Course: High Performance Computing Lab

Practical No 1

**PRN: 21510069**

**Name: Harsh Bankat Karwa**

**Batch: B3**

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*

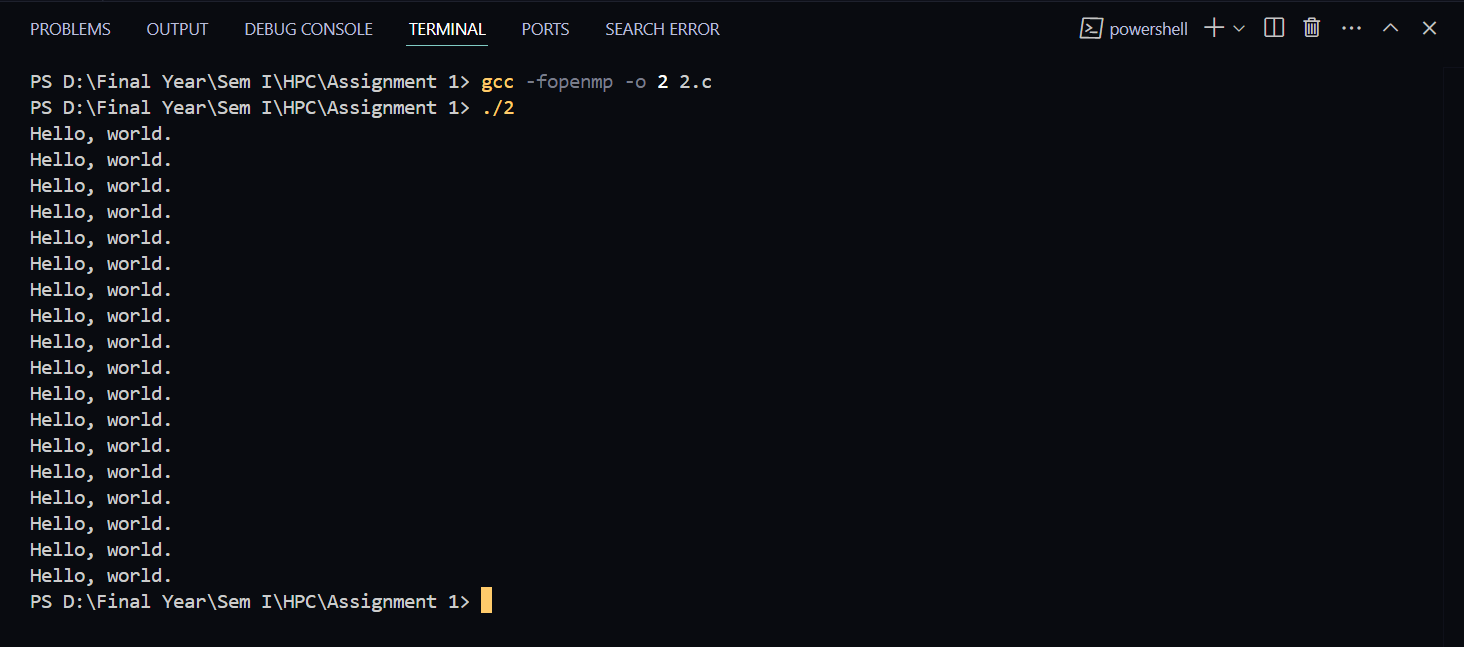
    printf("Hello, world.\n");

*return* 0;

}

gcc -fopenmp test.c -o hello

.\hello.exe



**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:

*#include* <stdio.h>

*#include* <omp.h>

int main(void)

{

    int num\_threads = 0;

    printf("Enter the number of threads: ");

    scanf("%d", &num\_threads);

    double start\_time = omp\_get\_wtime();

    printf("\nSequential printing:\n");

*for* (int i = 0; i < num\_threads; i++)

    {

        printf("Hello, World\n");

    }

    double end\_time = omp\_get\_wtime();

    printf("TIME FOR SEQUENTIAL PRINTING: %f\n", end\_time - start\_time);

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

    printf("\nParallel printing:\n");

    start\_time = omp\_get\_wtime();

*#pragma* omp parallel

    {

        printf("Hello, World from thread %d\n", omp\_get\_thread\_num());

