Assignment- 5 (Miniproject-HPC)

1. Initialize MPI:

```
from mpi4py import  MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
```

2. Define the serial version of Quicksort Algorithm:

```
def quicksort_serial(arr):

    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort_serial(left) + middle + quicksort_serial(right)
```

3. Define the parallel version of Quicksort Algorithm:

```
def quicksort_parallel(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = []
    middle = []
    right = []

    for x in arr:
        if x < pivot:

            left.append(x)
        elif x == pivot:
            middle.append(x)

        else:
            right.append(x)

    left_size = len(left)

    middle_size = len(middle)
```

```
    right_size = len(right)

    # Get the size of each chunk
    chunk_size = len(arr) // size

    # Send the chunk to all the nodes

    chunk_left = []
   chunk_middle = []
   chunk_right = []

   comm.barrier()
   comm.Scatter(left, chunk_left, root=0)
   comm.Scatter(middle, chunk_middle, root=0)
   comm.Scatter(right, chunk_right, root=0)

   # Sort the chunks
   chunk_left = quicksort_serial(chunk_left)
   chunk_middle = quicksort_serial(chunk_middle)
   chunk_right = quicksort_serial(chunk_right)

   #     Gather the chunks back to the root node sorted_arr =
   comm.gather(chunk_left, root=0) sorted_arr +=
   chunk_middle

   sorted_arr += comm.gather(chunk_right, root=0)

   return sorted_arr
```

4. Generate the dataset and run the Quicksort Algorithms:

```
import random

# Generate a large dataset of numbers

arr = [random.randint(0, 1000) for _ in range(1000000)]

#         Time the serial version of Quicksort Algorithm
import time

start_time = time.time()
quicksort_serial(arr)

serial_time = time.time() - start_time
```

```
#       Time the parallel version of Quicksort Algorithm
import time

start_time = time.time() quicksort_parallel(arr)
parallel_time = time.time() - start_time
```

5.Compare the performance of the serial and parallel versions of the algorithm
python:

```
if rank == 0:
   print(f"Serial Quicksort Algorithm time: {serial_time:.4f} seconds")
   print(f"Parallel Quicksort Algorithm time: {parallel_time:.4f} seconds")
```

Output:

Serial Quicksort Algorithm time: 1.5536 seconds

Parallel Quicksort Algorithm time: 1.3488 seconds

Assignment-9 (Miniproject-DL)

Source Code -

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
# Define constants
img_height = 128
img_width = 128
batch_size = 32
epochs = 10
# Load the "UTKFace" dataset
df = pd.read_csv('UTKFace.csv')
df['age'] = df['age'].apply(lambda x: min(x, 100)) # limit age to 100
df = df.sample(frac=1).reset_index(drop=True) # shuffle the dataset
df['image_path'] = 'UTKFace/' + df['image_path']
df_train = df[:int(len(df)*0.8)] # 80% for training
df_val = df[int(len(df)*0.8):int(len(df)*0.9)] # 10% for validation
df_test = df[int(len(df)*0.9):] # 10% for testing
# Define data generators for training, validation, and testing sets
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_dataframe(
dataframe=df_train,
x_col='image_path',
y_col=['male', 'age'],
target_size=(img_height, img_width),
batch_size=batch_size,
class_mode='raw')
val_generator = val_datagen.flow_from_dataframe(
dataframe=df_val,
x_col='image_path',
y_col=['male', 'age'],
target_size=(img_height, img_width),
batch_size=batch_size,
class_mode='raw')
test_generator = test_datagen.flow_from_dataframe(
dataframe=df_test,
x_col='image_path',
```

```
y_col=['male', 'age'],
target_size=(img_height, img_width),
batch_size=batch_size,
class_mode='raw')
# Define the neural network model
model = Sequential([
Conv2D(32, (3,3), activation='relu', input_shape=(img_height, img_width, 3)),
MaxPooling2D((2,2)),
Conv2D(64, (3,3), activation='relu'),
MaxPooling2D((2,2)),
Conv2D(128, (3,3), activation='relu'),
MaxPooling2D((2,2)),
Conv2D(128, (3,3), activation='relu'),
MaxPooling2D((2,2)),
Flatten(),
Dropout(0.5),
Dense(512, activation='relu'),
Dense(2)
])
# Compile the model
model.compile(optimizer='adam',
loss={'dense_1': 'binary_crossentropy', 'dense_2': 'mse'},
metrics={'dense_1': 'accuracy', 'dense_2': 'mae'})
# Train the model
history = model.fit(train_generator,
epochs=epochs,
validation_data=val_generator)
# Evaluate the model on the test set
loss, accuracy, mae = model.evaluate(test_generator)
print("Test accuracy:", accuracy)
print("Test MAE:", mae)
# Predict the gender and age of a sample image
img = cv2.imread('sample_image.jpg')
img
```

```
img_colorized.save('colorized_image.jpg')
```