Indian Institute of Information Technology Surat

Lab Report on

High Performance Computing

(CS 503) Practical

Submitted by

Aditya Kumar(UI22CS03)

Course Faculty
Dr. Sachin D. Patil

Department of Computer Science and Engineering
Indian Institute of Information Technology Surat
Gujarat-394190, India

August-2024

Table of Contents

Assignment 4

AIM:- Write an OpenMP program to test the Schedule clause, Section clause, and synchronization clause.

Introduction

OpenMP (Open Multi-Processing) is a parallel programming model used to write programs that can execute multiple threads concurrently on shared memory architectures. This report demonstrates three fundamental OpenMP constructs: the schedule clause, the sections clause, and synchronization using the critical directive. These constructs are essential for managing parallel execution and ensuring the correct behavior of multi-threaded programs.

1. Schedule Clause

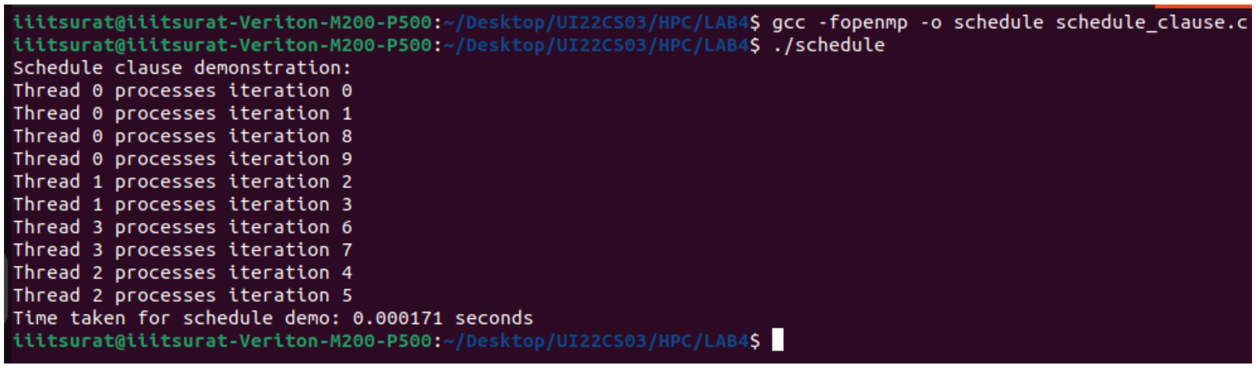
Theory: The schedule clause in OpenMP controls how loop iterations are distributed among threads. This distribution affects how workload is balanced and can impact the performance of parallel applications. Key types of scheduling strategies include:

- **Static Scheduling**: Iterations are divided into chunks of a fixed size and assigned to threads before the loop begins. For example, schedule(static, chunk_size) divides iterations into blocks of chunk_size.
- **Dynamic Scheduling**: Iterations are assigned to threads as they become available, which helps balance workloads when the iterations are of varying complexity. For example, schedule(dynamic, chunk_size) assigns chunks of iterations dynamically.

Program Demonstration: In the schedule_clauses program, static scheduling with a chunk size of 2 is used. The program measures the time taken to process a loop with 10 iterations across multiple threads, providing insights into how iterations are distributed and the effect of scheduling on performance.

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

#define NUM_ITERATIONS 10

void schedule_demo() {
        printf("Schedule clause demonstration:\n");

        double start_time = omp_get_wtime();

        #pragma omp parallel
        {
        #pragma omp for schedule(static, 2)
        for (int i = 0; i < NUM_ITERATIONS; i++) {
        printf("Thread %d processes iteration %d\n", omp_get_thread_num(), i);
        }
        }

        double end_time = omp_get_wtime();
        printf("Time taken for schedule demo: %f seconds\n", end_time - start_time);
}

int main() {
        omp_set_num_threads(4);
        schedule_demo();
        return 0;
}
```

Output:



