Course: High Performance Computing Lab

Practical No 1

PRN: 21510036

Name: Tushar Rathod

Batch: B7

Title: Introduction to OpenMP

Problem Statement 1 – Demonstrate Installation and Running of OpenMP code in C

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 <omp.h>

#include <stdio.h>

#include <stdlib.h>

int main(int *argc*, char \**argv*[]) {

#pragma omp parallel

{

printf("thread No. %d Hello World\n", omp\_get\_thread\_num());

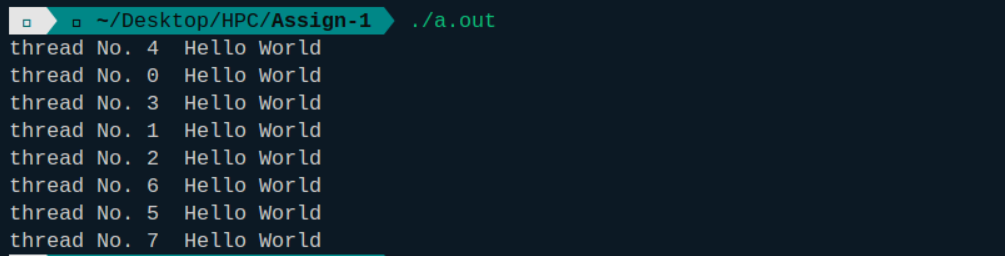
}

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:

Sequential -

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

int main(int *argc*, char \**argv*[]) {

printf("thread No. %d Hello World\n", omp\_get\_thread\_num());

return 0;

}

Output -



Parallel -

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

int main(int *argc*, char \**argv*[]) {

#pragma omp parallel

{

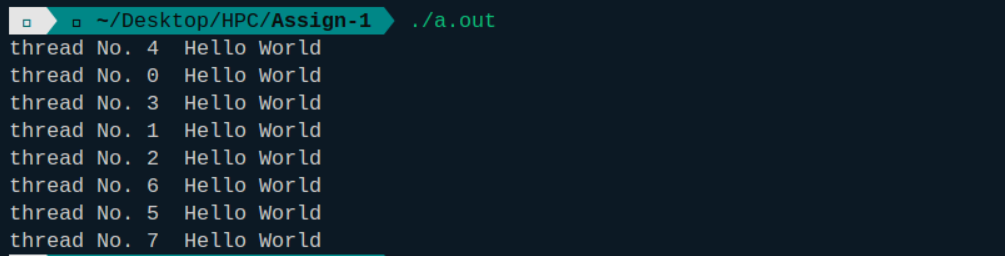
printf("thread No. %d Hello World\n", omp\_get\_thread\_num());

}

return 0;

}

Output -



Analysis:

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

*// FLOPS=Number of cores × Clock speed (GHz) × Number of FLOPs per cycle*

*// FLOPS= 4 × 2.4 × 8 = 76.8 GFLOPS*

Elaborate the parameters and show calculation -

Using lscpu i got the number of sockets, number of cores per socket, threads per core.

Which were -

Chip sockets = 1

Total cores = 4

Total Threads = 8