Strategies used in PC Assignment 2

# OpenMP

* I have created helper functions to initialize data and test the results from the program itself. This is common for the rest of the parallel versions too.
* This program uses a 2D array for inputs and 1D arrays for the rest of the containers. Since all the threads use the same memory, there is no need to share variables explicitly which gave me a chance to use a 2D array. In parallel versions such as MPI, this is quite inefficient.
* I have paralyzed the code in 3 places,
  + First penalization is a straightforward parallelization use case. Calculating each theta should be done independently which makes it a good opportunity to use penalization.
  + The second parallel version shared one variable. The sum of a set of numbers in each thread should be added to the shared variable. This makes it a good opportunity to use reduction.
  + The last one is a simple code that can be paralyzed. Even though an array is one variable, its elements are in different memory points. Therefore, assigning independent values to these elements can be done parallelly. There is no serial code involved in this one. Therefore, even if the M is a small number we still can get the advantage of penalization.
* It is worth mentioning that the MAX\_ITERATIONS loop cannot be parallelized because each iteration needs results from the previous iteration.

# MPI

* Sharing a 1D array is more efficient than sharing a 2D array in MPI. Therefore, the input is a 1D array that represents a 2D array.
* Here pointers are used for inputs, outputs, and thetas because they are only initialized in the main process. Those arrays are used in collective functions which are also executed in non-main processors but only the main process consumes their values.
* The scatter function is used to distribute inputs and outputs before starting the loops.
* Theta is shared inside the loop because it changes after every iteration.
* Reduction function is used to sum local summations of every process.

# OneAPI

* This one used a 2D array for inputs because OneAPI provides easy APIs to share 2D arrays between devices.
* Implementation is pretty straightforward,
  + A GPU is used to calculate thetas in every iteration.
  + Each theta is calculated using a parallel for loop because they are independent.
  + A GPU is used for this part because they are good at arithmetic operations.
* Input and output array buffers and theta buffers are in different scopes. The reasons for them are,
  + Inputs and outputs are the same for all the iterations in the main loop.
  + Thetas change for each iteration in the main loop.
  + When an iteration is over, the thetas are copied from the device to the host. Then host shares those thetas again in the next iteration.

# Cuda

* Cuda does not provide any easy APIs to share 2D arrays, therefore a 1D array is used to represent the 2D array inputs.
* Same as in OneAPI, GPUs are used to calculate thetas and each theta is calculated independently in a parallel for loop.