```
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ gcc -fopenmp -o schedule schedule_clause.c
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ ./schedule
Schedule clause demonstration:
Thread 0 processes iteration 0
Thread 0 processes iteration 1
Thread 0 processes iteration 8
Thread 0 processes iteration 9
Thread 1 processes iteration 2
Thread 1 processes iteration 3
Thread 3 processes iteration 6
Thread 3 processes iteration 7
Thread 2 processes iteration 4
Thread 2 processes iteration 5
Time taken for schedule demo: 0.000171 seconds
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ 
```

2. Sections Clause

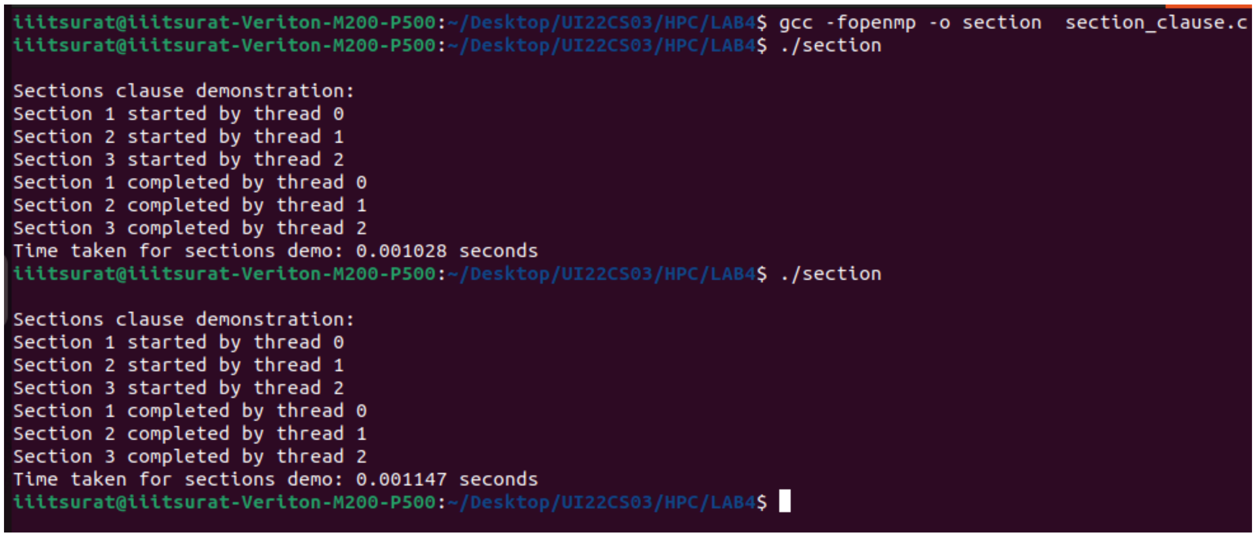
Theory: The section_clause clause allows different blocks of code to be executed in parallel by different threads. Each block is defined using the section directive within a sections block. This construct is useful for tasks that can be performed independently, such as different phases of a computation.

Program Demonstration: The sections_demo program illustrates how the sections construct can be used to execute different sections of code concurrently. Each section performs some simulated work, and the program measures the time taken to execute these sections in parallel.

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

void sections_demo() {
        printf("\nSections clause demonstration:\n");
```

```
	double start_time = omp_get_wtime();

	#pragma omp parallel
	{
	#pragma omp sections
	{
	#pragma omp section
	{
	printf("Section 1 started by thread %d\n", omp_get_thread_num());
	for (int i = 0; i < 1000000; ++i);
	printf("Section 1 completed by thread %d\n", omp_get_thread_num());
	}

	#pragma omp section
	{
	printf("Section 2 started by thread %d\n", omp_get_thread_num());
	for (int i = 0; i < 1000000; ++i);
	printf("Section 2 completed by thread %d\n", omp_get_thread_num());
	}

	#pragma omp section
	{
	printf("Section 3 started by thread %d\n", omp_get_thread_num());
	for (int i = 0; i < 1000000; ++i);
	printf("Section 3 completed by thread %d\n", omp_get_thread_num());
	}
	}
	}

	double end_time = omp_get_wtime();
	printf("Time taken for sections demo: %f seconds\n", end_time - start_time);
}

int main() {
	omp_set_num_threads(3);
	sections_demo();
	return 0;
}
```

