Indian Institute of Information Technology Surat

Lab Report on
High Performance Computing (CS 602) Practical

Submitted by

[RAHUL KUMAR SINGH] (UI21CS44)

Course Faculty

Dr. Sachin D. Patil

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

Jan-2024

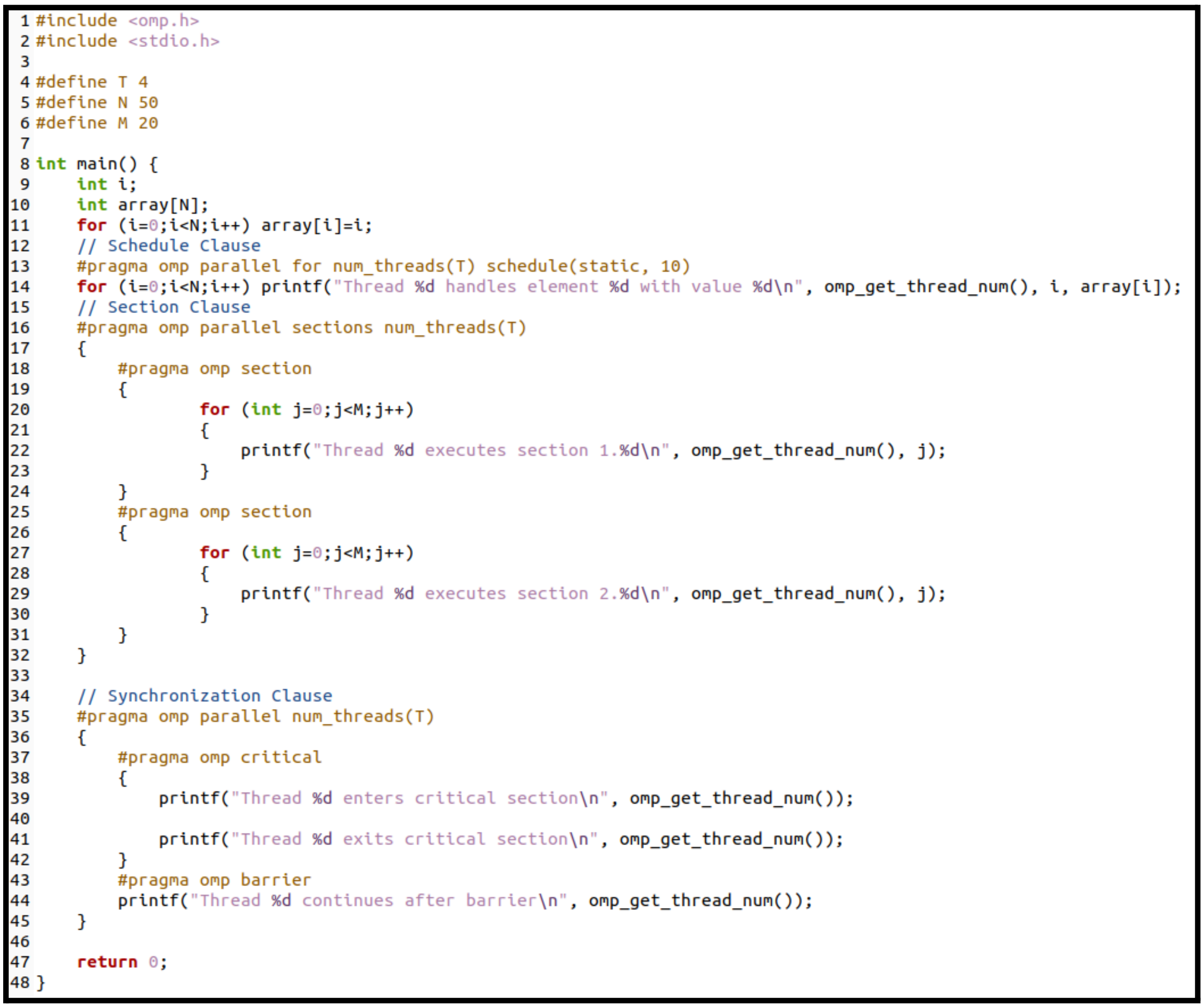
Lab No: 4

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

Description:

- Schedule Clause: Demonstrates static scheduling with a chunk size of 10 where each thread processes a chunk of the array.
- Section Clause: Defines parallel sections with designated code blocks where threads execute distinct sections concurrently.
- Synchronization Clause: Utilizes the \`critical\` directive to create a critical section and includes a \`barrier\` to synchronize threads, ensuring all reach a designated point before continuing.

