

## U23AI052 LAB ASSIGNMENT- 1 SUBMISSION

**Deep Das**

## Exercise 1: Basic Thread Parallelism

**Objective:** Initialize the OpenMP environment and verify multithreaded execution.

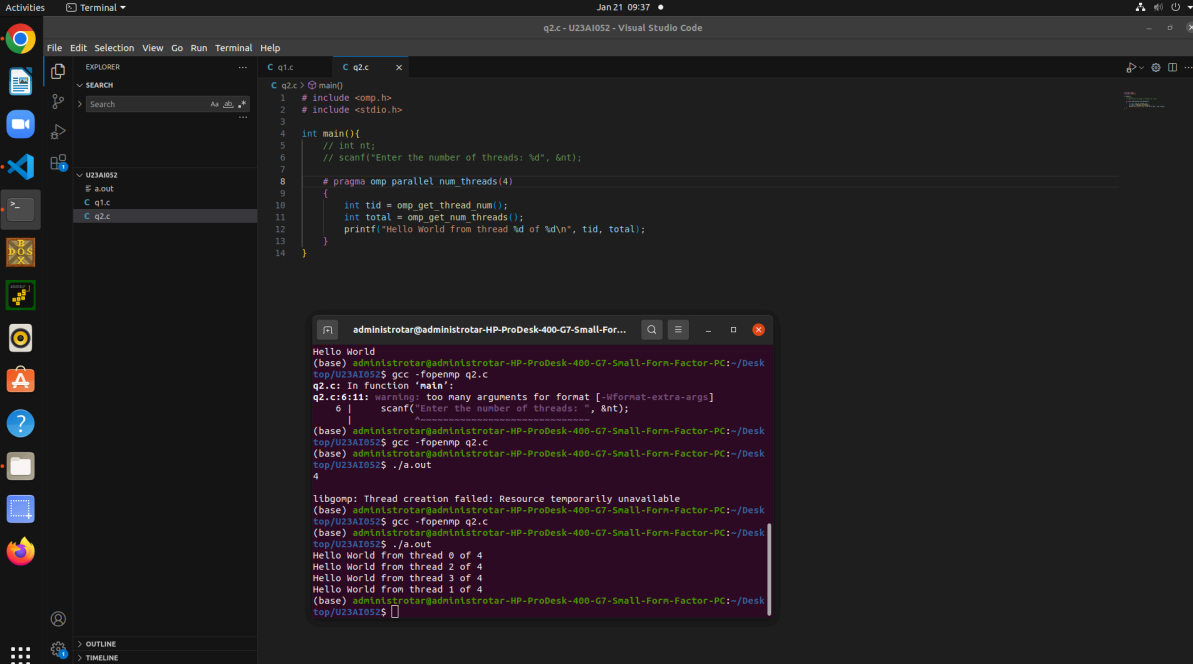
- **Task:** Write a C/C++ program that uses the `#pragma omp parallel` directive to print "Hello World."
- **Key Concept:** Understand the "Fork-Join" model where the master thread spawns a team of worker threads.

[illegible]

## Exercise 2: Runtime Configuration and Data Environment

**Objective:** Learn to pass parameters to the parallel region at runtime and use OpenMP clauses.

- **Task:** Write a program that:
  1. Prompts the user to enter the number of threads at runtime.
  2. Uses the `num_threads()` clause to set the thread size.
  3. Prints: "Hello World from Thread [ID] of [Total Threads]."



The screenshot shows a Visual Studio Code editor with a C program named `q2.c` and a terminal window displaying the program's execution. The C program is as follows:

```
1 #include <omp.h>
2 #include <stdio.h>
3
4 int main()
5 {
6     // int nt;
7     // scanf("Enter the number of threads: %d", &nt);
8     #pragma omp parallel num_threads(4)
9     {
10         int tid = omp_get_thread_num();
11         int total = omp_get_num_threads();
12         printf("Hello World from thread %d of %d\n", tid, total);
13     }
14 }
```

