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

Title: Implement Min, Max, Sum and Average operations using Parallel Reduction.”

```
import multiprocessing

import random

import time



# Function to find min in a chunk

def parallel_min(data):

    return min(data)



# Function to find max in a chunk

def parallel_max(data):

    return max(data)



# Function to find sum in a chunk

def parallel_sum(data):

    return sum(data)



# Function to find sum & count for average calculation

def parallel_sum_count(data):

    return sum(data), len(data)



# Main function to perform parallel reduction

def parallel_reduce(operation, data, num_workers=None):

    if num_workers is None:
```

```
    num_workers = multiprocessing.cpu_count()  # Use all available CPU cores

  chunk_size = len(data) // num_workers
  chunks = [data[i * chunk_size:(i + 1) * chunk_size] for i in range(num_workers)]

  # Handle leftover elements
  if len(data) % num_workers != 0:
    chunks.append(data[num_workers * chunk_size:])

  with multiprocessing.Pool(processes=num_workers) as pool:
    results = pool.map(operation, chunks)

  # Perform final reduction on the main process
  if operation == parallel_min:
    return min(results)
  elif operation == parallel_max:
    return max(results)
  elif operation == parallel_sum:
    return sum(results)
  elif operation == parallel_sum_count:
    total_sum, total_count = map(sum, zip(*results))
    return total_sum / total_count  # Compute average


# Performance comparison function
def measure_performance():
  size = 5000000  # Large dataset for effective parallelism
  data = [random.randint(1, 10000) for _ in range(size)]

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

```
# Sequential Operations
start_time = time.time()
min_val = min(data)
print(f"Sequential Min: {min_val}, Time: {time.time() - start_time:.5f} sec")

start_time = time.time()
max_val = max(data)
print(f"Sequential Max: {max_val}, Time: {time.time() - start_time:.5f} sec")

start_time = time.time()
sum_val = sum(data)
print(f"Sequential Sum: {sum_val}, Time: {time.time() - start_time:.5f} sec")

start_time = time.time()
avg_val = sum_val / len(data)
print(f"Sequential Average: {avg_val:.2f}, Time: {time.time() - start_time:.5f} sec")

# Parallel Operations
start_time = time.time()
min_val = parallel_reduce(parallel_min, data)
print(f"Parallel Min: {min_val}, Time: {time.time() - start_time:.5f} sec")

start_time = time.time()
max_val = parallel_reduce(parallel_max, data)
print(f"Parallel Max: {max_val}, Time: {time.time() - start_time:.5f} sec")

start_time = time.time()
sum_val = parallel_reduce(parallel_sum, data)
print(f"Parallel Sum: {sum_val}, Time: {time.time() - start_time:.5f} sec")

start_time = time.time()
```

```
    avg_val = parallel_reduce(parallel_sum_count, data)
    print(f"Parallel Average: {avg_val:.2f}, 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 informatic

============== RESTART: C:/Users/Rutuja/Desktop/hpc/practical3.py ====
Processing an array of size: 5000000
Sequential Min: 1, Time: 0.04834 sec
Sequential Max: 10000, Time: 0.06390 sec
Sequential Sum: 24990513581, Time: 0.08704 sec
Sequential Average: 4998.10, Time: 0.00000 sec
Parallel Min: 1, Time: 0.63045 sec
Parallel Max: 10000, Time: 0.53975 sec
Parallel Sum: 24990513581, Time: 0.52579 sec
Parallel Average: 4998.10, Time: 0.51762 sec
```