**Original program reference:**

<https://www.geeksforgeeks.org/factorial-of-an-array-of-integers/>

**Code Structure and Functionality**

1. **Header Inclusions:**
   * <iostream>: For input-output operations.
   * <cstdlib> and <ctime>: For random number generation and seeding.
   * <omp.h>: OpenMP library for parallel programming.
2. **Constants Definition:**
   * N: Number of computations.
   * MAX\_NUM: Maximum number for factorial calculation.
   * REPEAT: Number of repetitions for each factorial calculation.
3. **Factorial Function:**
   * A function factorial(int n) calculates the factorial of a given number n.
4. **Main Function:**
   * Random Number Generation: Generates N random numbers between 1 and MAX\_NUM, storing them in the array nums.
   * **Serial Computation:**
     + Measures the time taken to compute the factorial of each number in nums, repeated REPEAT times, in a serial manner.
   * **Parallel Computation with OpenMP:**
     + Repeats the factorial computation but in a parallel fashion using OpenMP.
     + #pragma omp parallel for schedule(dynamic) num\_threads(6): This directive enables parallel execution with dynamic scheduling and specifies the use of 6 threads.
   * **Time Calculation and Output:**
     + Calculates and displays the time taken for both serial and parallel execution.
     + Compares the execution times to determine the speedup achieved by parallelization.

**OpenMP Usage for Acceleration**

* Parallel Loop: The #pragma omp parallel for directive is used to parallelize the outer loop of the factorial calculations.
* Dynamic Scheduling: schedule(dynamic) allows OpenMP to dynamically assign iterations of the loop to different threads, which can be beneficial for load balancing when iterations take varying amounts of time.
* Number of Threads: Specifying num\_threads(6) suggests the use of 6 threads for parallel processing.

**Inclusion of Both Optimized and Original Code**

* The program includes both the serial (non-optimized) and parallel (optimized) versions of the factorial computation. This dual implementation is crucial for comparing the execution times to quantify the performance benefits of parallelization.
* Random Number Generation: The same set of random numbers is used for both serial and parallel computations to ensure a fair comparison.
* Execution Time Measurement: The program measures the time taken by both methods using omp\_get\_wtime(), allowing for a direct comparison of performance.

By combining both the optimized and original code, my program effectively demonstrates the actual execution time difference and performance improvement achieved through parallelization with OpenMP.

**Compilation steps and flags**

g++ -o openmps -fopenmp openmp.cpp

openmps

**Proof of Achieved Speedup**

A screen shot of a computer

Description automatically generated

g++ -o Original Original.cpp

Original

