Quiz 3 answers – Aditya Shankar (5454360)

1. No, we cannot use the same partitioning (such as striped partitioning) to parallelize both Gauss-Seidel and Jacobi. This is because Gauss-Seidel has data-dependencies that restrict the parallel implementation.

   In Jacobi method, each iteration requires the values of the neighboring points from the **previous iteration only**. i.e., $U^{m+1}(i,j) = (U^m(i-1,j) + U^m(i+1,j) + U^m(i,j-1) + U^m(i,j+1) + B(i,j))/4 \quad (in\ 2D\ implementation)$

   Hence, at each step, the value can be computed by making use of the previously obtained neighboring values. After the grid point values are computed in a particular iteration, they only have to be communicated to the neighboring processor before the next iteration. The figures below demonstrate the steps in a parallel Jacobi implementation (striped partitioning)

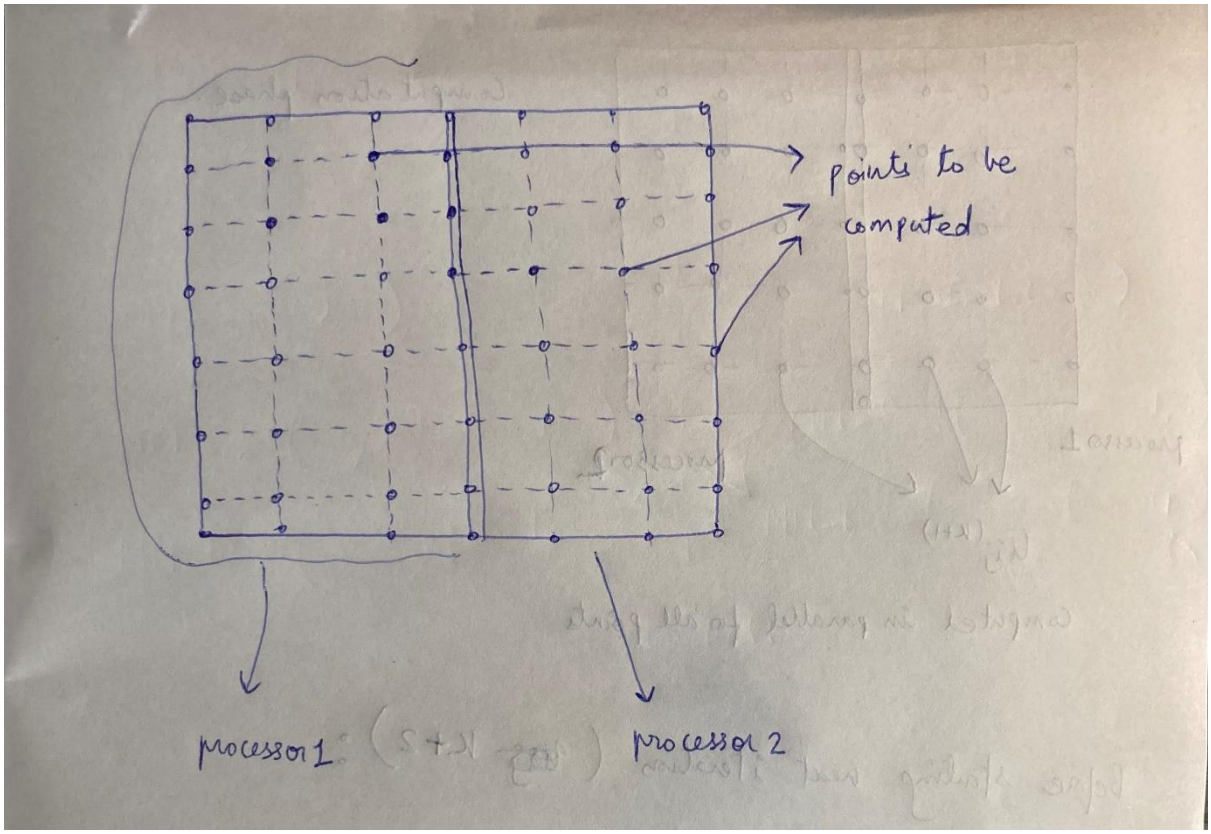


Figure 1 Striped partitioning example in Jacobi

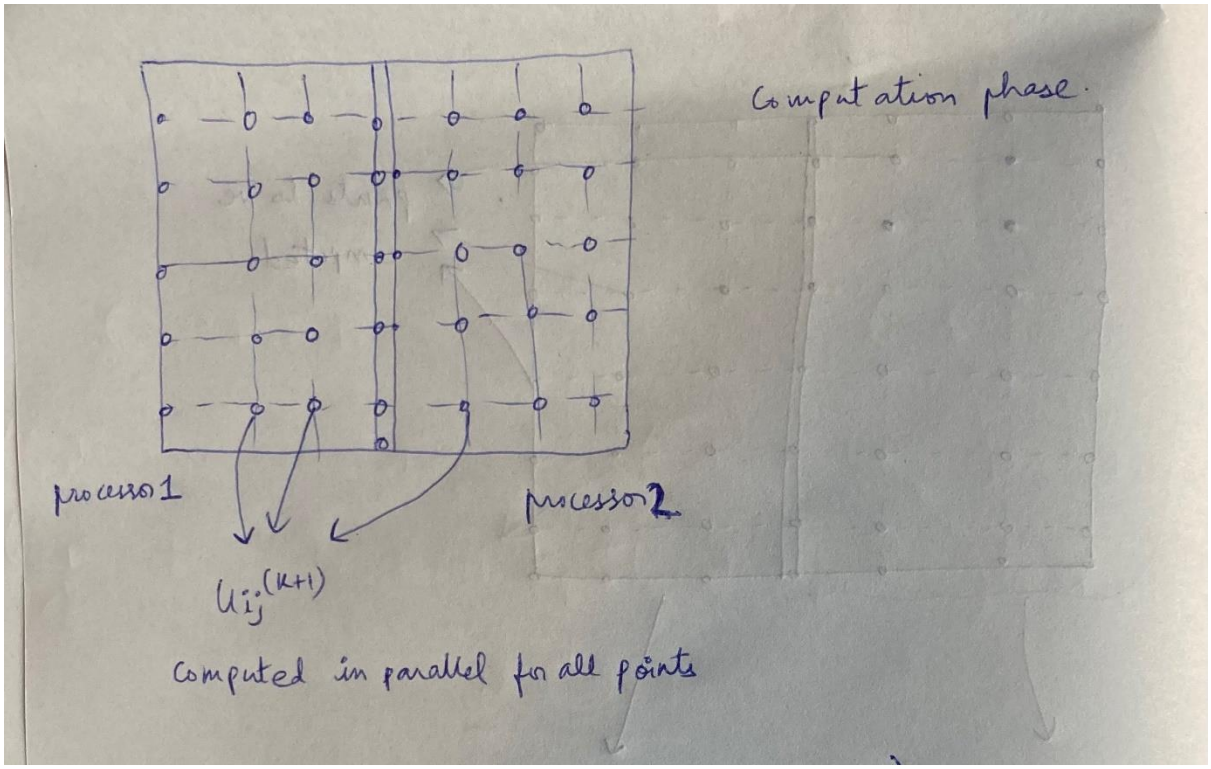
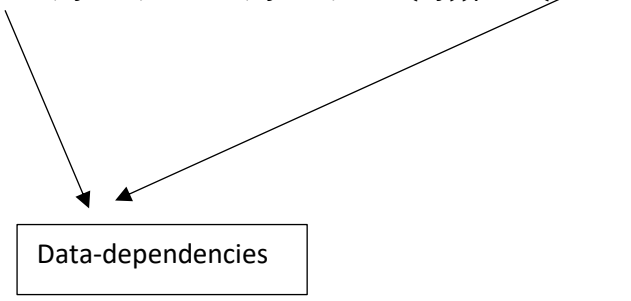
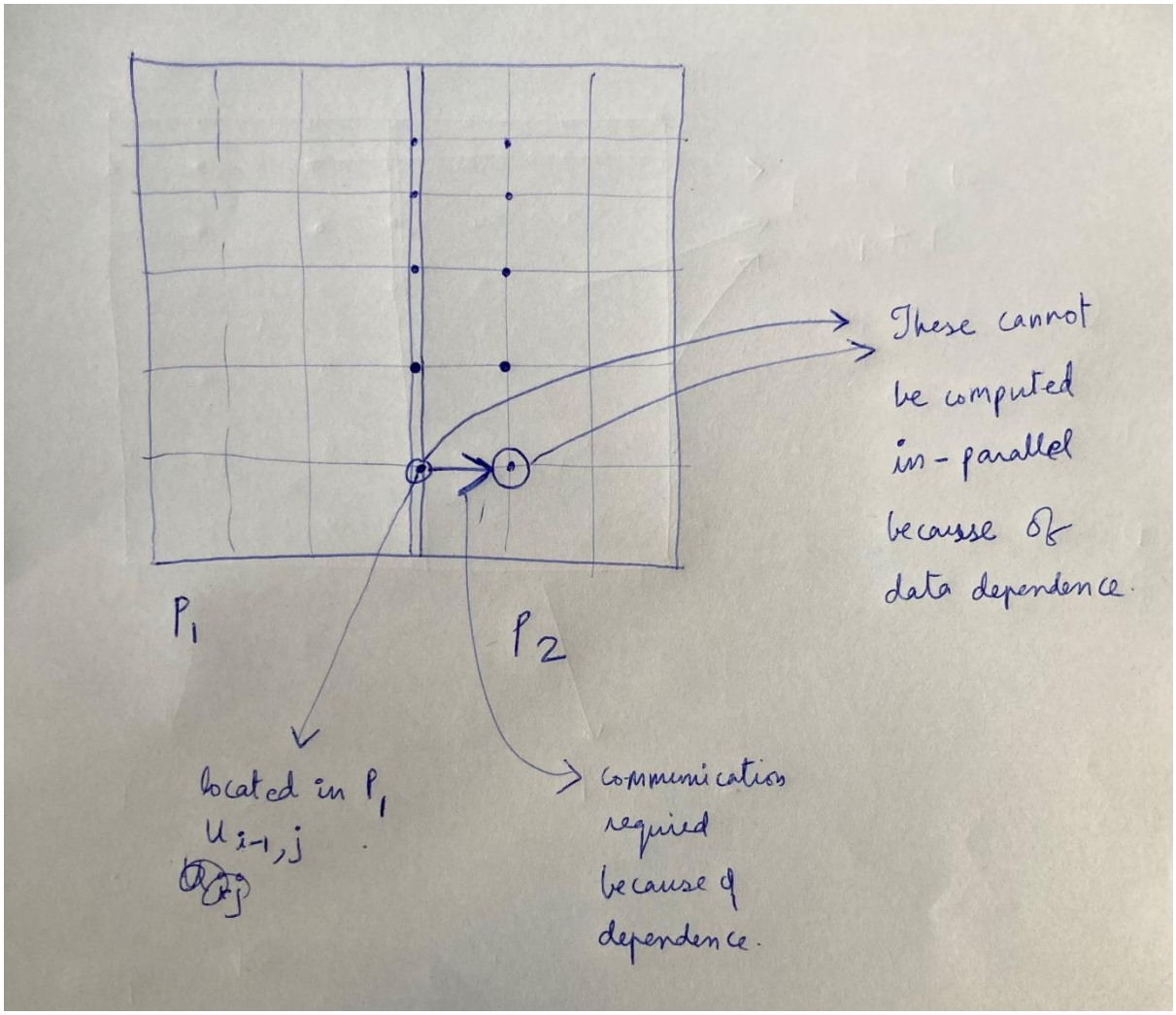


