Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Course: High Performance Computing Lab

Practical No 1

PRN: 22520007

Name: Manish Namdev Barage

Batch: B6

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.

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,

Program: Hello world basic program in C.

```
C V_S_add.c    C hello.c
Assign1 > C hello.c
#include <stdio.h>
#include <omp.h>

int main(void)
{
    #pragma omp parallel
    printf("Hello, world.\n");
    return 0;
}
```

```
wlug@wlug-optiplex:~/Desktop/22520007/Assign1$ ./hello
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
wlug@wlug-optiplex:~/Desktop/22520007/Assign1$
```

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.

V_S_add.c | hello.c | q2.c

Assign1 > q2.c

```c
#include<stdio.h>
#include<omp.h>

int main()
{
    int n;
    printf("Enter the number of threads : ");
    scanf("%d", &n);

    printf("Sequential : \n");

    for(int i=0;i<n;i++)
    {
        printf("Hello WOrld !\n");
    }

    omp_set_num_threads(n);
    printf("Parallel : \n");
    #pragma omp parallel
    {
        int id = omp_get_thread_num();
        printf("Hello World : Thread Id : %d \n",id);
    }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Hello, world.
Hello, world.
wlug@wlug-optiplex:~/Desktop/22520007/Assign1$ gcc -fopenmp -o q2 q2.c
wlug@wlug-optiplex:~/Desktop/22520007/Assign1$ ./q2
Enter the number of threads : 6
Sequential :
Hello WOrld !
Hello WOrld !
Hello WOrld !
Hello WOrld !
Hello WOrld !
Hello WOrld !
Parallel :
Hello World : Thread Id : 0
Hello World : Thread Id : 4
Hello World : Thread Id : 3
Hello World : Thread Id : 1
Hello World : Thread Id : 2
Hello World : Thread Id : 5
wlug@wlug-optiplex:~/Desktop/22520007/Assign1$
```

Analysis: We can provide how many threads we want to execute a task. Every thread runs the task parallely. The sequence of threads is random and any thread is selected for the execution randomly.

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

Code snapshot:

Assign1 > C flops.c

```c
#include <stdio.h>

int main() {

    double clock_speed_ghz = 3.2;
    int num_cores = 6;
    int ipc = 4;
    int flop_per_instruction = 2;


    double clock_speed_hz = clock_speed_ghz * 1e9;


    double flops = clock_speed_hz * num_cores * ipc * flop_per_instruction;


    printf("Theoretical FLOPS: %.2e\n", flops);

    return 0;
}
```

Output:

```
Theoretical FLOPS: 1.54e+11
wlug@wlug-optiplex:~/Desktop/22520007/Assign1$
```

Parameters:

1. FLOPS: Floating point operations per seconds, which means how many floating point arithmetic operations such as addition, subtraction, multiplication or division can be executed by CPU in one second.

FLOPS = (Time needed for calculation) / clocks per second

2. CPU clock speed: 2.4 Ghz

Can be found using command: *lscpu | grep "Ghz"*

This shows the number of clock cycles performed by the CPU