Course: High Performance Computing Lab

Practical No 1

Github Link : [Sem-7-Assign/HPC lab at main · parshwa913/Sem-7-Assign · GitHub](https://github.com/parshwa913/Sem-7-Assign/tree/main/HPC%20lab)

PRN: 22510064

Name: Parshwa Herwade

Batch: B1

Title: 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");

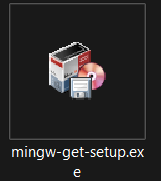
*return* 0;

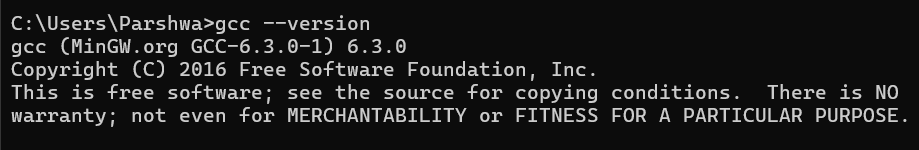
}

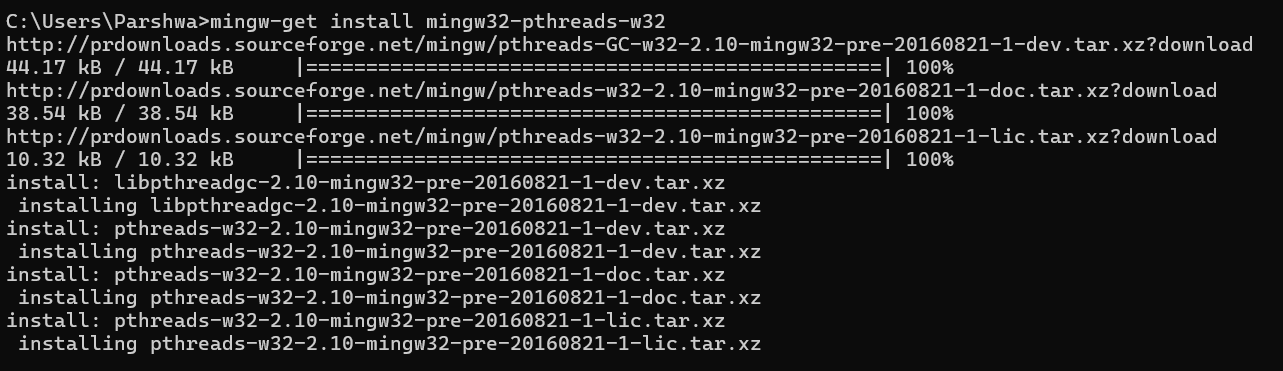
gcc -fopenmp test.c -o hello

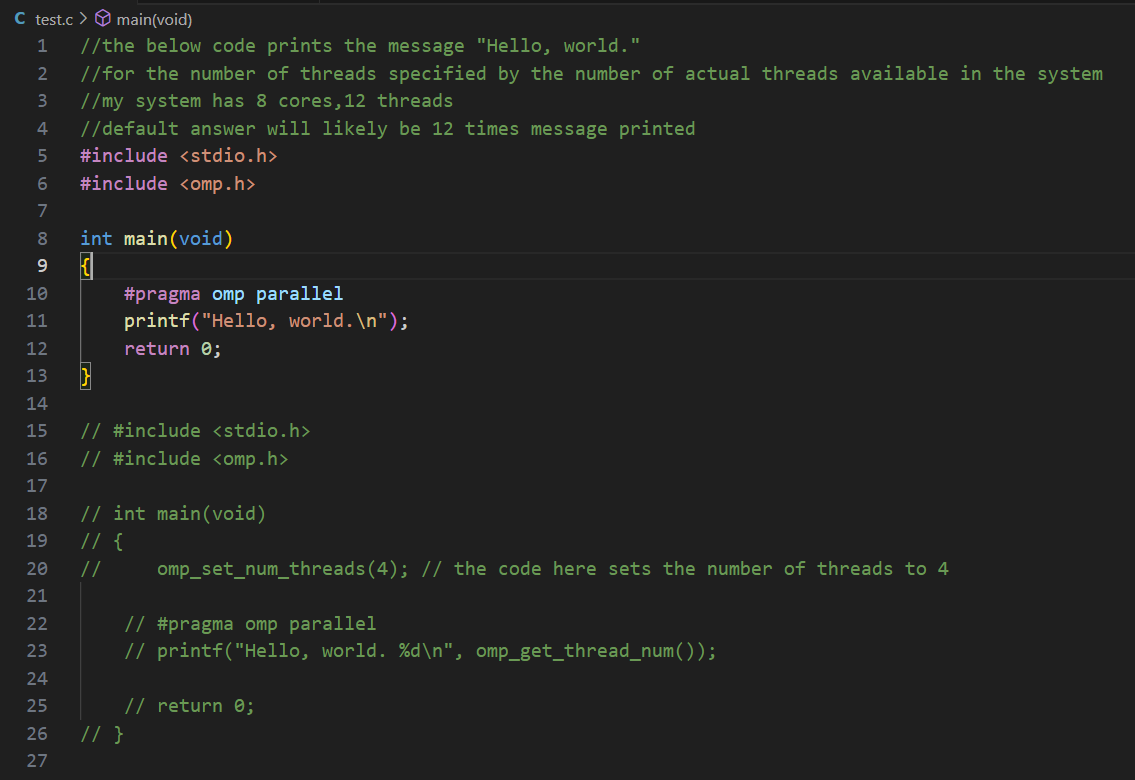
.\hello.exe

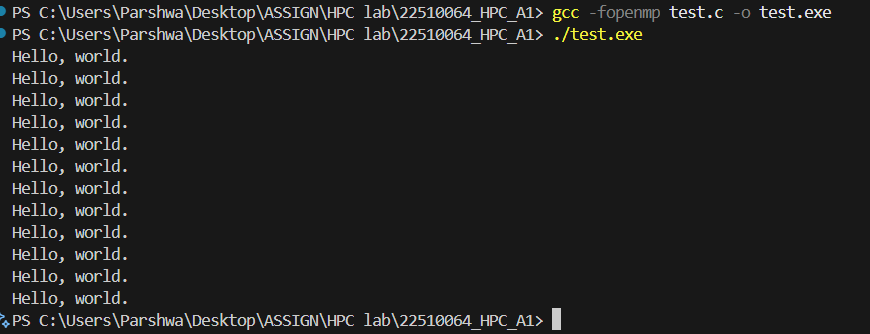


ANS.  
  
The mingw setup has previously been done successfully by me for codes in c,cpp to run.