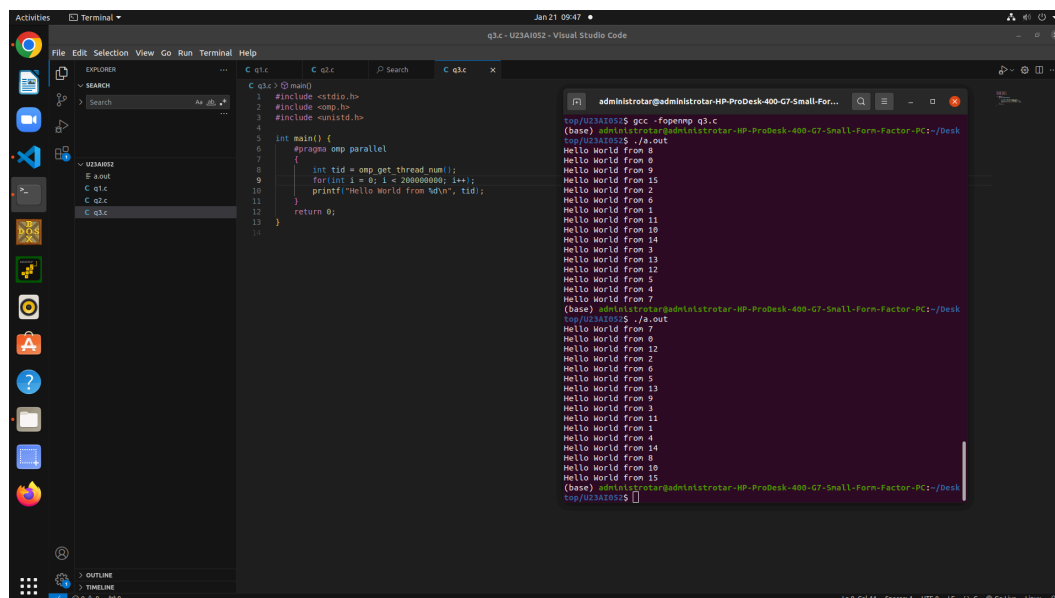
The terminal window shows the following output:

```
administrotar@administrotar-HP-ProDesk-400-G7-Small-For...
Hello World
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052$ gcc -fopenmp q2.c
q2.c: In function 'main':
q2.c:6:11: warning: too many arguments for format [-Wformat-extra-args]
6 |     scanf("Enter the number of threads: ", &nt);
  |     ~~~~~
  |
  |
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052$ gcc -fopenmp q2.c
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052$ ./a.out
4
libgomp: Thread creation failed: Resource temporarily unavailable
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052$ gcc -fopenmp q2.c
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052$ ./a.out
Hello World from thread 0 of 4
Hello World from thread 2 of 4
Hello World from thread 3 of 4
Hello World from thread 1 of 4
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052$
```

### Exercise 3: Analyzing Race Conditions and Solution

**Objective:** Observe the non-deterministic nature of parallel execution and the impact of shared resources.

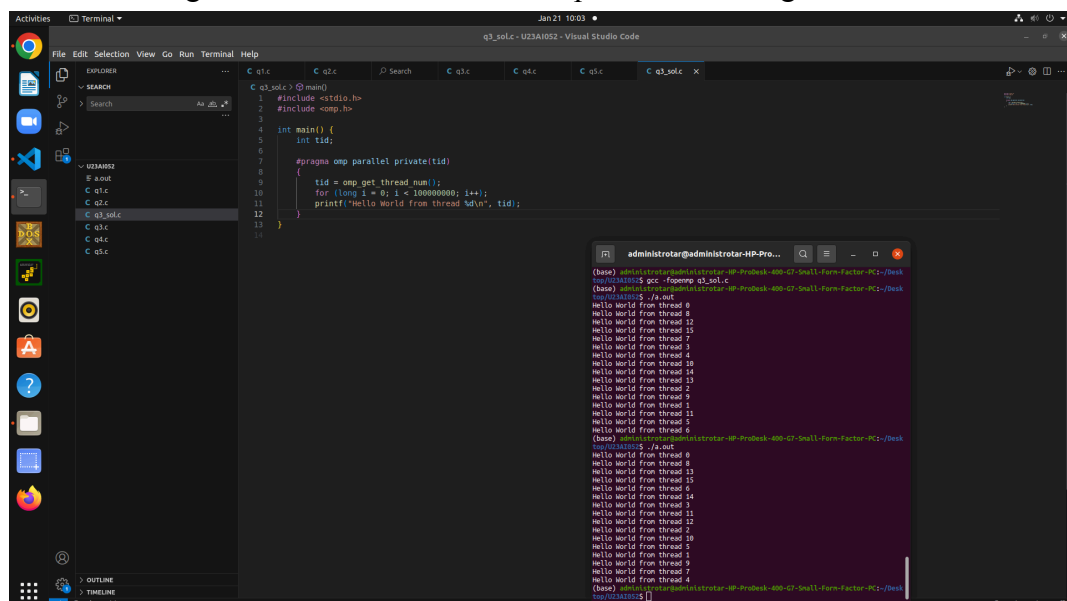
- **Task:** Write an OpenMP program to print "Hello World" do not ma
- **Requirement:** Introduce a manual delay (using a "wait" condition or a heavy dummy loop) inside the parallel block to force threads to overlap.
- **Observation:** Document how the output becomes scrambled (interleaved) due to multiple threads accessing the standard output simultaneously, demonstrating a **Race Condition**.
- **Solution:** Use the private() clause to ensure each thread has its own copy of the thread ID variable.



```
1 #include <stdio.h>
2 #include <omp.h>
3 #include <unistd.h>
4
5 int main() {
6     #pragma omp parallel
7     {
8         int tid = omp_get_thread_num();
9         for(int i = 0; i < 200000000; i++);
10        printf("hello world from %d\n", tid);
11    }
12    return 0;
13 }
```

```
top/U23A1052$ gcc -fopenmp q3.c
(base) administrator@administrator-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23A1052$ ./a.out
Hello world from 9
Hello world from 9
Hello world from 15
Hello world from 2
Hello world from 6
Hello world from 11
Hello world from 10
Hello world from 14
Hello world from 3
Hello world from 13
Hello world from 12
Hello world from 5
Hello world from 4
Hello world from 7
Hello world from 7
Hello world from 7
Hello world from 12
Hello world from 6
Hello world from 2
Hello world from 6
Hello world from 5
Hello world from 13
Hello world from 9
Hello world from 5
Hello world from 11
Hello world from 1
Hello world from 4
Hello world from 14
Hello world from 8
Hello world from 10
Hello world from 15
(base) administrator@administrator-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23A1052$
```