Output:



```
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ gcc -fopenmp -o section  section_clause.c
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ ./section

Sections clause demonstration:
Section 1 started by thread 0
Section 2 started by thread 1
Section 3 started by thread 2
Section 1 completed by thread 0
Section 2 completed by thread 1
Section 3 completed by thread 2
Time taken for sections demo: 0.001028 seconds
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ ./section

Sections clause demonstration:
Section 1 started by thread 0
Section 2 started by thread 1
Section 3 started by thread 2
Section 1 completed by thread 0
Section 2 completed by thread 1
Section 3 completed by thread 2
Time taken for sections demo: 0.001147 seconds
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$
```

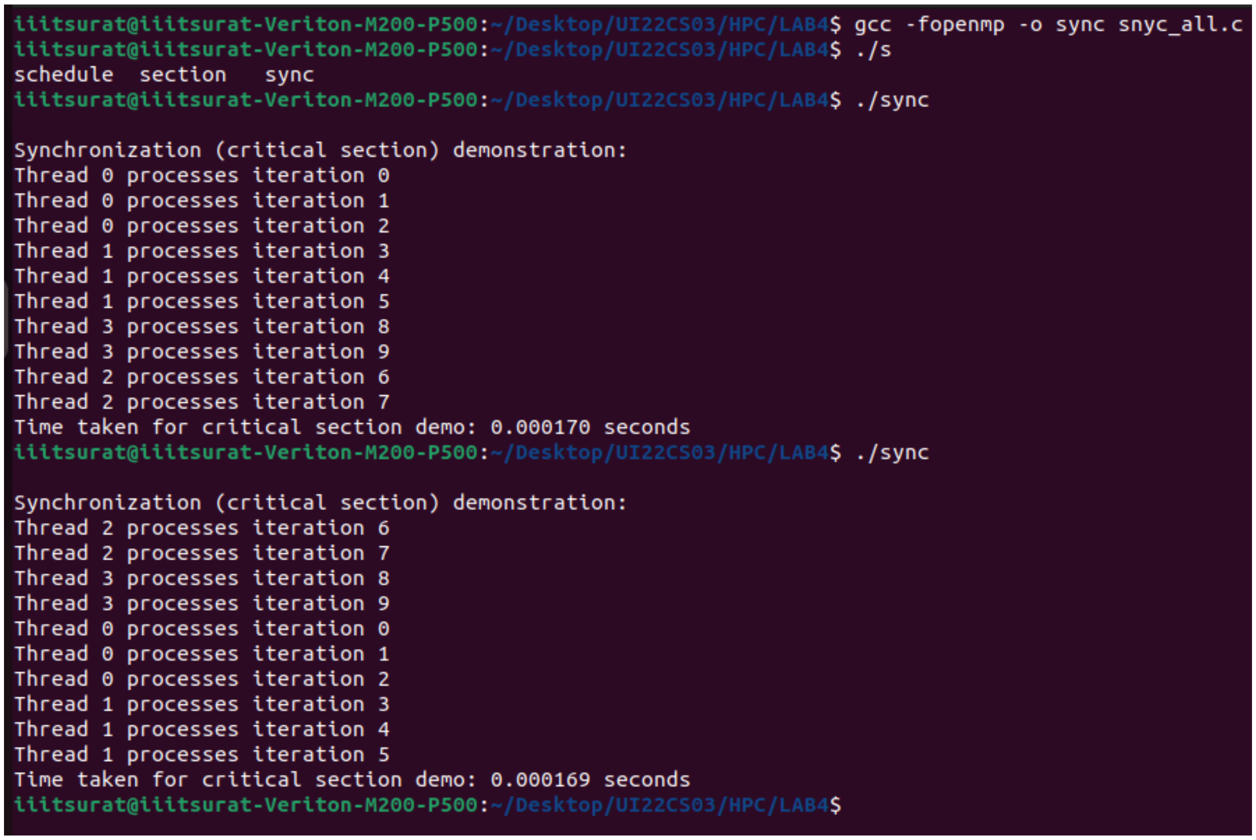
3. Synchronization with Critical Section

Theory: The critical directive is used to ensure that a section of code is executed by only one thread at a time. This is crucial when threads need to access shared resources to prevent data races and ensure consistency. The critical section protects code blocks that update shared variables or perform operations that must be executed atomically.

Program Demonstration: In the sync program, each thread processes iterations within a loop, and the critical directive ensures that only one thread prints its progress at a time. This synchronization prevents concurrent access issues and measures the time taken to complete the loop.