Figure 2 Parallel computation in Jacobi because of absence of data dependencies

In Gauss-Seidel, such an implementation is not possible because of data-dependencies with values computed in the same iteration. i.e., $U^{m+1}(i,j) = (U^{m+1}(i-1,j) + U^{m}(i+1,j) + U^{m+1}(i,j-1) + U^{m}(i,j+1) + B(i,j))/4 \quad (in\ 2D\ implementation)$



So, the computations in processor 1 and 2 cannot happen in-parallel for a striped partitioning as illustrated below.



For parallel implementation of Gauss-Seidel, Red-Black checkered partitioning is used. However, this requires two parallelization steps.

2. The normal iterative methods involve communicating the boundary grid point values to the neighboring processors after every iteration. However, this is inefficient because a large number of messages are sent after every iteration. An improvement can be achieved by decreasing the frequency of communication. The **communication-avoiding Jacobi** method does exactly this: it stores the values of k iterations by computing $A^1x, A^2x, \ldots . A^kx$ beforehand. This improves performance because communication is needed only every k iterations. The following figures explain how this is achieved.

   Consider a 1D Poisson equation. In this, each point X_i depends on X_{i+1} and X_{i-1}

   Representing this as a vector, we get:

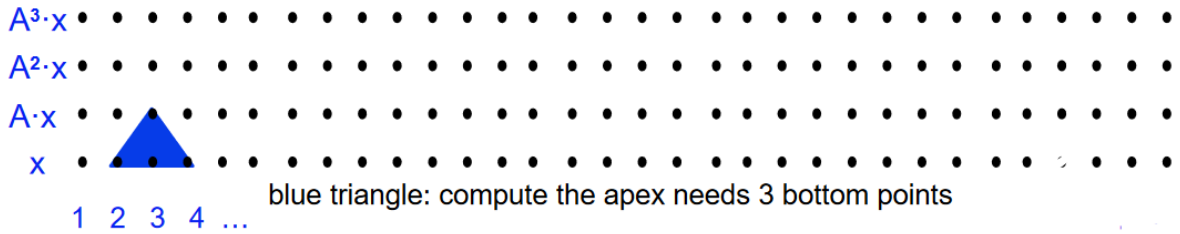
$$X^{m+1} = AX^m + b/2$$

If we decide to store the values of 3 iterations to reduce communication, we get:

