Course: High Performance Computing Lab

Practical No 1

PRN: 22510047

Name: Aadarsh Venkatesh Nandedkar

Batch: B8

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*

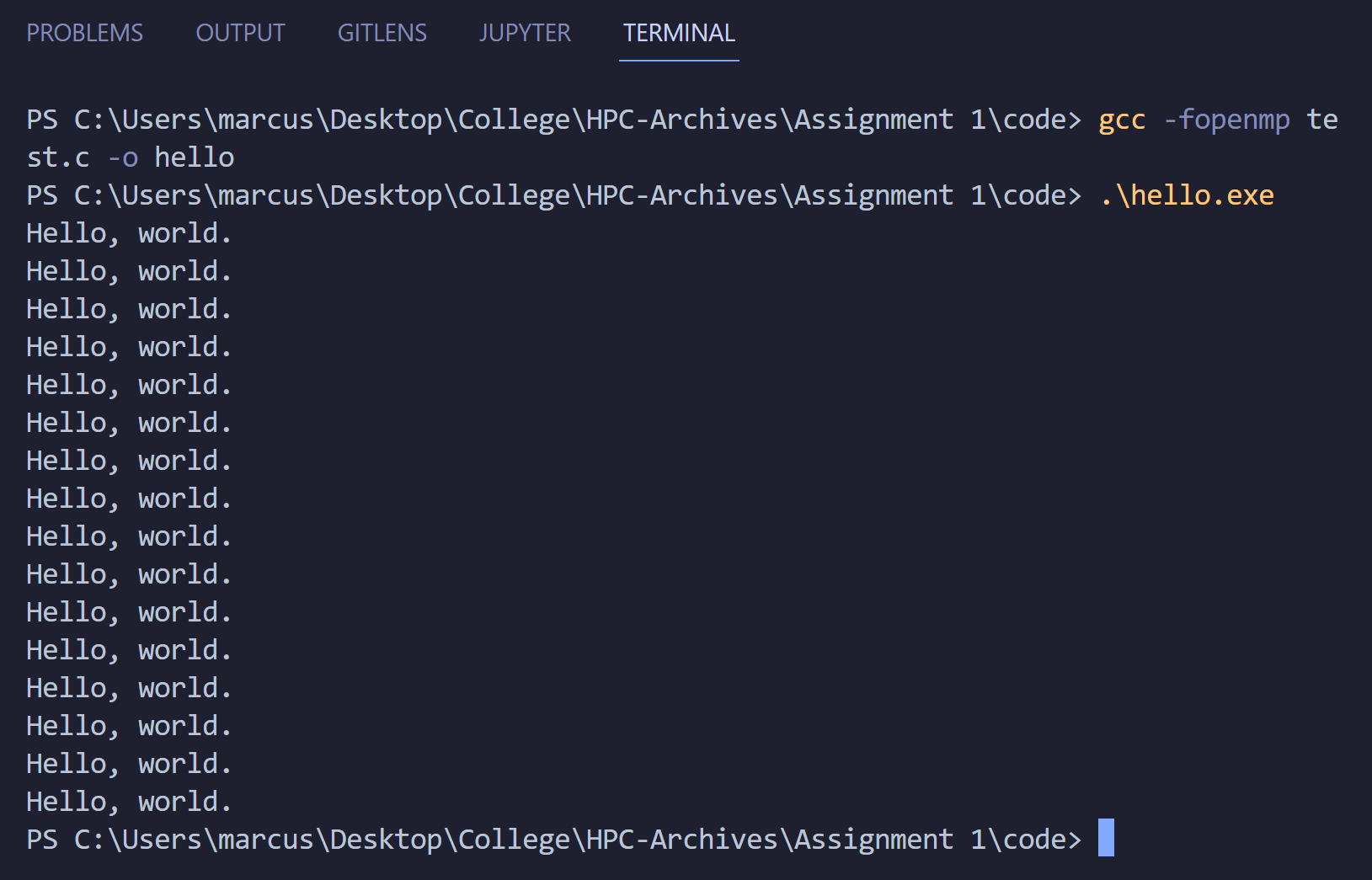
    printf("Hello, world.\n");

*return* 0;

