Name: Rahul Bhati

Roll no: 2223416 Sub: HPC

ASSIGNMENT Lab

1. Write an OpenMP program to print Hello world with thread ID.

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

int main() {
    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        int nthreads = omp_get_num_threads();
        printf("Hello World from thread %d out of %d\n", tid, nthreads);
    }
    return 0;
}
```

```
[mit108@login01 ~]$ gcc -fopenmp hello_omp.c -o hello_omp
[mit108@login01 ~]$ export OMP_NUM_THREADS=4
[mit108@login01 ~]$ ./hello_omp
Hello World from thread 0 out of 4
Hello World from thread 1 out of 4
Hello World from thread 3 out of 4
Hello World from thread 2 out of 4
[mit108@login01 ~]$
```

2. Write your first Parallel Program, with which you should be able to print your NAME from 4 underline cores.

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

int main() {
    // Set up a parallel region with 4 threads
    #pragma omp parallel num_threads(4)
    {
        int tid = omp_get_thread_num();
        printf("Hello, I am THREAD %d printing NAME: RAJ \n", tid);
    }
    return 0;
}
```

```
[mit108@login01 ~]$ nano name_parallel.c
[mit108@login01 ~]$ gcc -fopenmp name_parallel.c -o name_parallel
[mit108@login01 ~]$ ./name_parallel
Hello, I am THREAD 0 printing NAME: RAJ
Hello, I am THREAD 2 printing NAME: RAJ
Hello, I am THREAD 1 printing NAME: RAJ
Hello, I am THREAD 3 printing NAME: RAJ
[mit108@login01 ~]$
```

3. Write a C program using OpenMP with 4 threads to demonstrate the behavior of the firstprivate clause. Initialize an integer variable val = 1234 outside the parallel region and print its value. Inside the parallel region, declare val as firstprivate, print its initial value in each thread, increment it by 1, and print the updated value. Finally, print the value of val outside the parallel region. Explain what happens to val inside and outside the parallel region and why.

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

int main() {
    int val = 1234;

    // Print initial value before entering parallel region
    printf("Initial value of val (outside parallel region): %d\n", val);

    // Parallel region with 4 threads, using firstprivate(val)
    #pragma omp parallel num_threads(4) firstprivate(val)
    {
        int tid = omp_get_thread_num();

        // Print current value of val (copied from outside)
        printf("Thread %d: initial val = %d\n", tid, val);

        // Increment val
        val++;

        // Print updated value of val (private copy per thread)
        printf("Thread %d: updated val = %d\n", tid, val);
    }

    // Print final value of val after parallel region
    printf("Final value of val (outside parallel region): %d\n", val);

    return 0;
}
```

```
[mit108@login01 ~]$ nano private_demo.c
[mit108@login01 ~]$ gcc -fopenmp private_demo.c -o private_demo
[mit108@login01 ~]$ ./private_demo
Initial value of val (outside parallel region): 1234
Thread 0: initial val = 1234
Thread 0: updated val = 1235
Thread 2: initial val = 1234
Thread 2: updated val = 1235
Thread 3: initial val = 1234
Thread 3: updated val = 1235
Thread 1: initial val = 1234
Thread 1: updated val = 1235
Final value of val (outside parallel region): 1234
[mit108@login01 ~]$
```

4. Write a Parallel C program where the iterations of a loop should be scheduled statically across the team of threads. A thread should perform CHUNK iterations at a time before being scheduled for the next CHUNK of work.

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

int main() {
    int N = 20;          // total iterations
    int CHUNK = 4;       // chunk size
    int i;

    printf("Static scheduling demo with CHUNK = %d\n", CHUNK);

    #pragma omp parallel for schedule(static,4) num_threads(4)
    for (i = 0; i < N; i++) {
        int tid = omp_get_thread_num();
        printf("Thread %d is processing iteration %d\n", tid, i);
    }

    return 0;
}
```

```
Last login: Mon Aug 18 12:20:29 2025 from 106.213.86.207
[mit108@login02 ~]$ nano static_schedule.c
[mit108@login02 ~]$ gcc -fopenmp static_schedule.c -o static_schedule
[mit108@login02 ~]$ ./static_schedule
Static scheduling demo with CHUNK = 4
Thread 0 is processing iteration 0
Thread 3 is processing iteration 12
Thread 3 is processing iteration 13
Thread 1 is processing iteration 4
Thread 1 is processing iteration 5
Thread 1 is processing iteration 6
Thread 1 is processing iteration 7
Thread 3 is processing iteration 14
Thread 3 is processing iteration 15
Thread 0 is processing iteration 1
Thread 0 is processing iteration 2
Thread 0 is processing iteration 3
Thread 0 is processing iteration 16
Thread 0 is processing iteration 17
Thread 0 is processing iteration 18
Thread 0 is processing iteration 19
Thread 2 is processing iteration 8
Thread 2 is processing iteration 9
Thread 2 is processing iteration 10
Thread 2 is processing iteration 11
[mit108@login02 ~]$
```

