Name : Waqar Ahmed

Roll No: 20P-0750

Assignment #01

Task 1

1. Provide a list of run-time routines that are used in OpenMP

2. Why aren't you seing the Hello World output thread sequence as 0, 1, 2, 3 etc. Why are they disordered?

3. What happens to the thread_id if you change its scope to before the pragma?

4. Convert the code to serial code.

1: Some common OpenMP run-time routines:

omp_get_thread_num(): Returns the thread number of the calling thread.

omp_get_num_threads(): Returns the total number of threads in the current team.

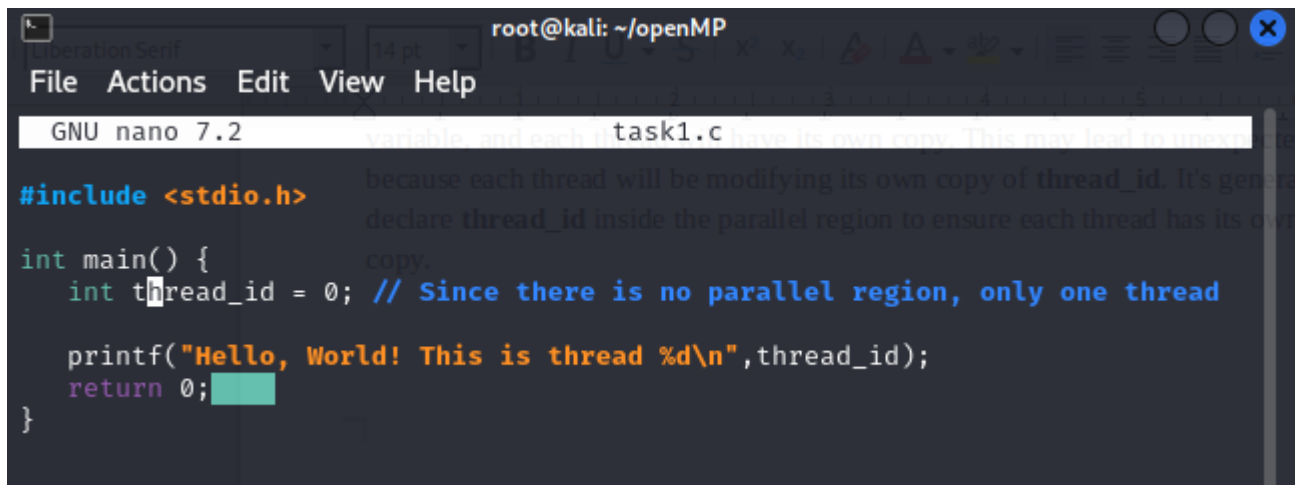**omp_set_num_threads():** Sets the number of threads to be used for subsequent parallel regions.

omp_get_max_threads(): Returns the maximum number of threads that could be used.

omp_get_num_procs(): Returns the number of processors available.

2: The order of output from the parallel region is not guaranteed to be in sequence (0, 1, 2, 3, etc.) because the order in which threads execute is not specified in OpenMP. The scheduling of threads is managed by the operating system and the OpenMP runtime. Each thread executes independently, and their order of execution is not deterministic.

3: If we change the scope of **thread_id** to before the pragma, it becomes a shared variable, and each thread will have its own copy. This may lead to unexpected behavior because each thread will be modifying its own copy of **thread_id**. It's generally safer to declare **thread_id** inside the parallel region to ensure each thread has its own private copy.
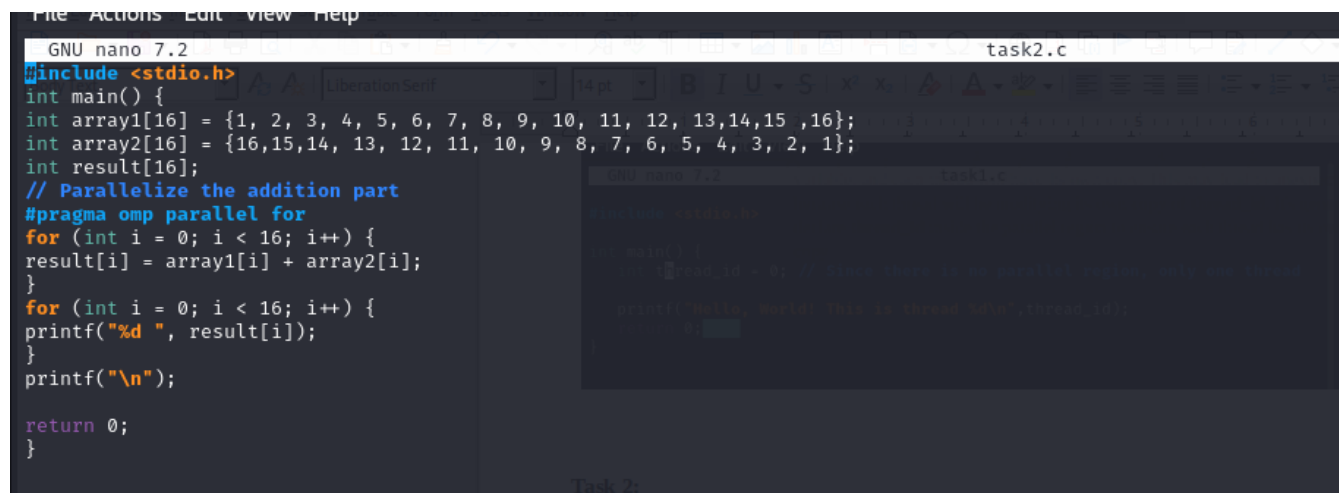
4: serial code

```
GNU nano 7.2                    task1.c

#include <stdio.h>

int main() {
    int thread_id = 0; // Since there is no parallel region, only one thread

    printf("Hello, World! This is thread %d\n",thread_id);
    return 0;
}
```

