

4. Write a OpenMP program to find the prime numbers from 1 to n employing parallel for directive. Record both serial and parallel execution times.

```
#include <stdio.h> // Standard input/output functions
#include <omp.h>    // OpenMP library for parallelism and timing

// Function to check if a number is prime
int is_prime(int n) {
    if (n < 2) return 0;           // Numbers less than 2 are not prime
    for (int i = 2; i * i <= n; i++) // Loop only up to sqrt(n)
        if (n % i == 0) return 0; // If divisible, it's not prime
    return 1;                     // If no divisor found, it's prime
}

int main() {
    int n = 100000; // Upper limit for checking primes
    double start, end; // Variables for timing

    // ----- Serial Execution -----
    start = omp_get_wtime(); // Record start time for serial run
    for (int i = 1; i <= n; i++) // Check all numbers from 1 to n
        is_prime(i);           // Call primality function (no printing)
    end = omp_get_wtime();      // Record end time
    printf("Serial Time: %f\n", end - start); // Print elapsed time

    // ----- Parallel Execution -----
    start = omp_get_wtime(); // Record start time for parallel run
    #pragma omp parallel for // Divide loop iterations among threads
    for (int i = 1; i <= n; i++) // Each thread checks part of the range
        is_prime(i);           // Threads work independently
    end = omp_get_wtime();      // Record end time
    printf("Parallel Time: %f\n", end - start); // Print elapsed time

    return 0; // Successful program termination
}
```

Compile : gcc -fopenmp prime_parallel.c -o prime_parallel

Run : ./prime_parallel

Serial Time: 0.018138

Parallel Time: 0.012131