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

#define NUM_ITERATIONS 10

void critical_demo() {
        printf("\nSynchronization (critical section) demonstration:\n");

        double start_time = omp_get_wtime();

        #pragma omp parallel
        {
        #pragma omp for
        for (int i = 0; i < NUM_ITERATIONS; i++) {
        #pragma omp critical
        {
        printf("Thread %d processes iteration %d\n", omp_get_thread_num(), i);
        }
        }
        }

        double end_time = omp_get_wtime();
        printf("Time taken for critical section demo: %f seconds\n", end_time - start_time);
}

int main() {
        omp_set_num_threads(4);
        critical_demo();
        return 0;
}
```

Output:



```
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ gcc -fopenmp -o sync snyc_all.c
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ ./s
schedule  section   sync
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ ./sync

Synchronization (critical section) demonstration:
Thread 0 processes iteration 0
Thread 0 processes iteration 1
Thread 0 processes iteration 2
Thread 1 processes iteration 3
Thread 1 processes iteration 4
Thread 1 processes iteration 5
Thread 3 processes iteration 8
Thread 3 processes iteration 9
Thread 2 processes iteration 6
Thread 2 processes iteration 7
Time taken for critical section demo: 0.000170 seconds
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ ./sync

Synchronization (critical section) demonstration:
Thread 2 processes iteration 6
Thread 2 processes iteration 7
Thread 3 processes iteration 8
Thread 3 processes iteration 9
Thread 0 processes iteration 0
Thread 0 processes iteration 1
Thread 0 processes iteration 2
Thread 1 processes iteration 3
Thread 1 processes iteration 4
Thread 1 processes iteration 5
Time taken for critical section demo: 0.000169 seconds
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$
```

Combined Program Overview

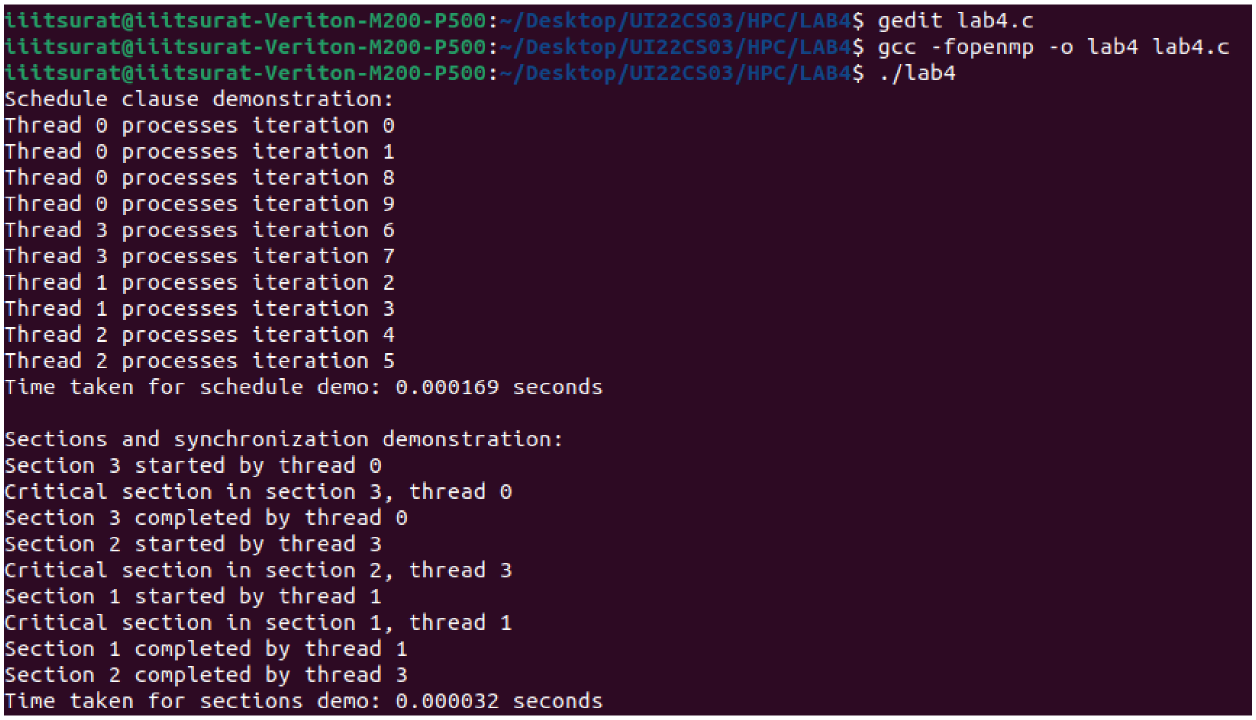
The combined OpenMP program integrates the following constructs into a single execution flow:

1. **Schedule Clause**: Demonstrates how loop iterations can be distributed among threads using different scheduling strategies.
2. **Sections Clause**: Illustrates how different code sections can be executed in parallel.