  
The pthreads package is installed too…

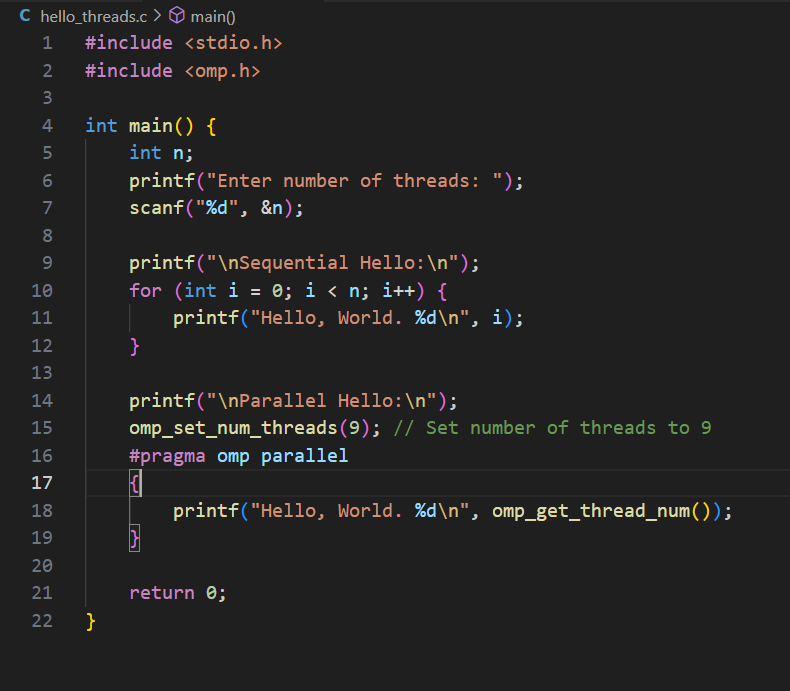


OUTPUT:  


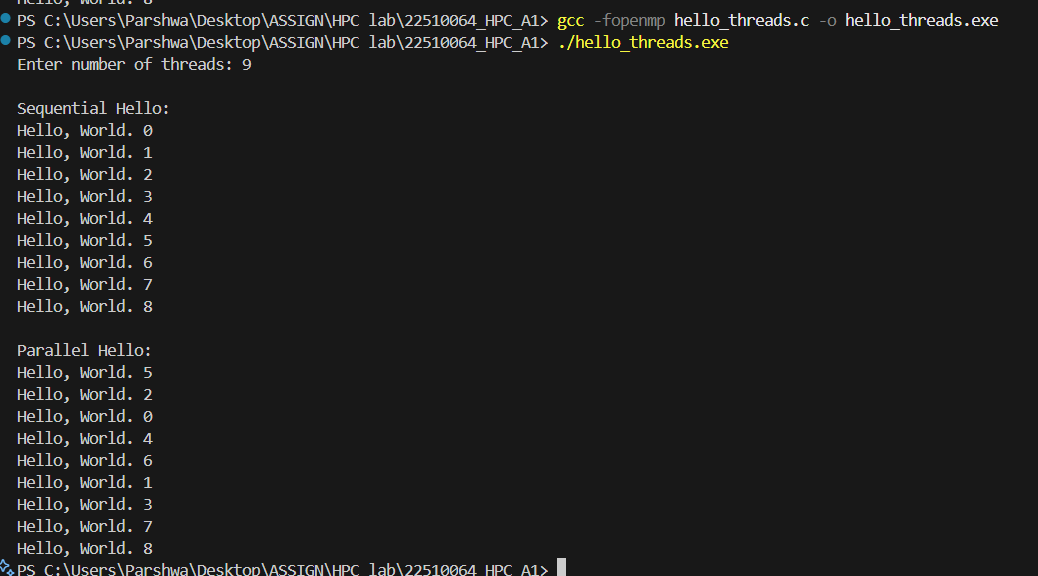
Problem Statement 2 – Print ‘Hello, World’ in Sequential and Parallel in OpenMP

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:

In the sequential part of the program, the messages are printed one after another in order because everything runs on a single thread. It's predictable and always prints the same way.

In the parallel part, multiple threads print messages at the same time. That’s why the output comes in a random order — it depends on how the system schedules the threads.

This shows how OpenMP allows different threads to work at the same time. It also helps us understand that when using parallel programming, output might not always be in order unless we manage the threads properly.

GitHub Link: make a public repository upload code of an assignment and paste its link here.

Problem statement 3: Calculate theoretical FLOPS of your system on which you are running the above codes.

Elaborate the parameters and show calculation.

**ANSWER:**  
To calculate the theoretical FLOPS of my system:

- Number of cores: 8  
- Number of threads : 12

- Clock speed: 2.00 GHz = 2.00 × 10⁹ Hz

- FLOPs per cycle (assumed): 8 (my CPU-i5 12450H)

Formula:

FLOPS = Number of Cores × Clock Speed × FLOPs per cycle

= 8 × 2.00 × 10⁹ × 8

= 128 × 10⁹ FLOPS

= **128 GFLOPS**

So, the theoretical performance of my system is approximately **128 GFLOPS**.