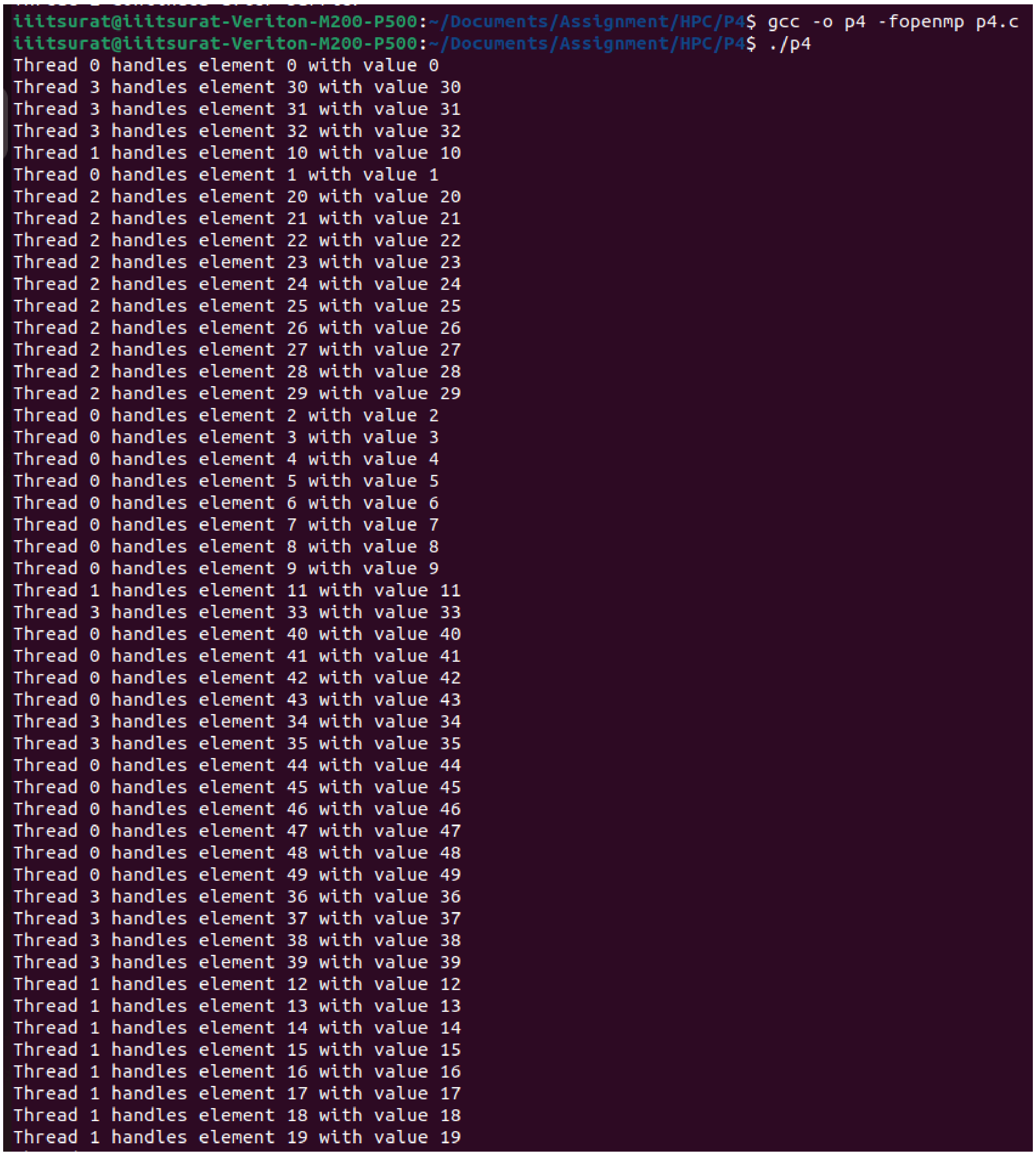
Source Code:



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

#define T 4
#define N 50
#define M 20

int main() {
    int i;
    int array[N];
    for (i=0;i<N;i++) array[i]=i;
    // Schedule Clause
    #pragma omp parallel for num_threads(T) schedule(static, 10)
    for (i=0;i<N;i++) printf("Thread %d handles element %d with value %d\n", omp_get_thread_num(), i, array[i]);
    // Section Clause
    #pragma omp parallel sections num_threads(T)
    {
        #pragma omp section
        {
                for (int j=0;j<M;j++)
                {
                    printf("Thread %d executes section 1.%d\n", omp_get_thread_num(), j);
                }
        }
        #pragma omp section
        {
                for (int j=0;j<M;j++)
                {
                    printf("Thread %d executes section 2.%d\n", omp_get_thread_num(), j);
                }
        }
    }

    // Synchronization Clause
    #pragma omp parallel num_threads(T)
    {
        #pragma omp critical
        {
            printf("Thread %d enters critical section\n", omp_get_thread_num());

            printf("Thread %d exits critical section\n", omp_get_thread_num());
        }
        #pragma omp barrier
        printf("Thread %d continues after barrier\n", omp_get_thread_num());
    }

    return 0;
}
```

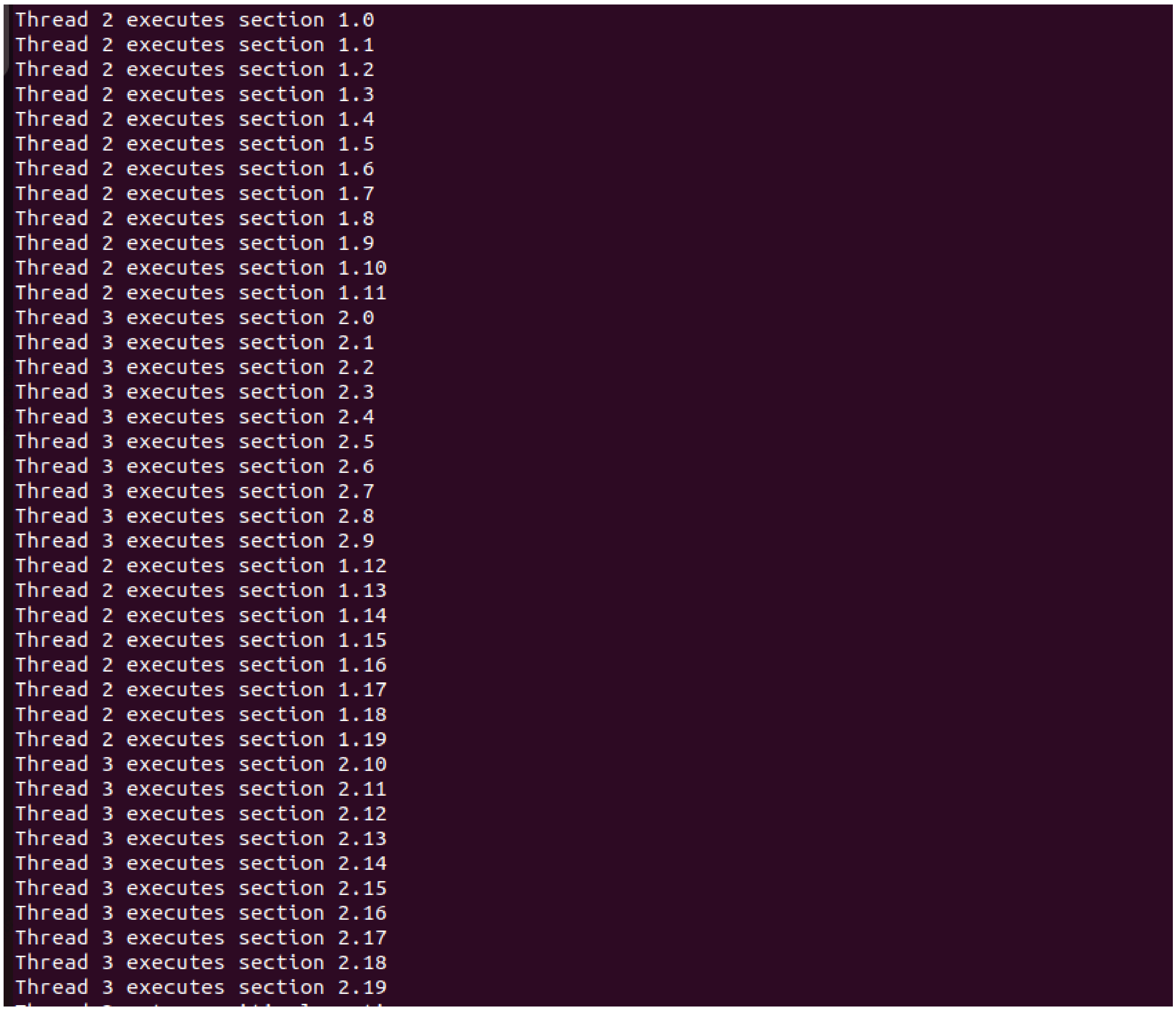
Output:

Schedule Clause:



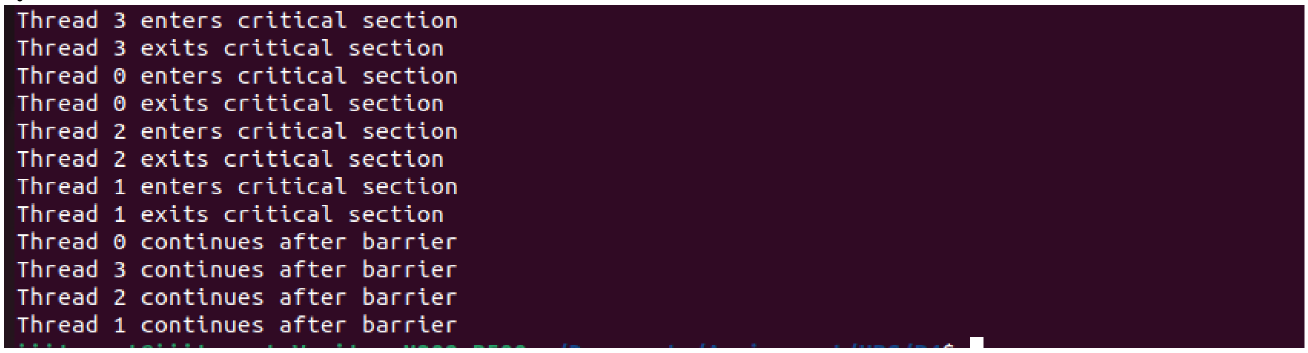
```
iiitsurat@iiitsurat-Veriton-M200-P500:~/Documents/Assignment/HPC/P4$ gcc -o p4 -fopenmp p4.c
iiitsurat@iiitsurat-Veriton-M200-P500:~/Documents/Assignment/HPC/P4$ ./p4
Thread 0 handles element 0 with value 0
Thread 3 handles element 30 with value 30
Thread 3 handles element 31 with value 31
Thread 3 handles element 32 with value 32
Thread 1 handles element 10 with value 10
Thread 0 handles element 1 with value 1