3. **Synchronization**: Uses the critical directive to ensure thread-safe access to shared resources.

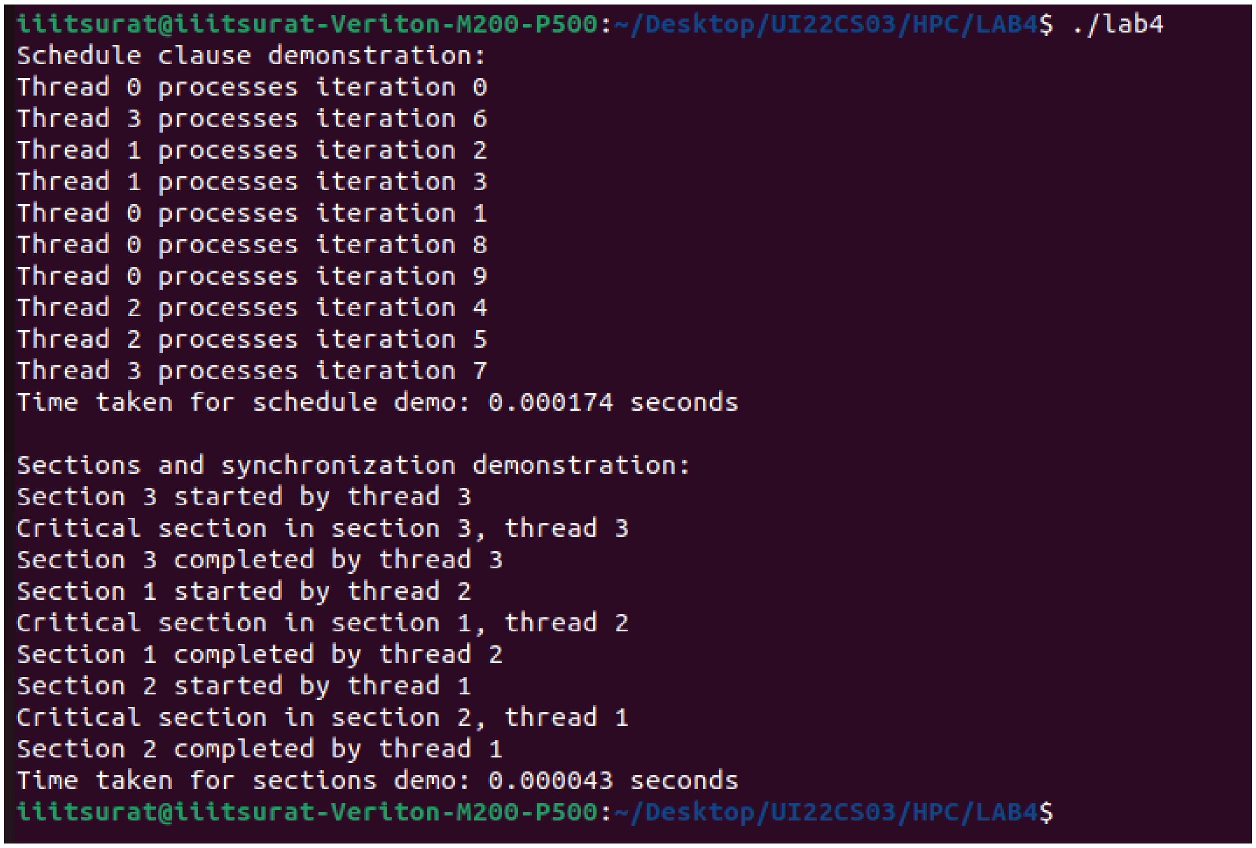
Combined Code:-

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

#define NUM_ITERATIONS 10
#define NUM_SECTIONS 3

void schedule_demo() {
        printf("Schedule clause demonstration:\n");

        double start_time = omp_get_wtime();

        #pragma omp parallel
        {
        #pragma omp for schedule(static, 2)
        for (int i = 0; i < NUM_ITERATIONS; i++) {
        printf("Thread %d processes iteration %d\n", omp_get_thread_num(), i);
        }
        }

        double end_time = omp_get_wtime();
        printf("Time taken for schedule demo: %f seconds\n", end_time - start_time);
}

void sections_demo() {
        printf("\nSections and synchronization demonstration:\n");

        double start_time = omp_get_wtime();

        #pragma omp parallel
        {
        #pragma omp sections
        {
        #pragma omp section
        {
        printf("Section 1 started by thread %d\n", omp_get_thread_num());
        #pragma omp critical
        {
                printf("Critical section in section 1, thread %d\n", omp_get_thread_num());
        }
        printf("Section 1 completed by thread %d\n", omp_get_thread_num());
        }

        #pragma omp section
        {
        printf("Section 2 started by thread %d\n", omp_get_thread_num());
        #pragma omp critical
        {
                printf("Critical section in section 2, thread %d\n", omp_get_thread_num());
        }
        printf("Section 2 completed by thread %d\n", omp_get_thread_num());
        }

        #pragma omp section
        {
        printf("Section 3 started by thread %d\n", omp_get_thread_num());
        #pragma omp critical
        {
                printf("Critical section in section 3, thread %d\n", omp_get_thread_num());
        }
        printf("Section 3 completed by thread %d\n", omp_get_thread_num());
        }
        }
        }

        double end_time = omp_get_wtime();
        printf("Time taken for sections demo: %f seconds\n", end_time - start_time);
}

int main() {
        omp_set_num_threads(4);

        schedule_demo();
        sections_demo();

        return 0;
}
```

