

**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.

## 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 Linux desktop environment with Visual Studio Code open. The code editor displays two files: q1.c and q2.c. The terminal window below shows the execution of the q2.c program, which prompts for the number of threads and prints "Hello World" from each thread. The terminal output is as follows:

```
libomp: Thread creation failed: Resource temporarily unavailable
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desktop
top/U23AI05$ gcc -fopenmp q2.c
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desktop
top/U23AI05$ ./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:~/Desktop
top/U23AI05$
```

## 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 make it sequential.
- **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.

A screenshot of a Linux desktop environment showing a terminal window and a Visual Studio Code editor. The terminal window shows the command `top/234052\$ gcc -fopenmp q3.c` being run, followed by a long list of interleaved "Hello World from [tid]" messages. The VS Code editor shows the source code for `q3.c` which contains a simple OpenMP parallel loop that prints "Hello World" with the thread ID. The code is as follows:

```
#include <stdio.h>
#include <omp.h>
int main() {
    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        for(int i = 0; i < 200000000; i++)
            printf("Hello World from %d\n", tid);
    }
    return 0;
}
```

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

A screenshot of a Linux desktop environment showing a terminal window and a Visual Studio Code editor. The terminal window shows the command `top/234052\$ gcc -fopenmp q3\_sol.c` being run, followed by a list of sequential "Hello World from thread [tid]" messages. The VS Code editor shows the source code for `q3\_sol.c` which is identical to `q3.c` except for the addition of the `private(tid)` clause in the OpenMP parallel directive. The code is as follows:

```
#include <stdio.h>
#include <omp.h>
int main() {
    int tid;
    #pragma omp parallel private(tid)
    {
        tid = omp_get_thread_num();
        for (long i = 0; i < 100000000; i++)
            printf("Hello World from thread %d\n", tid);
    }
    return 0;
}
```

## 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. Parallelly: 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 a Visual Studio Code interface with the following details:

- File Explorer:** Shows files q1.c, q2.c, q3.c, and q4.c in the U23AI052 workspace.
- Terminal:** Displays the following code for q4.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define N 1000000
int main() {
    double *A, *B, *C;
    double start, end;
    int i;
    A = (double *)malloc(N * sizeof(double));
    B = (double *)malloc(N * sizeof(double));
    C = (double *)malloc(N * sizeof(double));
    for (i = 0; i < N; i++) {
        A[i] = i * 1.0;
        B[i] = i * 2.0;
    }
    start = omp_get_wtime();
    for (i = 0; i < N; i++) {
        C[i] = A[i] + B[i];
    }
    end = omp_get_wtime();
    printf("Sequential Time: %f seconds\n", end - start);
    start = omp_get_wtime();
    #pragma omp parallel for
    for (i = 0; i < N; i++) {
        C[i] = A[i] + B[i];
    }
    end = omp_get_wtime();
    printf("Parallel Time: %f seconds\n", end - start);
    free(A);
    free(B);
    free(C);
}
```

- Terminal Output:** The terminal shows the execution of the program and its results:

```
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desktop/U23AI052$ gcc -fopenmp q4.c
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desktop/U23AI052$ ./a.out
Sequential Time: 0.003557 seconds
Parallel Time: 0.001098 seconds
(base) administrotar@administrotar-HP-ProDesk-400-G7-Small-Form-Factor-PC:~/Desktop/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. Parallelly: 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.

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows files q1.c, q2.c, q3.c, q4.c, and q5.c in the U23AI052 workspace.
- Terminal:** Displays the command line output:

```
top@U23AI052:~$ gcc -fopenmp q5.c
(base) top@U23AI052:~/Desktop
top@U23AI052:~/Desktop$ ./a.out
Maximum value: 99999
(base) top@U23AI052:~/Desktop$
```
- Code Editor:** Shows the content of q5.c: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() % 1000000;
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 }