**Exercise – 12**

**Parallel and Randomized Algorithm**

**Aim:** To write Python code to implement Parallel and Randomized algorithms.

**Algorithm:**

1. Define the problem and identify the parts that can be executed in parallel. Look for independent subproblems or computations that can be split among multiple threads or processes.

2. Design a randomized strategy or algorithm to solve the problem. Randomization can be used to introduce variability, improve efficiency, or avoid certain pitfalls. This could involve random sampling, shuffling, or random choices within the algorithm.

3. Determine the level of parallelism required. Decide whether you want to parallelize at a coarse-grained level (e.g., parallelizing independent iterations of a loop) or at a fine-grained level (e.g., parallelizing operations within a loop iteration).

4. Choose a parallel programming model or framework suitable for your needs. This could be multi-threading, multi-processing, or distributed computing frameworks, depending on the scale and architecture of your system.

5. Implement the parallel and randomized algorithm using the chosen programming model or framework. This involves structuring your code to distribute the workload across multiple threads, processes, or machines.

6. Ensure proper synchronization and communication between parallel threads or processes. Use appropriate synchronization primitives, such as locks, semaphores, or barriers, to coordinate access to shared resources and avoid race conditions.

7. Test and validate your implementation to ensure correctness and measure performance. Verify that the parallel and randomized algorithm produces the expected results and assess its efficiency compared to a sequential version or other existing algorithms.

8. Fine-tune and optimize the algorithm and its parallelization if necessary. Analyze the performance bottlenecks and identify areas for improvement. This could involve optimizing data access patterns, load balancing, or tuning the randomization strategy.

9. Repeat steps 7 and 8 as needed to refine your parallel and randomized algorithm.

10. Document your algorithm, including any assumptions, limitations, and trade-offs. Provide clear instructions on how to use and reproduce the results.

**Source Code:**

import multiprocessing

import random

def calculate\_sum\_of\_squares(numbers):

    total = 0

    for num in numbers:

        total += num \* num

    return total

if \_\_name\_\_ == "\_\_main\_\_":

    num\_processes = multiprocessing.cpu\_count()

# Get input from the user

    num\_elements = int(input("Enter the number of elements: "))

    numbers = []

    for \_ in range(num\_elements):

        numbers.append(random.randint(1, 10))

# Split the list into chunks for parallel processing

    chunk\_size = num\_elements // num\_processes

    chunks = [numbers[i:i+chunk\_size] for i in range(0, num\_elements, chunk\_size)]

# Create a pool of worker processes

    pool = multiprocessing.Pool(processes=num\_processes)

# Perform parallel computations

    results = pool.map(calculate\_sum\_of\_squares, chunks)

# Combine the results

    total\_sum = sum(results)

    print("Total sum of squares:", total\_sum)

**Sample Input and Output:**

Input: 23

OUTPUT 1:

A black text on a white background

Description automatically generated with low confidence

OUTPUT 2:

A picture containing text, font, screenshot, white

Description automatically generated

OUTPUT 3:

A picture containing text, font, white

Description automatically generated

**Result:**

Thus, the Parallel and Randomized algorithms have been successfully implemented using Python code and the output is verified.

**Result:**

Thus, the Parallel and Randomized algorithms have been successfully implemented using Python code and the output is verified.