Project 3

Bohao Tang

March 3, 2018

1. Please characterize (describe your answers):

   a. The deadlock-free approach of your solution.

   Answer: In my solution, since nodes number are always even number here (n_proc > 1). I used pair sends and receives. I send necessary message first from even nodes and receive by odd nodes and then send from odd nodes and receive by even nodes. The dependecy graph will then be a bipartite graph, therefore no cycles no dead-lock.

   b. The efficiency of your implementation using big-O notation (define all variables used), in terms of the

   (i). Number of messages passed per process for a single iteration of the game,

   Answer: Let N be the number of processes and D be the grid dim. Then for the first iteration, I first do a data partition, so $\frac{N-1}{N}D^2$ int will be send out from process 0 to $N-1$ others. But when the data partition is done, in each iteration, only $2D$ int will be sent from each process to its adjacent processes for update states. In the last iteration, every process (>0) will additionally send its $\frac{D^2}{N}$ int grid back to process 0. So the answer will vary. If you only do one iteration at least you need to send back updated data, so you do 4 ($O(1)$) times message passing for $O(\frac{D^2}{N})$ int messages per process. If you do a lot of iteration, then in average, in each iteration, 2 ($O(1)$) times for $O(D)$ int messages will be passed per process. If you count the total iteration number T, then the average answer will be $O(1)$ sends and receives for $O(\frac{D^2}{NT}+D)$ int.

   (ii). The amount of memory used per process for a single iteration of the game.

   Answer: Let N be the number of processes and D be the grid dim. Then in each iteration, a process will hold exactly $\frac{D^2}{N}$ int for local grid and $2D$ int for adjacent rows, since we slice rows, N should be less than D. So the answer is $O(\frac{D^2}{N})$.

   c. The dependencies between the decomposed parts of the problem i.e., inter-process data dependencies.

   Answer: We divided data by rows, then each process will need its two adjacent rows in the whole grid to update states. That is, each process will need the last row in its previous process and the first row in its next process. So each process will have data dependencies from its two adjacent process (0 is adjacent to N-1 for N processes).

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 a big-O expression for each decomposition for the size and number of messages as a function of the number of processes.)

   Answer: Denote N be the number of processes and D be the grid dim. Now N should be a squared integer. Then also answer will vary whether we send back data to process 0 after each update. If we send back data, then this amount will dominant, we send 9 (thus $O(1)$) messages and in total $O(\frac{D^2}{N})$ int for each decomposition each iteration. If not, in the middle of iterations, we send 8 (thus $O(1)$) messages and in total $O(\frac{D}{\sqrt{N}})$ int for each decomposition each iteration.

b. On the memory overhead at each processor? (Again, give a big-O expression for each decomposition.)

   Answer: Denote N be the number of processes and D be the grid dim. Each process will need exactly $\frac{D^2}{N} + 4\frac{D}{\sqrt{N}} + 4$ int to hold for necessary data. Therefore $O(\frac{D^2}{N})$.

c. 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.

   Answer: Let $N = n^2$ be the number of processes. Let function `Send(data,dest)` be a simplified notion for sending `data` to `dest` and function `Recv(source)` be reciving some data from `source`. We basically just need to communicate by pair between even nodes and odd nodes to gain the deadlock free property. Here is the pseudo mpi code(Let `ID` be the process id):

```
// Preparation
i = ID / n;
j = ID % n;

// Find eight direction adjacent processes just as the arrow
← = i * n + (j - 1) mod n;
↑ = (i - 1) mod n * n + j;
→ = i * n + (j + 1) mod n;
↓ = (i + 1) mod n * n + j;
↖ = (i - 1) mod n * n + (j - 1) mod n;
↗ = (i - 1) mod n * n + (j + 1) mod n;
↘ = (i + 1) mod n * n + (j + 1) mod n;
↙ = (i + 1) mod n * n + (j - 1) mod n;
```

```
// Begin messaging
// Send Receive Top and Bottom
if(ID % 2 == 0){
    Send(bottom, ↓);
    Recv(↓);
    Send(top, ↑);
    Recv(↑);
}
```

```
else{
    Recv(↑);
    Send(top, ↑);
    Recv(↓);
    Send(bottom, ↓)
}

// Send Receive left and right
if(ID % 2 == 0){
    Send(left, ←);
    Recv(←);
    Send(right, →);
    Recv(→);
}

else{
    Recv(→);
    Send(right, →);
    Recv(←);
    Send(left, ←)
}

// Send Receive top left and bottom right
if(ID % 2 == 0){
    Send(bottom right, ↘);
    Recv(↘);
    Send(top left, ↖);
    Recv(↖);
}

else{
    Recv(↖);
    Send(top left, ↖);
    Recv(↘);
    Send(bottom right, ↘)
}

// Send Receive top right and bottom left
if(ID % 2 == 0){
    Send(bottom left, ↙);
    Recv(↙);
    Send(top right, ↗);
    Recv(↗);
}

else{
    Recv(↗);
    Send(top right, ↗);
    Recv(↙);
    Send(bottom left, ↙)
}
```