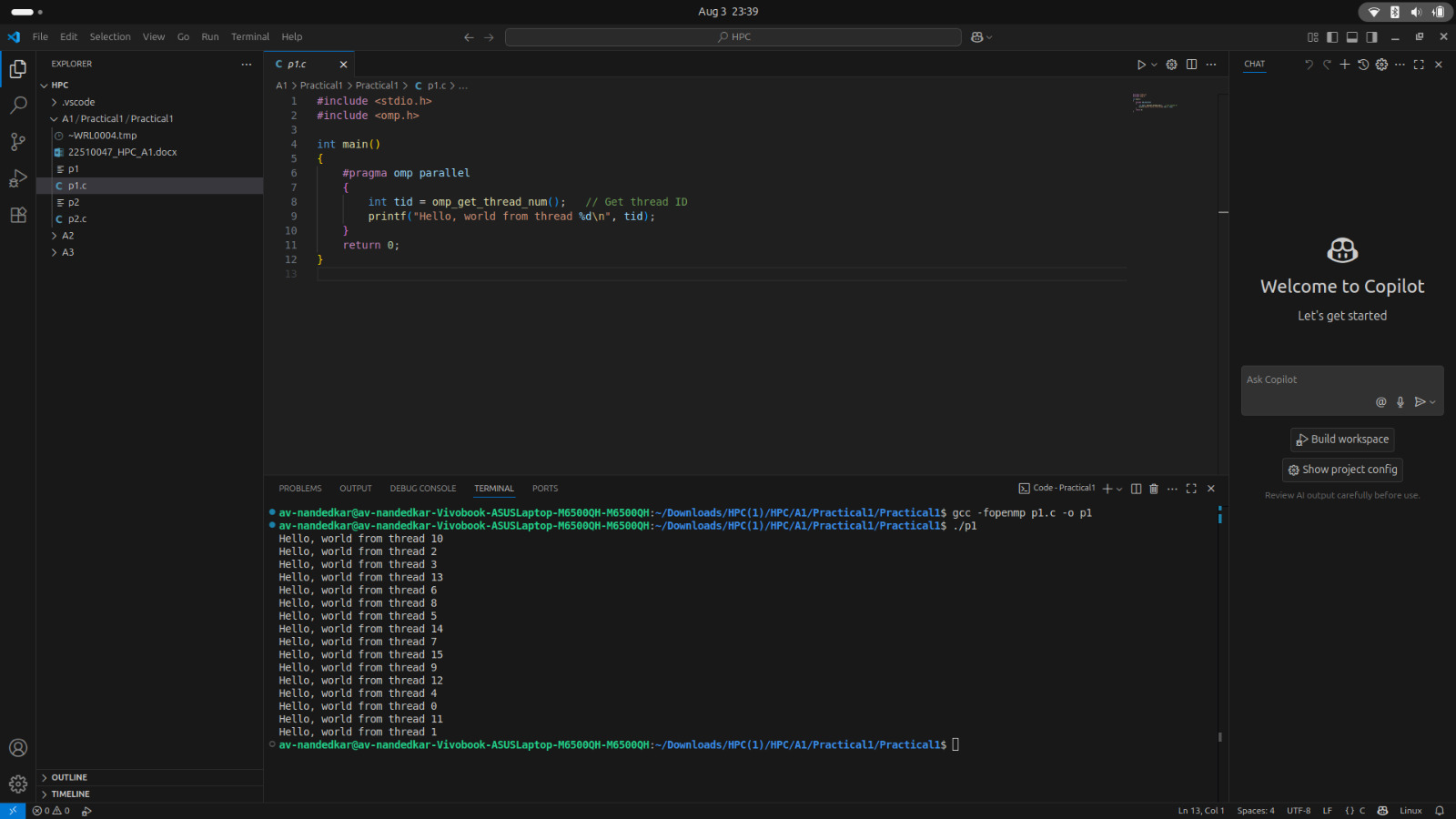
}

gcc -fopenmp test.c -o hello

