 data rows and last 2 data rows to the leader process(two separate gather calls).

1.7 Communication: sending to and receiving from the process below

Only the leaders(recognized by their newrank == 0) perform this step. The condition to check that their low isn't the last is $(myrank/Px) < Py - 1$. They communicate with the leader of the row of process lying below their row in the process virtual topology. These communications happen through leader_

1.8 Communication: sending to and receiving from the process above

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

1.9 The Scatter Call

The leader_scatter_buffer is the buffer from which data is scattered, and is received in recv_buffer of the various processes.

1.10 Computations within the same process

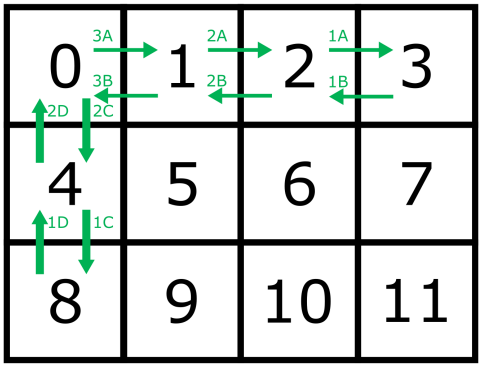
We next perform computations that need data points within the same process. That is, to a cell, we add values from its neighboring data points in the same process. We have implemented this using if conditions. Note that the condition for checking stencil as 9 is outside for loop to avoid executing that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

1.11 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.

1.12 Overall view of the communication

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



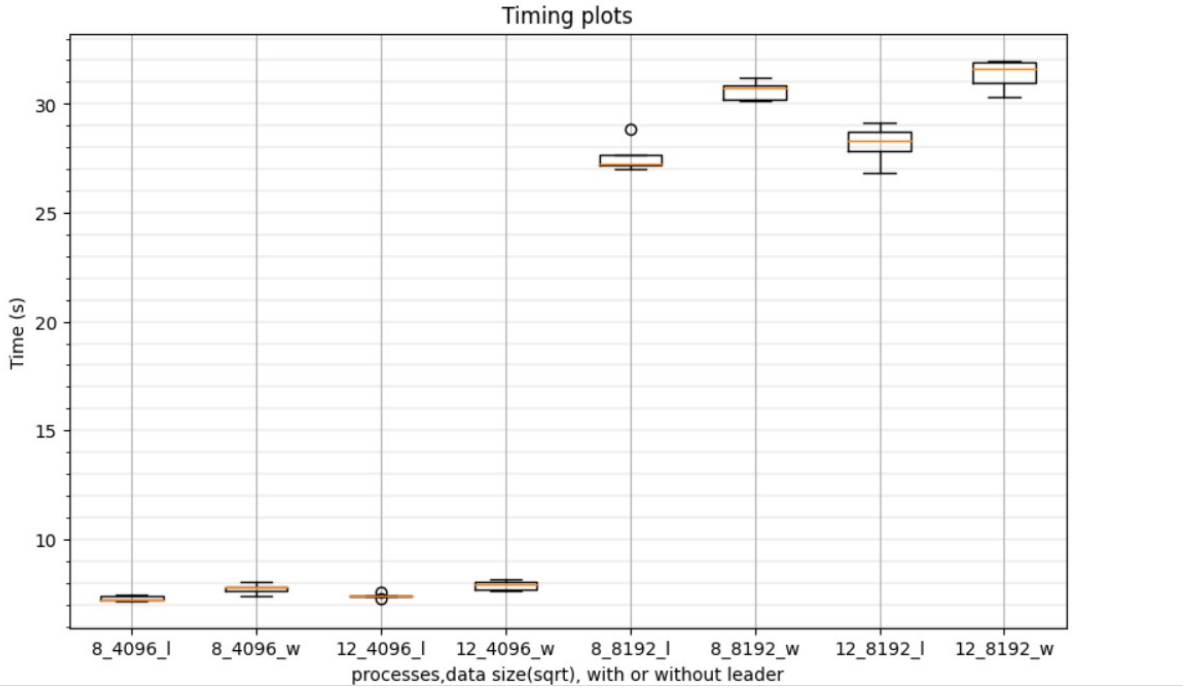
The diagram above shows how communications happen for P = 12, Px = 3.

In the "with leaders" case, first communication happens between processes in the same "row", as can be seen by the thin arrows. Sends are posted to the right first, and then receives are posted, so that the first communication (in the first row) happens between processes #2 and #3. After this, the leader of each row (i.e. the first process, #0 for the first row, #4 for the second row and #8 for the third) issues a gather call, and then communicates with the other leaders, as shown by the thick arrows. Finally, after the data has been spread, the leaders issue scatter calls to spread the data to their rows.

In the "without leaders" case, data is shared between processes as in the previous assignment: processes communicate with each other in their rows first, and then with others in their column, in the order according to the numbering of the arrows. In this case the varying thickness of the arrows does not denote anything.

2 Observations

We used timing data from 5 iterations and number of time steps as 5 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. Communications with leader is found to take less time on average, both for small data and large data. Very few times it happens that for small data, with leader time is more. Even rarer for large data.

3. Communication with leader takes less time : it can be explained by the following reason: The number of inter node communications decreases. In the case without leader, all processes in a row in virtual topology perform inter node communications, but now only the leader does it with a bigger message size. This observation is similar to the paper discussed in class about topology aware collectives. The gather and scatter taking place are all intra-node.

4. For different number of processes(8 and 12), almost same time taken, with very slightly higher time for 12 processes.

5. Data size is a more dominating factor than whether leader is used or not. 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 5 different executions. Our plots were constructed using this data.

5 Contirbution

Each person contributed equally.