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

Course: High-Performance Computing Lab

Practical No 1

PRN: 21510111

Name: Siddharth Salunkhe

Batch: B-5

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)

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,

```c
#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
```

```
PROBLEMS    OUTPUT    GITLENS    JUPYTER    TERMINAL

PS C:\Users\marcus\Desktop\College\HPC-Archives\Assignment 1\code> gcc -fopenmp te
st.c -o hello
PS C:\Users\marcus\Desktop\College\HPC-Archives\Assignment 1\code> .\hello.exe
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
PS C:\Users\marcus\Desktop\College\HPC-Archives\Assignment 1\code>
```

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(int argc, char *argv[])
{
    int numThreads;
    printf("Enter the number of threads : ");
    scanf("%d", &numThreads);

    // Sequential Output
    printf("\nSequential Output :\n");
    for(int i=0; i<numThreads; i++){
        printf("Hello World!!\n");
    }

    // Setting number of threads
    omp_set_num_threads(numThreads);

    // Prallel Output
    printf("\nPrallel Output\n");
    #pragma omp parallel
    {
        printf("Hello World for thread : %d\n", omp_get_thread_num());
    }

    return 0;
}
```

Output snapshot:

```
[~/acad/hpc_lab/as1]$ gcc -fopenmp assignment.c && ./a.out
Enter the number of threads : 5

Sequential Output :
Hello World!!
Hello World!!
Hello World!!
Hello World!!
Hello World!!

Prallel Output
Hello World for thread : 2
Hello World for thread : 4
Hello World for thread : 0
Hello World for thread : 3
Hello World for thread : 1
[~/acad/hpc_lab/as1]$
```

Analysis:

Sequential Execution Time: Since this involves a simple loop, it will depend on the number of threads specified. The time complexity is O(n), where `n` is the number of threads.

Parallel Execution Time: The time taken should ideally be less than the sequential time, depending on the overhead of creating threads and synchronization. It should also decrease as the number of threads increases up to a certain point.

But in practice due to some other affecting factor parallel execution is taking more time :

Code run for 100 threads:

```
Hello World for thread : 74
Time taked for sequtial code : 0.000437 seconds
Time taken for parallel code : 0.008628 seconds
[~/acad/hpc_lab/as1]$
```

GitHub Link: https://github.com/Sidd-77/hpc-lab

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

Ans:

FLOPS=Number of Cores (C)×Clock Speed (F)×FLOPs per Cycle per Core (IPC)

Where:

C = Number of Cores

F = Clock Speed in Hz

IPC = FLOPs per Cycle per Core

For Intel i5 10th gen

C = 6,

F = 4.30 GHz = 4.30 * 10^9 Hz

IPC = AVX2 which provides 16 FLOPs per cycle per core.

FLOPS = 6 * 4.30 * 10^9 * 16

FLOPS = 412.8 * 10^9

FLOPS = 412.8 GFLOPS

Actual flops : 240 GFLOPS