5. Write a C program utilizing OpenMP directives to demonstrate the behavior of the private clause. Steps to follow : ▪ Open text editor. ▪ write the below program in it. ▪ Save the file with .c extentation. ▪ Compile and execuate with given commands.

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

int main() {
    int val = 1234;

    // Print initial value before entering parallel region
    printf("Initial value of val (outside parallel region): %d\n", val);

    // Parallel region with private(val)
    #pragma omp parallel num_threads(4) private(val)
    {
        int tid = omp_get_thread_num();

        // val is uninitialized (garbage value) because of private
        printf("Thread %d: private val before assignment = %d\n", tid, val);

        // Assign a value based on thread ID
        val = tid + 100;
        printf("Thread %d: private val after assignment = %d\n", tid, val);
    }

    // Outside region val is still unchanged
    printf("Final value of val (outside parallel region): %d\n", val);

    return 0;
}
```

```
[mit108@login02 ~]$ nano private_demo.c
[mit108@login02 ~]$ gcc -fopenmp private_demo.c -o private_demo
[mit108@login02 ~]$ ./private_demo
Initial value of val (outside parallel region): 1234
Thread 0: private val before assignment = 0
Thread 3: private val before assignment = 0
Thread 3: private val after assignment = 103
Thread 0: private val after assignment = 100
Thread 1: private val before assignment = 0
Thread 1: private val after assignment = 101
Thread 2: private val before assignment = 0
Thread 2: private val after assignment = 102
Final value of val (outside parallel region): 1234
```

6. Write a Parallel C program which should print the series of 2 and 4. Make sure both should be executed by different threads.

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

int main() {
    int N = 10;   // length of series

    // Parallel region with 2 threads
    #pragma omp parallel num_threads(2)
    {
        int tid = omp_get_thread_num();

        if (tid == 0) {  // Thread 0 prints multiples of 2
            printf("Thread %d printing series of 2:\n", tid);
            for (int i = 1; i <= N; i++) {
                printf("%d ", 2 * i);
            }
            printf("\n");
        }

        if (tid == 1) {  // Thread 1 prints multiples of 4
            printf("Thread %d printing series of 4:\n", tid);
            for (int i = 1; i <= N; i++) {
                printf("%d ", 4 * i);
            }
            printf("\n");
        }
    }

    return 0;
}
```

```
[mit108@login02 ~]$ nano series_threads.c
[mit108@login02 ~]$ gcc -fopenmp series_threads.c -o series_threads
[mit108@login02 ~]$ ./series_threads
Thread 0 printing series of 2:
2 4 6 8 10 12 14 16 18 20
Thread 1 printing series of 4:
4 8 12 16 20 24 28 32 36 40
[mit108@login02 ~]$
```

7. Familiarization with Linux commands.

1️⃣ Working with Files & Directories

- **Show current directory : pwd**

```
[mit108@login01 ~]$ pwd
/home/mit108
[mit108@login01 ~]$
```

List files

```
[mit108@login01 ~]$ ls
demo                  hello_error.txt     private_demo
demo1                 hello_omp           private_demo.c
dot_product           hello_omp.c         reduction_demo
dotproduct            hello_output.txt    reduction_demo.c
dot_product.c         hits_openmp         reduction_job.slurm
dotproduct.c          hits_openmp.c       script.sh
firstprivate_demo     lastprivate_demo    series_thread.c
firstprivate_demo.c   lastprivate_demo.c  series_threads
hello                 multiples           series_threads.c
hello1                multiples.c         static_schedule
hello1.c              name_parallel       static_schedule.c
hello.c               name_parallel.c     sum.cpp
[mit108@login01 ~]$
```

8. Familiarization with SLURM commands

Submit a Job

Submit a batch script to SLURM:

```
sbatch reduction_job.slurm
```

Check Job Queue

```
squeue
```

Cancel a Job

```
scancel 12345
```

Job Status and Info

```
scontrol show job 12345
```

Check cluster/node status

```
sinfo
```