1.a: The deadlock-free approach of your solution and its efficiency.

Answer:
In my approach, at the beginning the original array is sliced into several regions horizontally according to the number of processors. Then in each iteration, each processors will send its boundaries to its neighbors in order to update data. I force the processors with even IDs to send first so that there are no circular dependencies(pair sends and receives). To guarantee the efficiency, my approach try to avoid any unnecessary message passing. Message passing in my approach occurs only when:

1. Passing the boundaries of the region to its neighbors.
2. Passing the updated region to the processor with ID 0 at the end of each iteration.

Therefore at the end of each iteration, the complete array in processor 0 will be updated and printed. The local slices in each processors will be reused for the next iteration.

1.b: The dependencies between the decomposed parts of the problem.

Answer:
Each decomposed part has dependency on the neighbors above and below it. Because the update procedure requires the boundary information from these two neighbors and that's why we need message passing.

2.a: On the number and size of MPI messages in the simulation?

Answer:
Assuming that the grid is $n \times n$ and the number of processors is m:

1. Horizontal decomposition:
   - Number of MPI messages between neighbors in each iteration: $2m$.
   - Size of MPI messages between neighbors in each iteration: n.
   - Number of MPI messages to the first processor in each iteration: $m - 1$.
   - Size of MPI messages to the first processor in each iteration: $\frac{n^2}{m}$.
2. Spatial decomposition:
   - Number of MPI messages between neighbors sharing a side: $4m$.
   - Size of MPI messages between neighbors sharing a side: $\frac{n}{\sqrt{m}}$.
   - Number of MPI messages between neighbors sharing a corner: $4m$.
   - Size of MPI messages between neighbors sharing a corner: 1.
   - Number of MPI messages to the first processor in each iteration: $m - 1$.

- Size of MPI messages to the first processor in each iteration: $\frac{n^2}{m}$.

Therefore the disadvantage of spatial decomposition is that the number and total size of messages are larger than horizontal decomposition which means it takes more time for messages passing. On the other hand, the advantage is that it can divide the task into more smaller tasks so as to solve them quickly. This is important when the size of task is very big and we want more processors to help us handle this task.

2.b: On the memory overhead at each processor?

Answer:
Assuming that the grid is $n \times n$ and the number of processors is m:

1. Horizontal decomposition: In my approach, each processor has an array of length $\frac{n^2}{m}$ for storing the slice and four arrays of length n(two for sending messages and two for receiving messages).

2. Spatial decomposition: For each processor, there should be an array of length $\frac{n^2}{m}$ for storing the square. And we also need eight arrays of length $\frac{n}{\sqrt{m}}$ and eight variables(four for sending messages and four for receiving messages).

2.c: On the flexibility of decomposition?

Answer:
Obviously the spatial decomposition is more flexible because it can handle different shapes of grid. For instance, if we have a $1 \times n$ grid, the horizontal decomposition can't divide it into multiple tasks while the spatial decomposition can.

2.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.

Answer:
We can reduce the two dimensions message passing into one dimension by using four loops. Firstly, if we only consider passing messages horizontally between squares sharing a side, then the problem is very similar to horizontal decomposition. We can force the squares in even columns to pass messages first in order to prevent deadlock. Similarly, we can pass messages vertically in such an approach too. Also, we can pass messages between squares sharing a corner in diagonal and anti-diagonal directions using other two loops. Hence we divide the original problem into four subproblems which can be solved using our experience from horizontal decomposition.