DSAI 3202 – Parallel and distributed computing

Lab – 6: Distributed computing using mpi4py

# Objectives:

* Develop Python programs that take advantage of multiple machines to run.

# Tools and Concepts:

* Python: Programming language.
* Packages: multiprocessing, mpi4py, pandas.

# Exercises in conjunction with the lecture

## Square program

* Create e function square that computes the square number of a series of integers from 1 to n.
* The program must be parallel and uses mpi4py.
* Save the provided Python script as calculate\_squares.py.

## Environment Setup:

* Ensure mpi4py is installed on all targeted machines.
* Copy the ssh-id from the main machine to the others.
* Use SSH to access each machine.

## Distribute the Program:

* Copy calculate\_squares.py to each machine in the same directory path or use a shared file system.
* Create a Host File: On your main machine, create machines.txt listing the IP addresses of all

## Run the Program:

* Execute the program across all machines with

## **Observe Results**:

* The root process (rank 0) gathers all partial results, prints the size of final array of squares and the last square as well as the time taken.

## Modify you program to compute the square up to 1e8

## Bonus (5% in the second assignment):

* You are given 300 to run your program. What is the highest square you can reach?
* Bonus is only given to the group with the highest square with all other results not lost.

# Building an MPI Program for Virus Spread Simulation

## Step 1: Initialize the MPI Environment

* Import the necessary libraries.
* Initialize the MPI communicator.
* Get the rank and size of the current process.

## Step 2: Define Parameters

* Define the population size, chance of virus spread, and generate a random vaccination rate for each process.
* Example: population\_size = 100, spread\_chance = 0.3, vaccination\_rate = np.random.uniform(0.1, 0.5).

## Step 3: Initialize the Population

* Initialize the population array with zeros representing uninfected individuals.
* Infect a small percentage of individuals at the start.
* Example:

1. if rank == 0:

2. infected\_indices = np.random.choice(population\_size,

int(0.1 \* population\_size),

replace=False)

3. population[infected\_indices] = 1

## Step 4: Implement Virus Spread Function

* Create a function to simulate virus spread based on spread chance and vaccination rate.
* Update the population array based on infection rules.
* Example:

def spread\_virus(population):

new\_population = population.copy()

# Implement virus spread logic here

return new\_population

## Step 5: Simulate Virus Spread

* Iterate over multiple time steps to simulate virus spread through the population.
* Exchange data between processes using MPI communication.
* Example:

for \_ in range(10):

population = spread\_virus(population)

if rank != 0:

comm.send(population, dest=0)

else:

for i in range(1, size):

received\_data = comm.recv(source=i)

population += received\_data

## Step 6: Calculate Infection Rate

* Calculate the infection rate for each process based on the final infected population count.
* Print the infection rate for each process.
* Example:

total\_infected = np.sum(population)

infection\_rate = total\_infected / population\_size

print(f"Process {rank} Infection Rate: {infection\_rate}")

## Step 7: Run and Experiment

* Run the program on multiple processors to observe different infection rates based on random vaccination rates assigned to each process.
* Experiment with changing parameters like spread chance and vaccination rates to observe their impact on virus spread dynamics.