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

Course: High Performance Computing Lab

Practical No 1

PRN: 21510042

Name:Omkar Rajesh Auti

Batch:CSE

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 <iostream>

#include <omp.h>
#include <chrono>

int main() {
const int numIterations = 1000;

// Sequential execution
auto start_time = std::chrono::high_resolution_clock::now();
for (int i = 0; i < numIterations; ++i) {
// Print "Hello, World" for each iteration (can be commented out for
performance measurement)
}
auto end_time = std::chrono::high_resolution_clock::now();
std::chrono::duration<double, std::milli> sequential_duration =
end_time - start_time;
std::cout << "Sequential Time taken: " << sequential_duration.count()
<< " milliseconds" << std::endl;

// Parallel execution
start_time = std::chrono::high_resolution_clock::now();

#pragma omp parallel
{
#pragma omp for
for (int i = 0; i < numIterations; ++i) {
// Print "Hello, World" for each iteration (can be commented out for
performance measurement)
}
}

end_time = std::chrono::high_resolution_clock::now();
```

```
std::chrono::duration<double, std::milli> parallel_duration = end_time
- start_time;
std::cout << "Parallel Time taken: " << parallel_duration.count() << "
milliseconds" << std::endl;

return 0;
}
```

Output snapshot:

```
ubuntu@ubuntu-VirtualBox:~/Documents/Assignment01$ g++ -fopenmp -o a 01_02_a.cpp
ubuntu@ubuntu-VirtualBox:~/Documents/Assignment01$ ./a
Sequential Time taken: 0.001873 milliseconds
Parallel Time taken: 1.77879 milliseconds
ubuntu@ubuntu-VirtualBox:~/Documents/Assignment01$
```

Analysis:

Parallel Time > Sequential Time
(For smaller size of iterations)

GitHub Link: make a public repository upload code of an assignment and paste its link here.

https://github.com/omkarauti11/HPC_LAB

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

theoreticalFLOPS = clockSpeedHz * numCores * operationsPerCycle;

Code :

```
#include <iostream>

#include <cmath>
#include <chrono>

int main() {
// Define the number of floating-point operations
long long numOperations = 1e8; // 100 million operations

// Sequential execution
auto start = std::chrono::high_resolution_clock::now();

double result = 0.0;
for (long long i = 0; i < numOperations; ++i) {
result += std::sin(i * 0.1); // Example floating-point operation
}

auto end = std::chrono::high_resolution_clock::now();
std::chrono::duration<double, std::milli> elapsed = end - start;
double sequential_time_ms = elapsed.count();
double sequential_time_s = sequential_time_ms / 1000.0; // Convert
milliseconds to seconds
double sequential_flops = numOperations / sequential_time_s; // FLOPS =
operations / time (in seconds)
double sequential_gflops = sequential_flops / 1e9; // Convert to GFLOPS

std::cout << "Sequential Execution:" << std::endl;
std::cout << "Elapsed Time: " << sequential_time_ms << " milliseconds"
<< std::endl;
std::cout << "Estimated FLOPS: " << sequential_flops << " FLOPS" <<
std::endl;
std::cout << "Estimated GFLOPS: " << sequential_gflops << " GFLOPS" <<
std::endl;

return 0;
}
```

Output:

```
ubuntu@ubuntu-VirtualBox:~/Documents/Assignment01$ g++ -fopenmp -o a 01_03_a.cpp
ubuntu@ubuntu-VirtualBox:~/Documents/Assignment01$ ./a
Sequential Execution:
Elapsed Time: 2026.81 milliseconds
Estimated FLOPS: 4.93385e+07 FLOPS
Estimated GFLOPS: 0.0493385 GFLOPS
ubuntu@ubuntu-VirtualBox:~/Documents/Assignment01$
```

Elaborate the parameters and show calculation.

Example for a GPU:

- **Clock Speed:** 1.5 GHz (1.5 × 10^9 Hz)
- **Number of Cores:** 256
- **Floating-Point Operations per Clock Cycle:** 2

Theoretical FLOPS (GPU) = 1.5 × 10^9 Hz × 256 Cores × 2 FLOP/Cycle = 768 × 10^9 FLOPS = 768 GFLOPS