    }

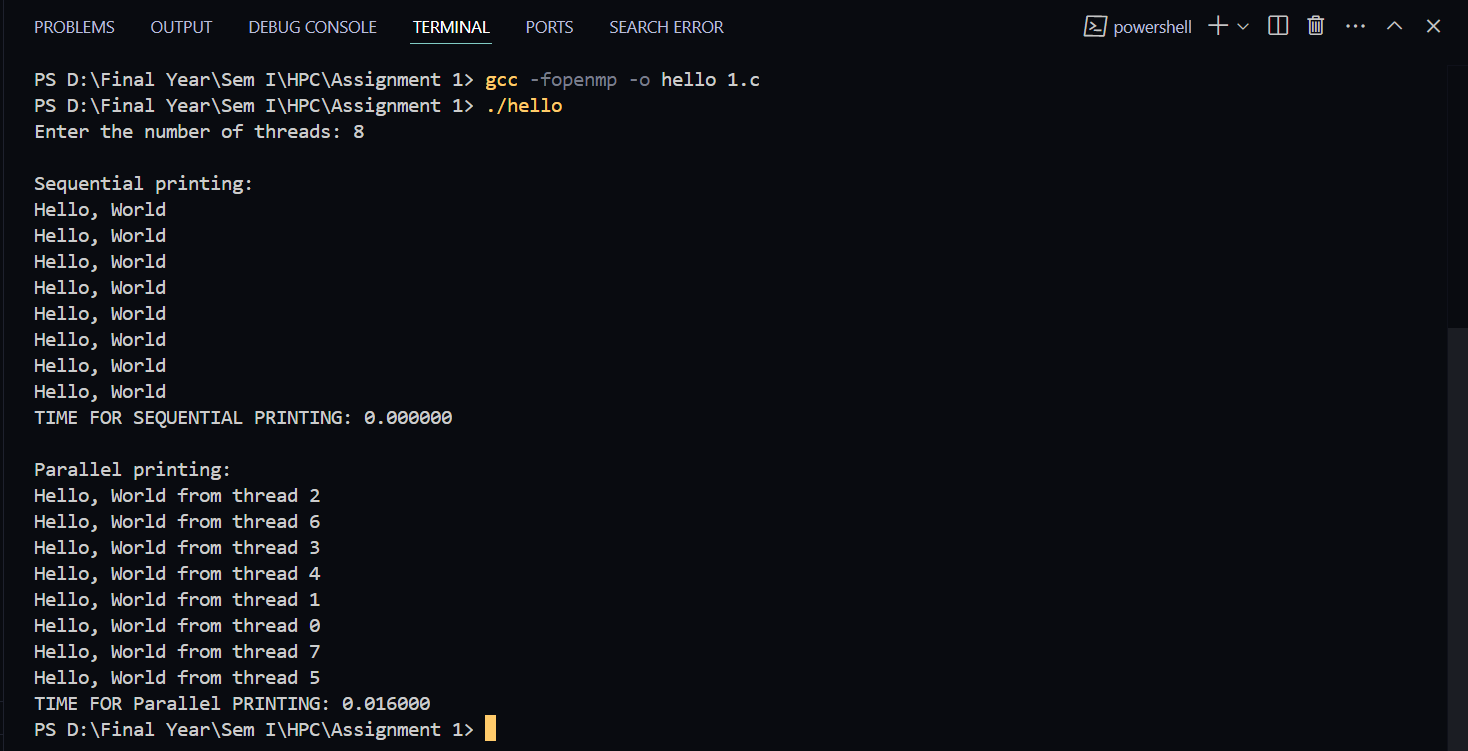
    end\_time = omp\_get\_wtime();

    printf("TIME FOR Parallel PRINTING: %f\n", end\_time - start\_time);

*return* 0;

}

Output snapshot:



Analysis:

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.**

cat /proc/cpuinfo

lscpu | grep “Mhz”

No of Flops: No of Core \* ClockSpeed \* IPC \* 10\*\*9

No of Core = 18

Clock Speed = 1.2GHz

IPC = 16

**ANS:= 345.6 GFLOPS**

**Github Link :** [**https://github.com/harsh-1503/High-Performance-Computing**](https://github.com/harsh-1503/High-Performance-Computing)