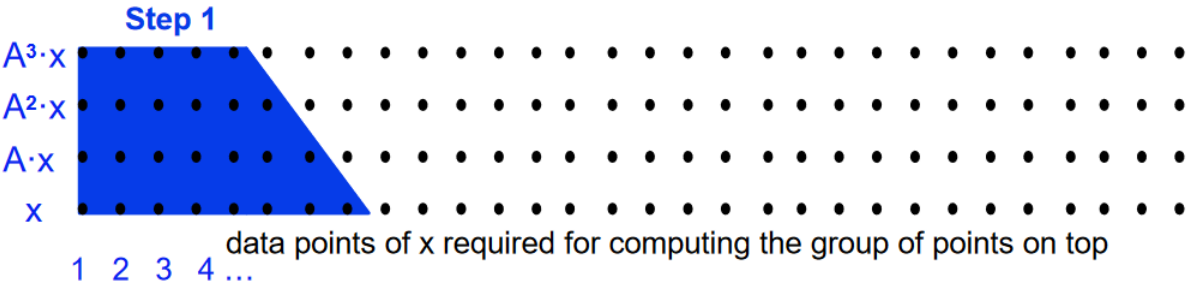
$$X^{m+2} = A^2X^m + some\ term$$
$$X^{m+3} = A^3X^m + some\ term$$

So, we can reduce communication by computing A^1X^m, A^2X^m, A^3X^m

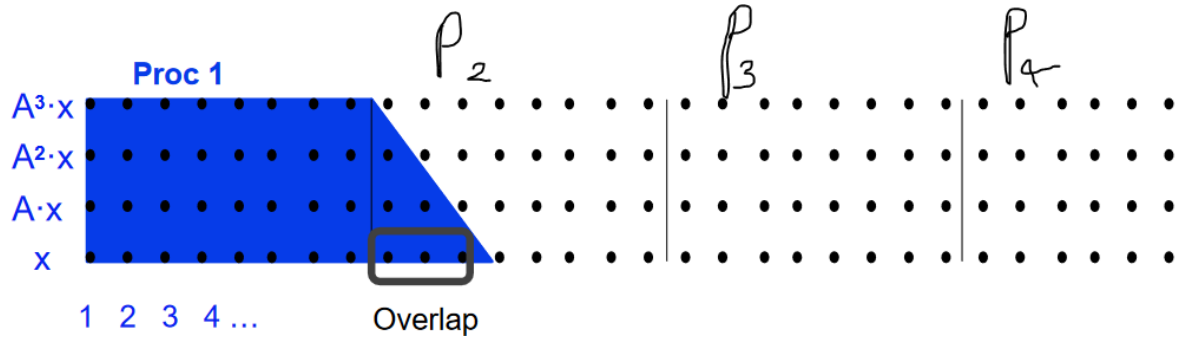
However, we know that each point depends on points before and after it:

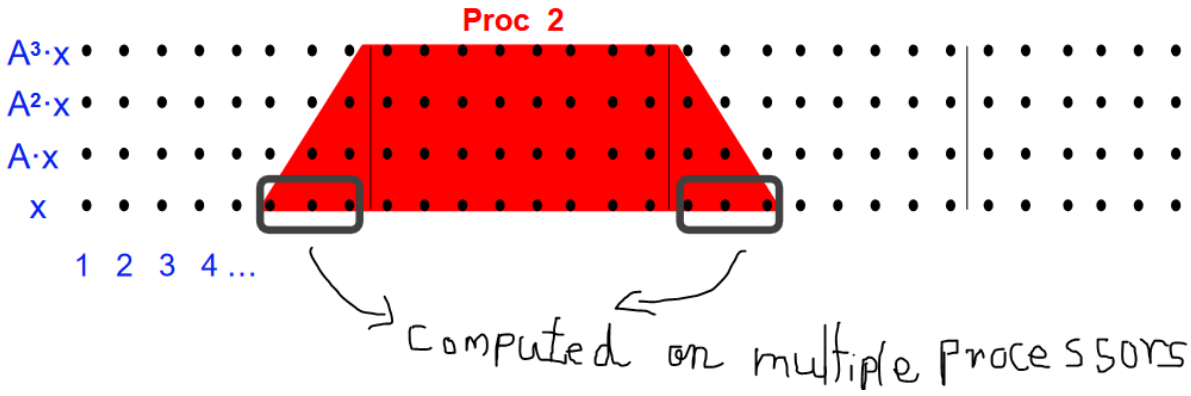


On computing higher powers of matrix A, we get a dependence as shown below:



This results in overlapping/redundant computations when implemented parallelly by partitioning the grid points across devices.





Similar overlaps can be obtained by drawing the diagrams for processors 3 and 4.

We hence arrive at the following trade-off:

> *Number of communications can be reduced but the number of redundant computations increases. (Number of messages sent is divided by k)*