after making the thread id as private we get the same order



```
1 #include <stdio.h>
2 #include <omp.h>
3 #include <unistd.h>
4
5 int main() {
6     int tid;
7     #pragma omp parallel private(tid)
8     {
9         tid = omp_get_thread_num();
10        for (long i = 0; i < 100000000; i++);
11        printf("hello world from thread %d\n", tid);
12    }
13 }
```

```
(base) administrator@administrator-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23A1052$ gcc -fopenmp q3_sol.c
(base) administrator@administrator-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23A1052$ ./a.out
Hello world from thread 0
Hello world from thread 1
Hello world from thread 2
Hello world from thread 3
Hello world from thread 4
Hello world from thread 5
Hello world from thread 6
Hello world from thread 7
Hello world from thread 8
Hello world from thread 9
Hello world from thread 10
Hello world from thread 11
Hello world from thread 12
Hello world from thread 13
Hello world from thread 14
Hello world from thread 15
(base) administrator@administrator-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23A1052$
```

## Exercise 4: Large-Scale Vector Addition (Data Parallelism)

**Objective:** Implement data decomposition on large arrays to measure parallel efficiency.

- Task: 1. Create three 1D arrays of size 1,000,000.
2. Sequentially: Initialize two arrays with random or incremental values.
  3. Parallely: Use `#pragma omp parallel for` to calculate the sum:  $C[i] = A[i] + B[i]$ .
- **Requirement:** Compare the performance of the parallel loop against a standard sequential loop.

