**Course:** High Performance Computing Lab

**Practical No 1**

PRN: 22510057

Name: Ashutosh Gundu Birje

Batch: B8

**Title of practical:**

Introduction to OpenMP

**Problem Statement 1**

Demonstrate Installation and Running of OpenMP code in C

Recommended Linux based System:

Following steps are for windows:

OpenMP – Open Multi-Processing is an API that supports multi-platform shared-memory multiprocessing programming in C, C++ and Fortran on multiple OS. OpenMP uses a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from the standard desktop computer to the supercomputer.

To set up OpenMP,

We need to first install C, C++ compiler if not already done. This is possible through the MinGW Installer.  
Reference: Article on GCC and G++ installer ([Link](https://www.scaler.com/topics/c/c-compiler-for-windows/))

Note: Also install `mingw32-pthreads-w32` package.

Then, to run a program in OpenMP, we have to pass a flag `-fopenmp`.

Example:

To run a basic Hello World,

*#include* <stdio.h>

*#include* <omp.h>

*int* main(*void*)

{

*#pragma* *omp* *parallel*

    printf("Hello, world.\n");

*return* 0;

}

gcc -fopenmp test.c -o hello

.\hello.exe



**Problem Statement 2**

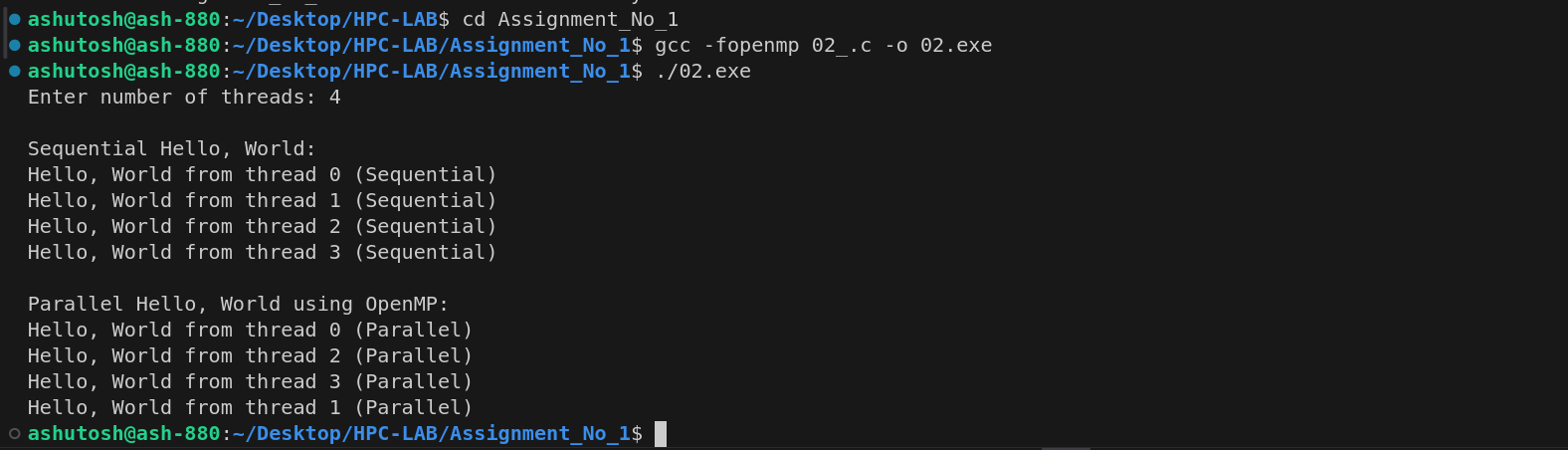
Print ‘Hello, World’ in Sequential and Parallel in OpenMP

We first ask the user for number of threads – OpenMP allows to set the threads at runtime. Then, we print the Hello, World in sequential – number of times of threads count and then run the code in parallel in each thread.

Code snapshot:



Output snapshot:



Analysis:

Sequential Section:

Executes on main thread (single thread).

Iterates from 0 to num\_threads-1, printing messages sequentially.

Parallel Section:

OpenMP creates num\_threads threads.

Each thread independently prints its message.

Thread execution is concurrent — may lead to out-of-order output.

Key Points:

Use of omp\_set\_num\_threads() to set runtime thread count.

#pragma omp parallel to parallelize a block.

omp\_get\_thread\_num() returns unique thread ID.

**Problem statement 3**

Calculate theoretical FLOPS of your system on which you are running the above codes.

FLOPS (Floating Point Operations Per Second)

It is a measure of a computer's performance, especially in scientific computations.

**Formula**

FLOPS = Number of cores × Clock speed × FLOPs per cycle

**Calculation**

FLOPS = 8 cores × 2.3 × 10⁹ cycles/sec × 16 FLOPs/cycle

= 294.4 × 10⁹ FLOPS

= 294.4 GFLOPS (GigaFLOPS)

**Parameter**

|  |  |
| --- | --- |
| Parameter | Example (replace with actual system values) |
| CPU Name | Intel Core i7-12700H |
| Base Clock Speed | 2.3 GHz = 2.3 × 10⁹ cycles/sec |
| Cores | 8 Performance Cores + 4 Efficiency Cores = 12 total |
| FLOPs per cycle | 16 (assuming AVX-512 or FMA with 512-bit vector width) |

**GitHub Link:** [**https://github.com/Ashutoshbirje/HPCLAB/tree/master/Assignment\_No\_1**](https://github.com/Ashutoshbirje/HPCLAB/tree/master/Assignment_No_1)