Output:



```
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ gedit lab4.c
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ gcc -fopenmp -o lab4 lab4.c
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ ./lab4
Schedule clause demonstration:
Thread 0 processes iteration 0
Thread 0 processes iteration 1
Thread 0 processes iteration 8
Thread 0 processes iteration 9
Thread 3 processes iteration 6
Thread 3 processes iteration 7
Thread 1 processes iteration 2
Thread 1 processes iteration 3
Thread 2 processes iteration 4
Thread 2 processes iteration 5
Time taken for schedule demo: 0.000169 seconds

Sections and synchronization demonstration:
Section 3 started by thread 0
Critical section in section 3, thread 0
Section 3 completed by thread 0
Section 2 started by thread 3
Critical section in section 2, thread 3
Section 1 started by thread 1
Critical section in section 1, thread 1
Section 1 completed by thread 1
Section 2 completed by thread 3
Time taken for sections demo: 0.000032 seconds
```



```
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$ ./lab4
Schedule clause demonstration:
Thread 0 processes iteration 0
Thread 3 processes iteration 6
Thread 1 processes iteration 2
Thread 1 processes iteration 3
Thread 0 processes iteration 1
Thread 0 processes iteration 8
Thread 0 processes iteration 9
Thread 2 processes iteration 4
Thread 2 processes iteration 5
Thread 3 processes iteration 7
Time taken for schedule demo: 0.000174 seconds

Sections and synchronization demonstration:
Section 3 started by thread 3
Critical section in section 3, thread 3
Section 3 completed by thread 3
Section 1 started by thread 2
Critical section in section 1, thread 2
Section 1 completed by thread 2
Section 2 started by thread 1
Critical section in section 2, thread 1
Section 2 completed by thread 1
Time taken for sections demo: 0.000043 seconds
iiitsurat@iiitsurat-Veriton-M200-P500:~/Desktop/UI22CS03/HPC/LAB4$
```

Conclusion: In this assignment, I explored how OpenMP handles parallel programming using three main features: the schedule clause, sections, and critical sections. I tested each feature separately to see how they work and then combined them in one program. The schedule clause helped manage how loop iterations are distributed among threads, while the sections clause allowed different parts of the code to run in parallel. The critical section ensured that only one thread at a time could access shared resources. By measuring execution times and observing the outputs, I was able to understand how these features affect performance and how they help in writing efficient, thread-safe parallel programs.