Task 2:

2. Convert it into Parallel, such that only the addition part is parallelized.

```
GNU nano 7.2                                                          task2.c
#include <stdio.h>
int main() {
int array1[16] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,14,15 ,16};
int array2[16] = {16,15,14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
int result[16];
// Parallelize the addition part
#pragma omp parallel for
for (int i = 0; i < 16; i++) {
result[i] = array1[i] + array2[i];
}
for (int i = 0; i < 16; i++) {
printf("%d ", result[i]);
}
printf("\n");

return 0;
}
```

3. The display loop at the end displays the result. Modify the code such that this is also parallel, but only thread of id 0 is able to display the entire loop. The others should not do anything. When making it parallel, make sure its the old threads and new threads are not created. What output do you see?

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

int main() {
    int array1[16] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};
    int array2[16] = {16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    int result[16];

    // Parallelize the addition part
    #pragma omp parallel for
    for (int i = 0; i < 16; i++) {
        result[i] = array1[i] + array2[i];
    }

    // Parallelize the display loop
    #pragma omp parallel
    {
        int thread_id = omp_get_thread_num();

        // Only thread of id 0 should display the entire loop
        if (thread_id == 0) {
            for (int i = 0; i < 16; i++) {
                printf("%d ", result[i]);
            }
            printf("\n");
        }
    }

    return 0;
}
```
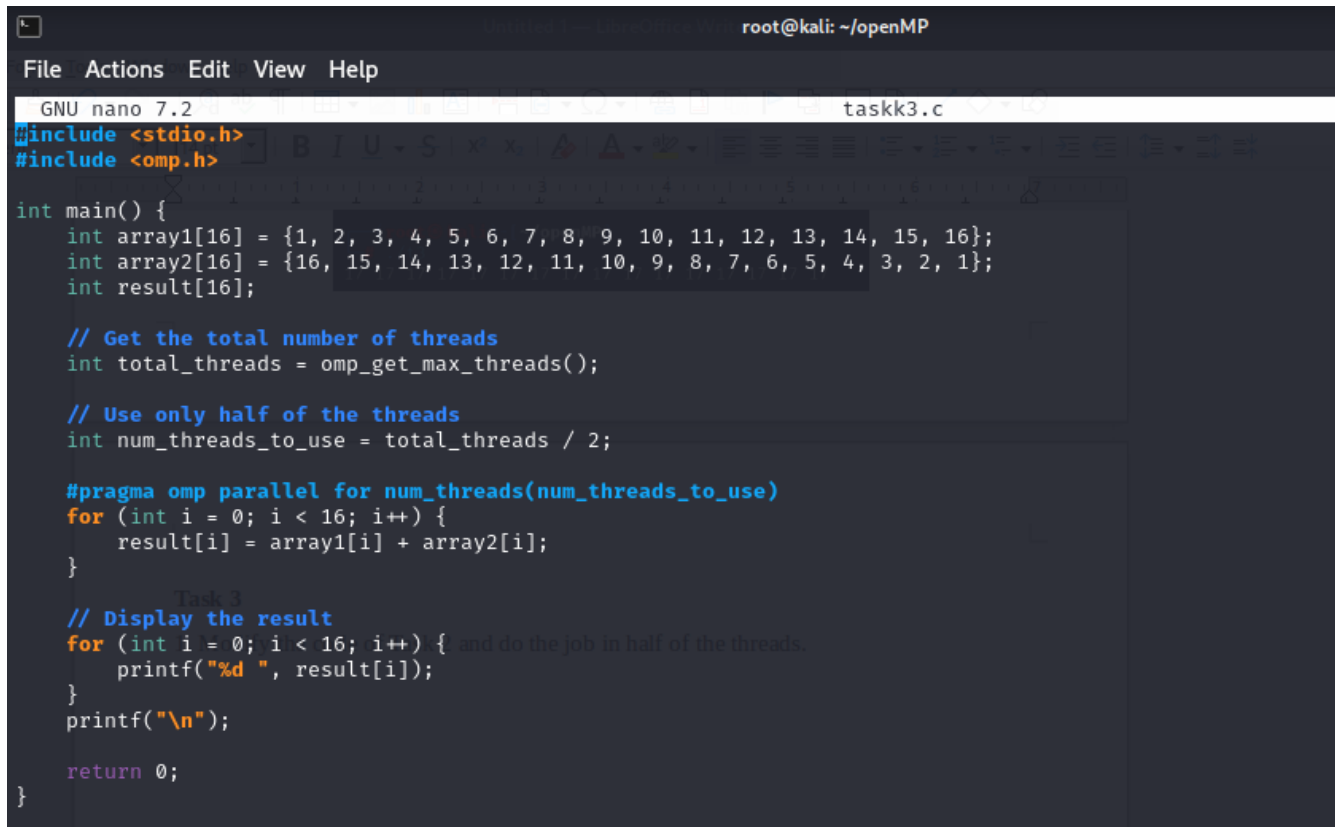
output:

```
┌──(root㉿kali)-[~/openMP]
└─# gcc -o t3 -fopenmp task2-b.c

┌──(root㉿kali)-[~/openMP]
└─# ./t3
17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
```

Task 3

1. Modify the code of Task 2 and do the job in half of the threads.

```
File   Actions   Edit   View   Help
  GNU nano 7.2                                                    taskk3.c
#include <stdio.h>
#include <omp.h>

int main() {
    int array1[16] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};
    int array2[16] = {16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    int result[16];

    // Get the total number of threads
    int total_threads = omp_get_max_threads();

    // Use only half of the threads
    int num_threads_to_use = total_threads / 2;

    #pragma omp parallel for num_threads(num_threads_to_use)
    for (int i = 0; i < 16; i++) {
        result[i] = array1[i] + array2[i];
    }

    // Display the result
    for (int i = 0; i < 16; i++) {
        printf("%d ", result[i]);
    }
    printf("\n");

    return 0;
}
```

Task 4

1. The following code adds the contents of the array array1 and array2.

#include <stdio.h>

int main() {

int array1[16] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, ..., 16};

int array2[16] = {16, ..., 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};

int result1 = 0, result2 = 0;

for (int i = 0; i < 16; i++) {

```c
result1 += array1[i];
}
if (result1 > 10) {
result2 = result1;
for (int i = 0; i < 16; i++) {
result2 += array2[i];
}
}
printf("%d\n", result2);
return 0;
}
```

2. Convert it into Parallel using 16 threads.

File Actions Edit View Help

```
GNU nano 7.2                                              t4.c
#include <stdio.h>
#include <omp.h>

int main() {
    int array1[16] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};
    int array2[16] = {16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    int result1 = 0, result2 = 0;

    #pragma omp parallel num_threads(16)
    {
        #pragma omp for reduction(+:result1)
        for (int i = 0; i < 16; i++) {
            result1 += array1[i];
        }

        #pragma omp sections
        {
            #pragma omp section
            {
                if (result1 > 10) {
                    #pragma omp parallel for reduction(+:result2)
                    for (int i = 0; i < 16; i++) {
                        result2 += array2[i];
                    }
                }
            }
        }
    }

    printf("%d\n", result2);
```
```
                                              [ Read 32 lines ]
^G Help          ^O Write Out    ^W Where Is     ^K Cut          ^T Execute      ^C Locati
```

3. Try removing the reduction() clause and add #pragma omp atomic just beore the +=. What is the effect on result? Explain.

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

int main() {
    int array1[16] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};
    int array2[16] = {16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    int result1 = 0, result2 = 0;

    #pragma omp parallel num_threads(16)
    {
        #pragma omp for
        for (int i = 0; i < 16; i++) {
            #pragma omp atomic
            result1 += array1[i];
        }

        #pragma omp sections
        {
            #pragma omp section
            {
                if (result1 > 10) {
                    #pragma omp parallel for
                    for (int i = 0; i < 16; i++) {
                        #pragma omp atomic
                        result2 += array2[i];
                    }
                }
            }
        }
    }
```

Effect of result: The **#pragma omp atomic** directive ensures that the specified operation is executed atomically, avoiding race conditions that may occur in parallel regions. However, using atomic operations can introduce contention, and in some cases, it might lead to decreased performance compared to using a reduction clause.

In this specific code, since the updates to result1 and result2 are performed atomically, the final result should still be correct. However, the performance characteristics may vary depending on the specifics of the system and workload.