’’Name:Rutuja Ashok Jagtap
std:BE(comp)-A

Title: Write a program to implement Parallel Bubble Sort and Merge sort using OpenMP. Use existing algorithms and measure the performance of sequential and parallel algorithms.”

```
import multiprocessing

import random

import time



# Sequential Merge Sort (returns sorted array instead of modifying in-place)

def merge_sort_return(arr):

  if len(arr) <= 1:

    return arr


  mid = len(arr) // 2

  left_half = merge_sort_return(arr[:mid])

  right_half = merge_sort_return(arr[mid:])


  return merge(left_half, right_half)



# Merging function

def merge(left_half, right_half):

  sorted_arr = []

  i = j = 0


  while i < len(left_half) and j < len(right_half):

    if left_half[i] < right_half[j]:

      sorted_arr.append(left_half[i])

      i += 1

    else:
```

```
            sorted_arr.append(right_half[j])
            j += 1

    sorted_arr.extend(left_half[i:])
    sorted_arr.extend(right_half[j:])
    return sorted_arr


# Parallel Merge Sort Helper
def parallel_merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    # Create a multiprocessing pool
    with multiprocessing.Pool(processes=2) as pool:
        left_sorted, right_sorted = pool.map(merge_sort_return, [left_half, right_half])

    return merge(left_sorted, right_sorted)


# Performance Comparison Function
def measure_performance():
    size = 5000  # Change size for larger inputs
    arr1 = [random.randint(1, 10000) for _ in range(size)]
    arr2 = arr1[:]

    print("Sorting an array of size:", size)
```

```
    # Measure Sequential Merge Sort
    start_time = time.time()
    sorted_arr1 = merge_sort_return(arr1)
    print(f"Sequential Merge Sort Time: {time.time() - start_time:.5f} sec")

    # Measure Parallel Merge Sort
    start_time = time.time()
    sorted_arr2 = parallel_merge_sort(arr2)  # Needs reassignment
    print(f"Parallel Merge Sort Time: {time.time() - start_time:.5f} sec")


if __name__ == "__main__":
    measure_performance()
```

Output:

```
Python 3.10.11 (tags/v3.10.11:7d4cc5a, Apr  5 2023, 00:38:17) [MSC v.1
 (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more informatio

========== RESTART: C:/Users/Rutuja/Desktop/hpc/practical2.py ==========
Sorting an array of size: 5000
Sequential Merge Sort Time: 0.01586 sec
Parallel Merge Sort Time: 0.09920 sec
```