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

Course: High Performance Computing Lab

Practical No 1

PRN: 22510057

Name: Ashutosh Gundu Birje

Batch: B8

Title of practical:

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.

Class: Final Year (CSE) Year: 2025-26 Semester: 1

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.
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:

Assignment_No_1 > C 02_.c

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

int main() {
    int num_threads;

    printf("Enter number of threads: ");
    scanf("%d", &num_threads);

    printf("\nSequential Hello, World:\n");
    for (int i = 0; i < num_threads; i++) {
        printf("Hello, World from thread %d (Sequential)\n", i);
    }

    omp_set_num_threads(num_threads);

    printf("\nParallel Hello, World using OpenMP:\n");
    #pragma omp parallel
    {
        int thread_id = omp_get_thread_num();
        printf("Hello, World from thread %d (Parallel)\n", thread_id);
    }

    return 0;
}
```

Output snapshot:

```
ashutosh@ash-880:~/Desktop/HPC-LAB$ cd Assignment_No_1
ashutosh@ash-880:~/Desktop/HPC-LAB/Assignment_No_1$ gcc -fopenmp 02_.c -o 02.exe
ashutosh@ash-880:~/Desktop/HPC-LAB/Assignment_No_1$ ./02.exe
Enter number of threads: 4

Sequential Hello, World:
Hello, World from thread 0 (Sequential)
Hello, World from thread 1 (Sequential)
Hello, World from thread 2 (Sequential)
Hello, World from thread 3 (Sequential)

Parallel Hello, World using OpenMP:
Hello, World from thread 0 (Parallel)
Hello, World from thread 2 (Parallel)
Hello, World from thread 3 (Parallel)
Hello, World from thread 1 (Parallel)
ashutosh@ash-880:~/Desktop/HPC-LAB/Assignment_No_1$
```

Analysis:

Sequential Section:

Executes on main thread (single thread).

Iterates from 0 to num_threads-1, printing messages sequentially.

Parallel Section:

OpenMP creates num_threads threads.

Each thread independently prints its message.

Thread execution is concurrent — may lead to out-of-order output.

Key Points:

Use of omp_set_num_threads() to set runtime thread count.

#pragma omp parallel to parallelize a block.

omp_get_thread_num() returns unique thread ID.

Problem statement 3

Calculate theoretical FLOPS of your system on which you are running the above codes.

FLOPS (Floating Point Operations Per Second)

It is a measure of a computer's performance, especially in scientific computations.

Formula

FLOPS = Number of cores × Clock speed × FLOPs per cycle

Calculation

FLOPS = 8 cores × 2.3 × 10^9 cycles/sec × 16 FLOPs/cycle

= 294.4 × 10^9 FLOPS

= 294.4 GFLOPS (GigaFLOPS)

Parameter

Parameter	Example (replace with actual system values)
CPU Name	Intel Core i7-12700H
Base Clock Speed	2.3 GHz = 2.3×10^9 cycles/sec
Cores	8 Performance Cores + 4 Efficiency Cores = 12 total
FLOPs per cycle	16 (assuming AVX-512 or FMA with 512-bit vector width)

GitHub Link:
https://github.com/Ashutoshbirje/HPCLAB/tree/master/Assignment_No_1