Q1(a) : The deadlock-free approach of your solution and its efficiency.

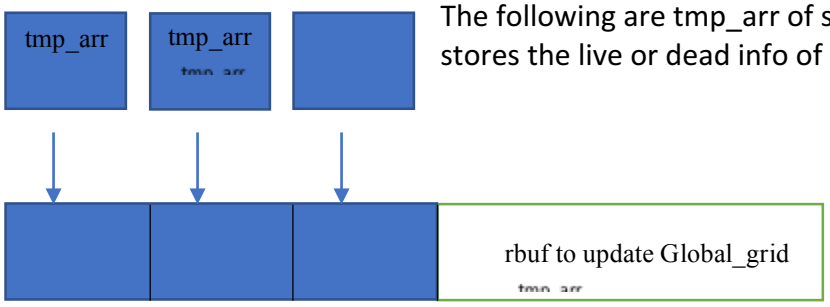
The program follows a compute, gather and broadcast method to create the game of life. Algorithm is describes here as follow :

- For loop over number of iterations
  1. Divide the array into n parts(n=number of processors available)
  2. Each processor calculates the number of neighbors of element of the part of the array that belongs to it (neighbors seen from global_grid array)
  3. Each process make changes to its own part of the array in a buffer list tmp_arr of size 256/num_procs.
  4. MPI_Gather collects tmp_arr from all processes and sends them to process 0
  5. Global_grid is updated by the gathered tmp_arr from each process(through mapping each element).
  6. Process 0 Broadcasts new Global_grid to all the processes

The communication structure is as follows :

There are 2 places in the program where the nodes communicate with each other.

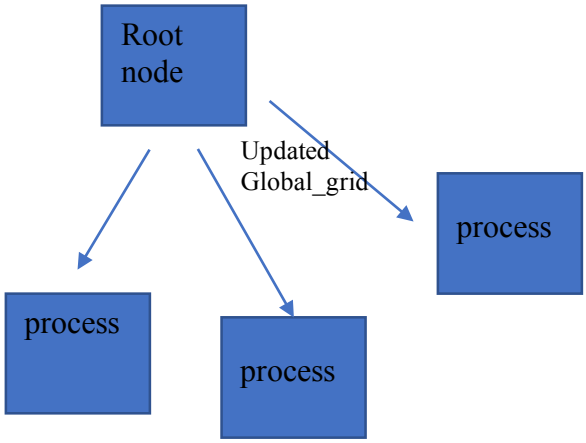
1. MPI_Gather :
   All the processes compute the number of neighbors of each entity in their part of the grid and decide whether the element is alive or dead. They then store the information in a dynamically allocated array (tmp_arr) of size 256/num_procs. MPI_Gather collects tmp_arr's from all processes and sends them to node 0. Since this is a collective process, there is no deadlock in this part.



The following are tmp_arr of size= 256/num_procs. It stores the live or dead info of each cell of the part of grid.

rbuf- collective array sent to root process to update Global_grid

2. MPI_Bcast :
   After the Global_grid is updated with the help of tmp_arr, new updated Global_grid (new world information) is sent to each process. They may now continue to compute the living status of each element in the grid based on the new Global_grid information.



Efficiency :

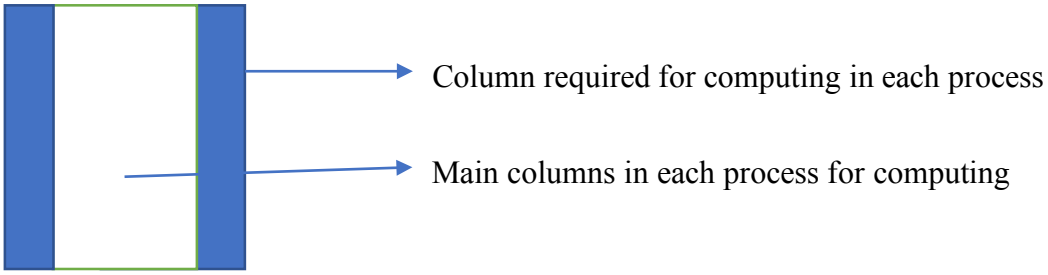
Each node is highly efficient but since there is a collective process (MPI_Gather and MPI_Bcast), there is a skew delay in the parallel process.