Thread 2 handles element 20 with value 20
Thread 2 handles element 21 with value 21
Thread 2 handles element 22 with value 22
Thread 2 handles element 23 with value 23
Thread 2 handles element 24 with value 24
Thread 2 handles element 25 with value 25
Thread 2 handles element 26 with value 26
Thread 2 handles element 27 with value 27
Thread 2 handles element 28 with value 28
Thread 2 handles element 29 with value 29
Thread 0 handles element 2 with value 2
Thread 0 handles element 3 with value 3
Thread 0 handles element 4 with value 4
Thread 0 handles element 5 with value 5
Thread 0 handles element 6 with value 6
Thread 0 handles element 7 with value 7
Thread 0 handles element 8 with value 8
Thread 0 handles element 9 with value 9
Thread 1 handles element 11 with value 11
Thread 3 handles element 33 with value 33
Thread 0 handles element 40 with value 40
Thread 0 handles element 41 with value 41
Thread 0 handles element 42 with value 42
Thread 0 handles element 43 with value 43
Thread 3 handles element 34 with value 34
Thread 3 handles element 35 with value 35
Thread 0 handles element 44 with value 44
Thread 0 handles element 45 with value 45
Thread 0 handles element 46 with value 46
Thread 0 handles element 47 with value 47
Thread 0 handles element 48 with value 48
Thread 0 handles element 49 with value 49
Thread 3 handles element 36 with value 36
Thread 3 handles element 37 with value 37
Thread 3 handles element 38 with value 38
Thread 3 handles element 39 with value 39
Thread 1 handles element 12 with value 12
Thread 1 handles element 13 with value 13
Thread 1 handles element 14 with value 14
Thread 1 handles element 15 with value 15
Thread 1 handles element 16 with value 16
Thread 1 handles element 17 with value 17
Thread 1 handles element 18 with value 18
Thread 1 handles element 19 with value 19
```

Section Clause:



```
Thread 2 executes section 1.0
Thread 2 executes section 1.1
Thread 2 executes section 1.2
Thread 2 executes section 1.3
Thread 2 executes section 1.4
Thread 2 executes section 1.5
Thread 2 executes section 1.6
Thread 2 executes section 1.7
Thread 2 executes section 1.8
Thread 2 executes section 1.9
Thread 2 executes section 1.10
Thread 2 executes section 1.11
Thread 3 executes section 2.0
Thread 3 executes section 2.1
Thread 3 executes section 2.2
Thread 3 executes section 2.3
Thread 3 executes section 2.4
Thread 3 executes section 2.5
Thread 3 executes section 2.6
Thread 3 executes section 2.7
Thread 3 executes section 2.8
Thread 3 executes section 2.9
Thread 2 executes section 1.12
Thread 2 executes section 1.13
Thread 2 executes section 1.14
Thread 2 executes section 1.15
Thread 2 executes section 1.16
Thread 2 executes section 1.17
Thread 2 executes section 1.18
Thread 2 executes section 1.19
Thread 3 executes section 2.10
Thread 3 executes section 2.11
Thread 3 executes section 2.12
Thread 3 executes section 2.13
Thread 3 executes section 2.14
Thread 3 executes section 2.15
Thread 3 executes section 2.16
Thread 3 executes section 2.17
Thread 3 executes section 2.18
Thread 3 executes section 2.19
```

Synchronization Clause:



```
Thread 3 enters critical section
Thread 3 exits critical section
Thread 0 enters critical section
Thread 0 exits critical section
Thread 2 enters critical section
Thread 2 exits critical section
Thread 1 enters critical section
Thread 1 exits critical section
Thread 0 continues after barrier
Thread 3 continues after barrier
Thread 2 continues after barrier
Thread 1 continues after barrier
```

Conclusion:

- Schedule Clause: Allows for efficient workload distribution among threads by controlling how iterations of a loop are assigned to threads.
- Section Clause: Facilitates parallel execution of distinct code sections.
- Synchronization Clause: Ensures orderly execution and prevents race conditions in critical sections.
- Efficiently utilizing clauses like \`schedule\`, \`section\`, and synchronization techniques improves code parallelism and resource management.