Name: Shreeyash Shripad Dongarkar

PRN: 22510025

High Performance Computing Lab (B – 1)

Assignment – 1

Title: Introduction to OpenMP

[GitHub Repository](https://github.com/TerminatorShri/22510025_HPCL)

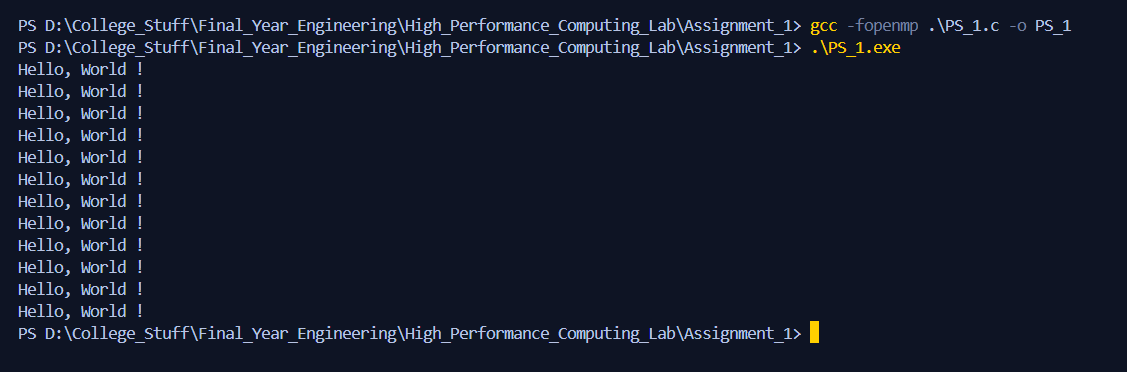
**What is OpenMP?**

OpenMP (Open Multi-Processing) is an API that enables shared-memory multiprocessing programming in C, C++, and Fortran across multiple operating systems. It provides a portable and scalable approach to parallel programming, offering developers a simple yet powerful interface for building high-performance applications. Whether you're targeting a standard desktop or a large-scale supercomputer, OpenMP makes it easier to harness the power of multiple processor cores efficiently.

Problem Statement – 1

Demonstrate Installation and Running of OpenMP Code in C

*(The code for the output shown above is available in the GitHub repository, along with detailed instructions for execution provided in the README.md file.)*



Problem Statement – 2

Print `Hello, World! ` in sequential and parallel in OpenMP

Code:

1. #include <stdio.h>

2. #include <omp.h>

3.

4. int main() {

5.     int num\_threads;

6.

7.     printf("Enter number of threads: ");

8.     scanf("%d", &num\_threads);

9.

10.     omp\_set\_num\_threads(num\_threads);

11.

12.     printf("\n--- Sequential Execution ---\n");

13.     for (int i = 0; i < num\_threads; i++) {

14.         printf("Hello, World from thread %d out of %d (Sequential)\n", i, num\_threads);

15.     }

16.

17.     printf("\n--- Parallel Execution ---\n");

18.

19.     #pragma omp parallel

20.     {

21.         int tid = omp\_get\_thread\_num();

22.         int total = omp\_get\_num\_threads();

23.         printf("Hello, World from thread %d out of %d (Parallel)\n", tid, total);

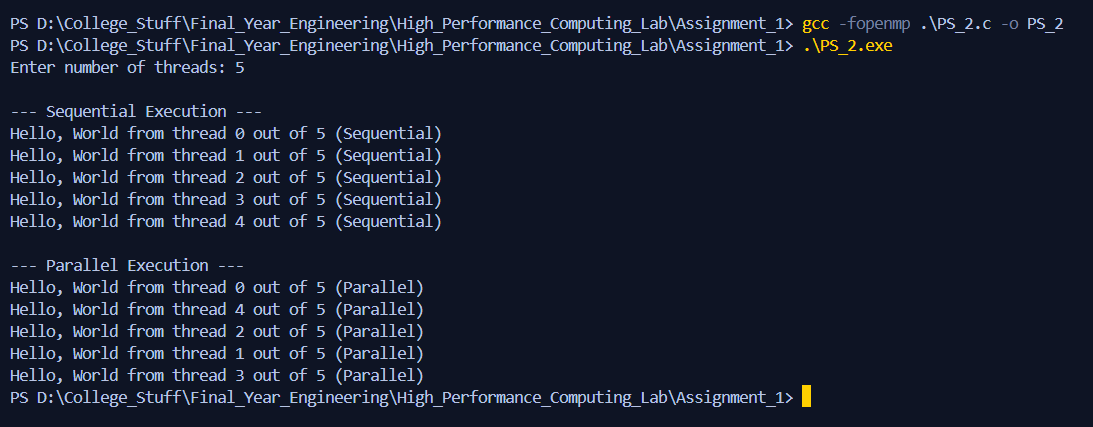
24.     }

25.

26.     return 0;

27. }

Output:



Analysis:

* In **sequential execution**, messages are printed one by one in order from thread 0 to num\_threads - 1.
* In **parallel execution**, multiple threads run concurrently. Each thread prints its own message using OpenMP's omp\_get\_thread\_num() and omp\_get\_num\_threads().
* This clearly shows how parallelism allows multiple parts of the program to run at the same time, which can improve performance for compute-intensive tasks.
* omp\_set\_num\_threads(num\_threads) demonstrates **runtime control** of thread count, a key feature of OpenMP’s flexibility.

Problem Statement – 3

Calculate Theoretical FLOPS of your system. Elaborate Parameters and Show Calculation.

|  |  |  |
| --- | --- | --- |
| Parameter | Symbol | Significance |
| Clock Speed |  | The number of CPU cycles per second (in Hz). Higher clock speed means more operations per second. |
| Number of Cores |  | The number of **physical CPU cores**. More cores enable more operations to be executed in parallel. |
| SIMD Width / Factor |  | Number of floating-point values processed simultaneously by one vector instruction (based on SIMD register width). |
| FMA Factor |  | Accounts for **Fused Multiply-Add** instructions. If supported, each instruction counts as **2 FLOPs** (1 multiply + 1 add). |
| Superscalar Factor |  | Indicates how many SIMD instructions a core can issue per clock cycle (instruction throughput). Modern CPUs may issue 1-2 per cycle. |