The screenshot shows the Visual Studio Code editor with a C program for vector addition. The program defines a large array size `N = 1000000` and implements both sequential and parallel versions of the addition loop. The sequential version uses `omp_get_wtime()` to measure time. The parallel version uses `#pragma omp parallel for`. The output window shows the results of running the program, comparing sequential and parallel execution times.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4
5 #define N 1000000
6
7 int main() {
8     double *A, *B, *C;
9     double start, end;
10    int i;
11
12    A = (double *)malloc(N * sizeof(double));
13    B = (double *)malloc(N * sizeof(double));
14    C = (double *)malloc(N * sizeof(double));
15
16    for (i = 0; i < N; i++) {
17        A[i] = 1 * i;
18        B[i] = 1 * 2 * i;
19    }
20
21    start = omp_get_wtime();
22    for (i = 0; i < N; i++) {
23        C[i] = A[i] + B[i];
24    }
25    end = omp_get_wtime();
26    printf("Sequential Time: %f seconds\n", end - start);
27
28    start = omp_get_wtime();
29    #pragma omp parallel for
30    for (i = 0; i < N; i++) {
31        C[i] = A[i] + B[i];
32    }
33    end = omp_get_wtime();
34    printf("Parallel Time: %f seconds\n", end - start);
35
36    free(A);
37    free(B);
38    free(C);
39}
```

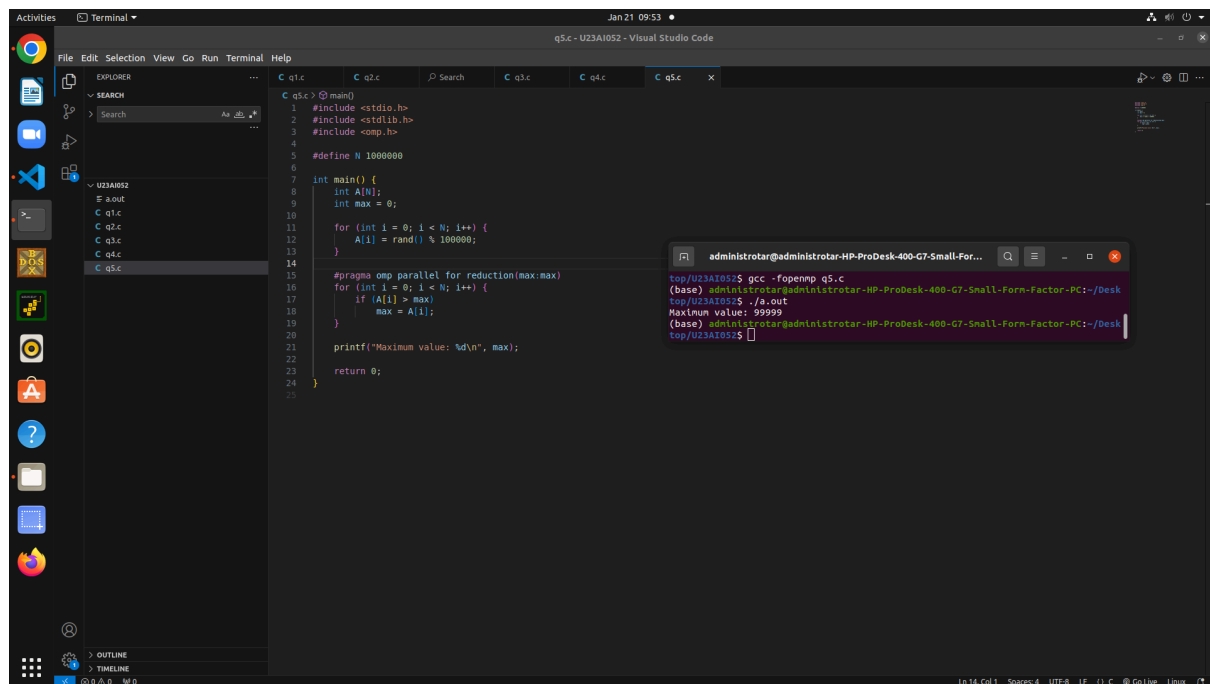
Output:

```
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052$ gcc -fopenmp q4.c
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052$ ./a.out
Sequential Time: 0.003557 seconds
Parallel Time: 0.001698 seconds
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desk
top/U23AI052$
```

## Exercise 5: Parallel Reduction for Maximum Value

**Objective:** Use OpenMP synchronization or reduction clauses to find maximum value in a dataset.

- Task: 1. Insert values in a 1D array of size 1,000,000 with random integers sequentially.
- 2. Parallely: Find the maximum value in the array.
- **Requirement:** Students should implement this using the reduction(max: variable) clause to avoid manual locking while ensuring thread safety.



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <omp.h>
4
5 #define N 1000000
6
7 int main() {
8     int A[N];
9     int max = 0;
10
11     for (int i = 0; i < N; i++) {
12         A[i] = rand() % 100000;
13     }
14
15     #pragma omp parallel for reduction(max:max)
16     for (int i = 0; i < N; i++) {
17         if (A[i] > max)
18             max = A[i];
19     }
20
21     printf("Maximum value: %d\n", max);
22
23     return 0;
24 }
25
```

```
administrotar@administrotar-HP-ProDesk-400-G7-Small-For...
top/U23AI052$ gcc -fopenmp q5.c
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC1~/Desk
top/U23AI052$ ./a.out
Maximum value: 99999
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC1~/Desk
top/U23AI052$
```