Tips for 1D stencil:

The 1D (and 2D) stencil version requires to think about how processes will exchange their borders values.
At the beginning, processes need the borders values of their neighbors to start computing. A way of coding this would like this, given for a single process k:

- process k allocates memory for his own values, as well as a buffer for the values he will get from his neighbors

For each time step:

- process k exchanges borders values with its neighbors k-1 (except if k=1) and k+1(except if k is last process)
- process k updates his own values using these values and its neighbors values
- All processes send their final values to process 0 who can then display the solution

End for

- End of code

 When 2 neighboring processes exchange values, you should find a way for them to do that so that they don't end up both sending their values and waiting for the other process to receive them.

A solution to handle values exchanging:

An effective way of solving this issue is to separate processes in two groups, the "odd" processes (ranked 1,3,5...) and the "even" processes (ranked 2,4,6..). Then, each odd process (except eventually for first and last one) send its border to the even process below, then send its other border to the even process above. Meanwhile, each even process wait for data from the odd process above, then waits again for data from the odd process below. Finally, odd and even processes can switch roles so that everyone has received the data needed.

This can be generalized in the case of a 2D stencil. This time, you would have to separate processes in two groups again, just like you separate black from white tiles on a chessboard.