Root node is the slowest process so the program time will be determined by the speed of root process.
Root node is doing 2 extra tasks other than computing neighbors (updating Global_grid and printing Global_grid) of **O(DIM^2).** Therefore the program will run by the speed of this slowest process i.e root process.

In terms of **memory**, the program sends **2 collective messages of size 256 int sized memory each(512 Bytes)**. Since the number of messeges as well as the size of messeges is too small, it would be fine. But in case of larger grid and more iterations, there is scope of improvement in this method.

Q b) The dependencies between the decomposed parts of the problem.

The program Grid is decomposed in as many parts as many processes are available. The grid is divided into columns and each column is computed by a different process. Each process needs the information of the adjacent column to the left and right of the grid. This information is provided by the message Broadcast through the root node to all other nodes.



2. Consider an alternative spatial decomposition in which we divide the grid recursively into smaller squares, e.g. an n x n grid may consist of 4 (n/2) x (n/2) squares or 16 (n/4) x (n/4) squares or 64 (n/8) x (n/8) squares. What are the relative advantages and disadvantages of this decomposition when compared with the horizontal slicing of the space in your implementation:

a. On the number and size of MPI messages in the simulation? (Give an expression for each decomposition for the size and number of messages as a function of the number of processes.)

The minimum size of total message required for this implementation in each iteration is $4(n/(num\_procs)^{1/2}+1)$

Since there are 8 surrounding squares around each node, these messges should come in a total of 8 messages. **(4 messeges of length $(n/(num\_procs)^{1/2}+1)$ each+4 messages of length 1 each)**

In the present approach, there are a minimum of 2 messages of length n each. To each process. (**2 messages of length n**)

Thus the new method will take more number of messages and more total message memory transfer as the present.

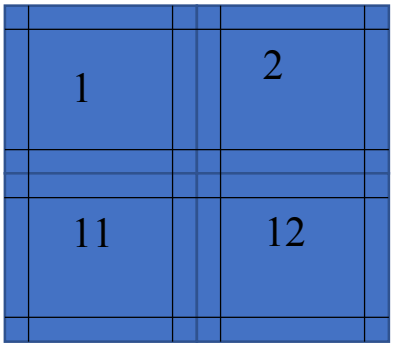
b. On the memory overhead at each processor? (Again, give an expression for each decomposition.)

Each processor will need to store the part of the array and the boundary elements. Thus it will need $(n^2/num\_procs) + 4(n/(num\_procs)^{1/2} + 1)$ memory locations at each process.

c. On the flexibility of decomposition?

We lose flexibility here as the number of possible parts the grid can be divided should be a perfect square number only and the grid DIM should be divisible by the route of the num_procs.. (4, 16, 64…).

d. For the new decomposition, describe a deadlock-free messaging discipline for transmitting the top, bottom, left, and right sides and the top left, top right, bottom left and bottom right corners among neighboring partitions. Pseudocode or a drawing will be helpful in describing the protocol.



Let us divide the grid on similar lines and number each part for simplicity. Each process needs to receive and send 8 messages.

1. First the column communication takes place (i.e messages to left and right processes).
   If process is even
   MPI_Send(column… to right process)
   MPI_Recv(column from left node)
   Else if process is odd
   MPI_Recv(column from left node)
   MPI_Send(column to right node)

   Once this communication is complete, Repeat the process in the reverse direction.
2. Next the communication happens in the vertical direction. On similar lines
   If process/10 is even
   MPI_Send(column… below)
   MPI_Recv(column from above)
   Else if process /10 is odd
   MPI_Recv(column from above)
   MPI_Send(column to below)
   And repeat
3. In the diagonal direction….

We follow a similar ideology and use the number of nodes to pass half the messages first where half nodes receive and half of them send, and reverse the roles when the half communication is completed.