.\hello.exe

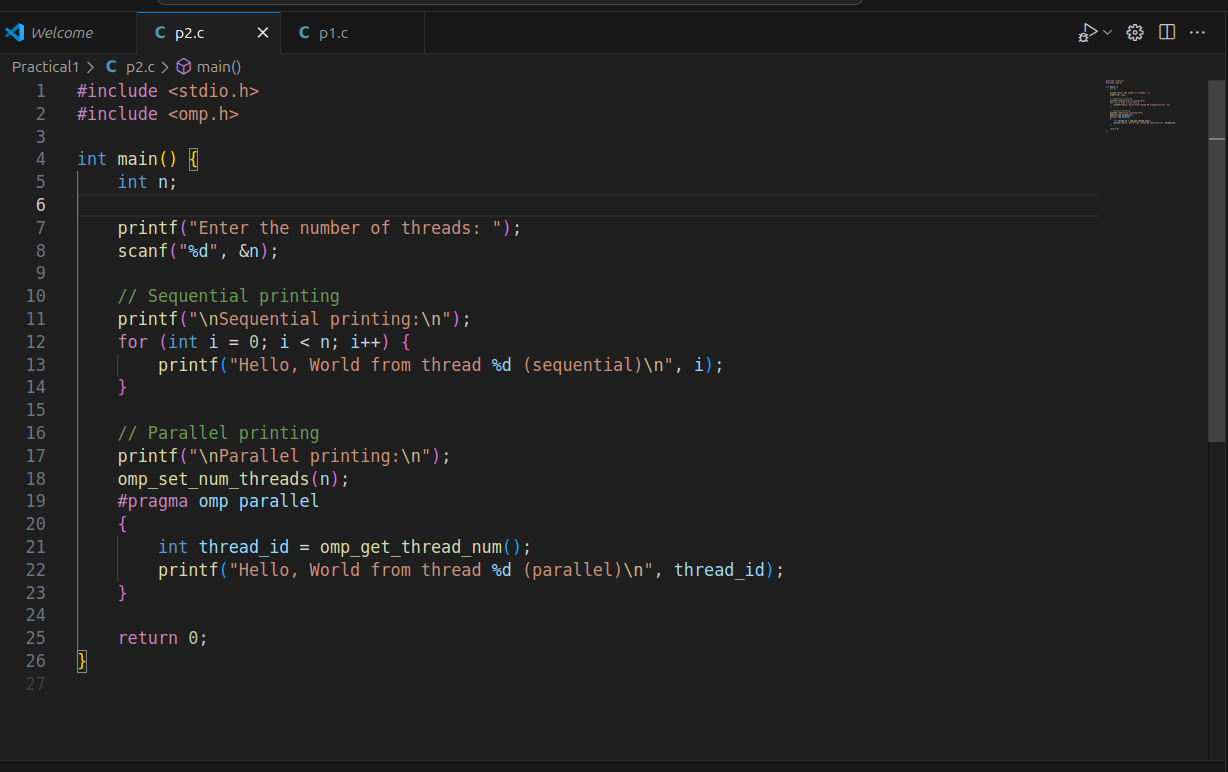


My code and Output:

