Course: **High Performance Computing Lab**

**Practical No 1**

PRN: 21510038

Name: Aniket Raju Ghotkar

Batch: B2

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()

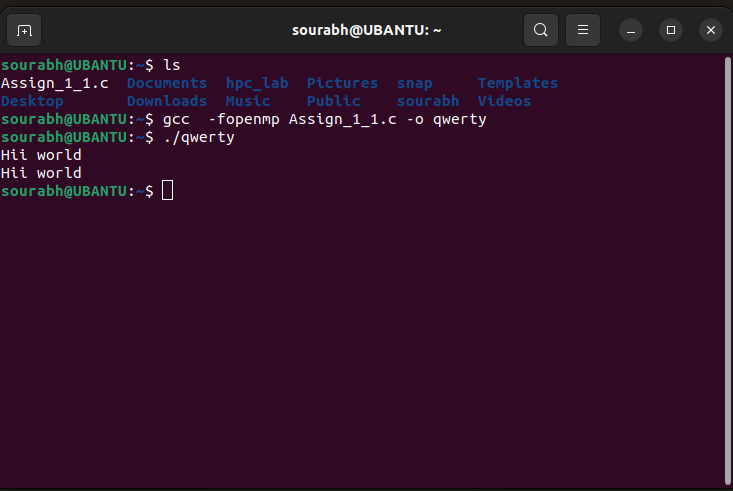
{

#pragma omp parallel

printf("Hii world\n");

return 0;

}



**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()

{

printf("Sequential : \n");

for(int i = 0 ; i<5 ; i++)

{

printf("omp \n");

}

printf("\n");

printf("Parallel : \n");

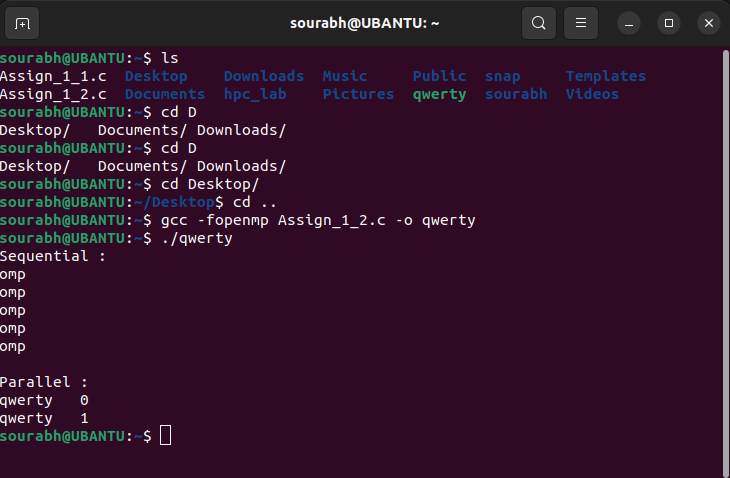
#pragma omp parallel

printf("qwerty \t %d \n",omp\_get\_thread\_num());

return 0;

}

Output snapshot:



Analysis:

This code demonstrates the basic use of OpenMP to parallelize a section of code. The sequential section runs as expected, while the parallel section utilizes multiple threads to print their respective thread numbers, illustrating the concurrent execution of code.

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.

**Ans.** Let's calculate the theoretical FLOPS of your system.

### Extracted Information

* Number of Cores: 2
* Clock Speed: 2496 MHZ

### FLOPs per Cycle per Core

Assuming the CPU supports AVX2 instructions, which typically allows 2 FLOPs per cycle per core.

### Calculation

Using the formula: FLOPS Speed of Cores per Cycle per CoreTotal FLOPS=Clock Speed×Number of Cores×FLOPs per Cycle per Core

Substitute the values:

* Clock Speed:2496 MHZ
* Number of Cores: 2
* FLOPs per Cycle per Core: 2

### Calculation Steps

1. Convert the clock speed to cycles per second: per second

2.496GHz = 2.496 × 109 cycles per second

1. Use the formula to calculate the total FLOPS: FLOPS

Total FLOPS = 2.496 × 109 cycles/second × 2 cores × 2 FLOPs/cycle

1. Perform the multiplication: FLOPS

TotalFLOPS = 2.496 × 2 × 2 × 109

1. Simplify the calculation:

FLOPSTotalFLOPS = 9.984 × 109

 FLOPSTotalFLOPS = 9.984 GFLOPS.

### Conclusion

Based on the provided system data, the theoretical peak performance of your CPU in terms of floating-point operations per second (FLOPS) is approximately 9.984 GFLOPS. This represents the maximum theoretical performance under ideal conditions.