SUMMARY OF THE CODE:

My code has changed a lot from the beginning that I started the coursework. I first started coding in serial and during MPI I did a lot of changes.

My final code is structured into 2 classes and 1 main program. The main program calls these 2 classes and has been used to observe time performance and optimisation of the code.

1. **Model:** Model class includes a header file Model.h and Model.cpp.This class declares, validates and calculates all the variables to successfully implement the Burgers Equation. Moreover, calculates the nodes per processor so that the domain is successfully divided. The constructor of this class calls the following functions.

   - **Program_Options**: Reads input variables from the terminal through the Boost Library (Program_Options). The variables are the following:
     - Parameters in the equation: ax, ay, b and c.
     - Nodes in the X and Y direction (Nx,Ny).
     - Timesteps (Nt).
     - For the parallel case I added Px (processors in x) and Py (processors in y).

     Also a help option has been added so that it returns all the parameters needed to run the code and the requirements to be validated.
   - **ValidateParameters**: Validates all the input variables from "programoptions" function. For example, checks if the subdomains equal to the number of processors.
   - **CalculateParameters**: Calculates nodes per processor and other parameters through the input variables. For example, it calculates CFL from $CFL = dt(2/dx + 2/dy)$. The number 2 corresponds to the maximum value in the initial function (Physical domain of dependence)
   - **Printparameters**: This function basically prints all the calculated and input parametes at the end of the constructor.

2. **BurguersNova:** BurguersNova includes a header file and BurguersNova.cpp. This class sets initial velocity field and computes time integration (for each processor). This class also computes energy of the velocity field and gathers all data from all processors. The functions that need to be called in the main.cpp (set initial conditon, iteration process ,gather and energy) are declared as public. Most of the other functions are

called in the constructor of the class and therefore are all declared as private. Summary of important functions:

- **setmodel()**: Read useful values from the class Model and stores them for the other functions.
- **gridprocessors()**: Computes MPI_Cart_create. Assigns coordinates(x,y) and neighbours. It also orders the processors in a cartesian grid.
- **setvalue() and double initilfunction(posi,posj)**: "Setvalue" allocates the value given by "initilfunction" for each node. The "initilfunction" declares and stores the indices(row,col) of the first node of each subdomain (Really useful for the gathering) and computes the radius and the initial value of each node.
- **iterativeprocess()**: Computes time integration exchanging rows and columns (through MPI_Sendrecv) from the processors neighbours declared in the function "Gridprocessors()".In order to exchange columns a new MPI_Datatype has been created and MPI_Sendrecv has been used.Moreover, since velocity fields (u,v) are the same only 1 field is used to iterate (optimizes the code).
- **gatherprocessors()**: Gatherprocessors gathers the following parameters (to rank 0): **Indices(row,col)** stored in the initilfunction, **coordinates** of each processor and **nodes in x and y** of each processor. After these operations, it computes offset and count arrays for the MPI_Gatherv. This gathering stores all the velocity fields into one 1d array called gathermatrix. Then, with the use of the coordinates and the indices all the values are assigned to the corresponding node in the final velocity field.
- **energy()**: Performs integration in the final matrix and prints the final value in the terminal.