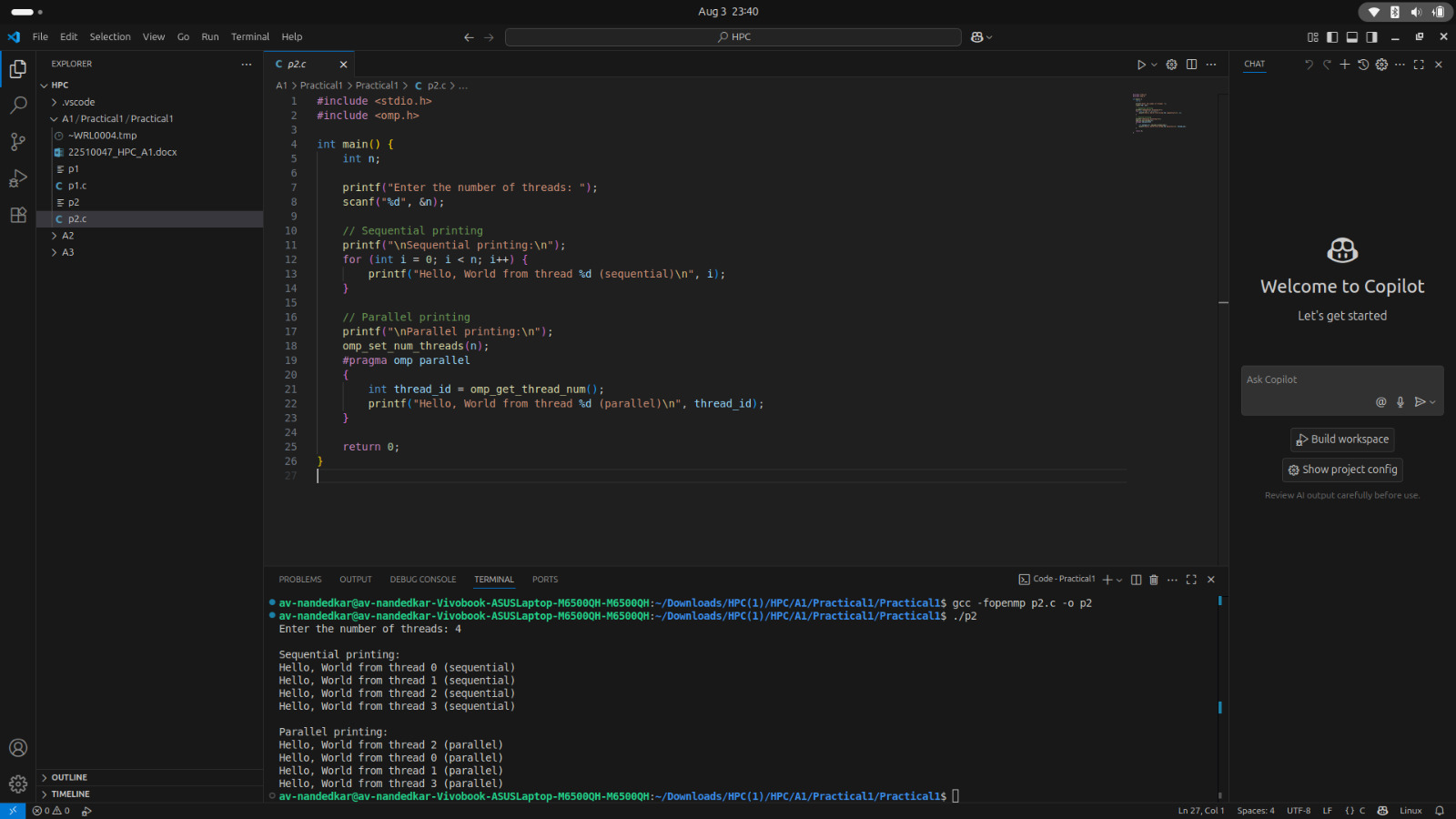
  
Problem Statement 2 – Print ‘Hello, World’ in Sequential and Parallel in OpenMP

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:

User enters the number of threads n.

**Sequential section:** Prints n lines in order using a single thread (main thread).

**Parallel section:** Spawns n threads using OpenMP. Each thread prints its ID. Output order is **not guaranteed** because threads run concurrently.

**Time complexity:** O(n) (work is split across threads in parallel section).

**Key concept:** Demonstrates the difference between sequential execution and parallel execution using OpenMP.

**Problem Statement 3: Calculate theoretical FLOPS of your system on which you are running the above codes.**  
Elaborate the parameters and show calculation.

**System Configuration:**  
**Processor:** AMD Ryzen 7 5800H  
**Cores:** 8  
**Base Clock Speed:** 3.2 GHz (Boost up to 4.4 GHz) — We’ll use 4.4 GHz for peak performance  
**FLOPs per cycle per core:**

16 FLOPs per cycle (for FP32, i.e., Single Precision)

8 FLOPs per cycle (for FP64, i.e., Double Precision)

### Formula:

**FLOPS = Number of Cores × Clock Speed × FLOPs per cycle (per core)**

### ****Single Precision (FP32)****

FLOPS = 8 cores × 4.4 GHz × 16 FLOPs/cycle  
FLOPS = 8 × 4.4 × 10⁹ × 16  
**= 563.2 GFLOPS**

### ****Double Precision (FP64)****

FLOPS = 8 cores × 4.4 GHz × 8 FLOPs/cycle  
FLOPS = 8 × 4.4 × 10⁹ × 8  
**= 281.6 GFLOPS**

The CPU can perform more **single-precision (FP32)** operations than **double-precision (FP64)** because:

Single precision uses 32 bits per number, allowing the processor to pack and compute more operations in a single instruction cycle.

Double precision requires 64 bits per number, reducing the number of simultaneous operations due to hardware constraints.

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

<https://github.com/av-nandedkar/HPC>