**Parallel and Randomized Algorithm**

**Aim:**

To write the Python code to implement Parallel and Randomized algorithms.

**Problem Description:**

The problem is to efficiently calculate the total sum of squares for a large list of numbers using parallel and randomized computation. The algorithm generates a list of numbers, shuffles them randomly, and then divides the list into chunks. Each chunk is processed independently by a separate worker process, calculating the sum of squares. The results from each process are combined to obtain the final total sum of squares. The use of parallel processing and randomization improves the efficiency and introduces variability in the computation.

**Algorithm:**

* Generate a list of numbers.
* Shuffle the list randomly.
* Split the list into chunks based on the number of CPU cores available.
* Assign each chunk to a separate worker process.
* Each worker process calculates the sum of squares for its assigned chunk.
* Combine the results from each worker process by summing them up.
* Output the total sum of squares.

**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()

    num\_elements = int(input("\nEnter no of elements: "))

    numbers = [random.randint(1, 100) for \_ in range(num\_elements)]

    random.shuffle(numbers)

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

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

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

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

    total\_sum = sum(results)

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

**Output:**

