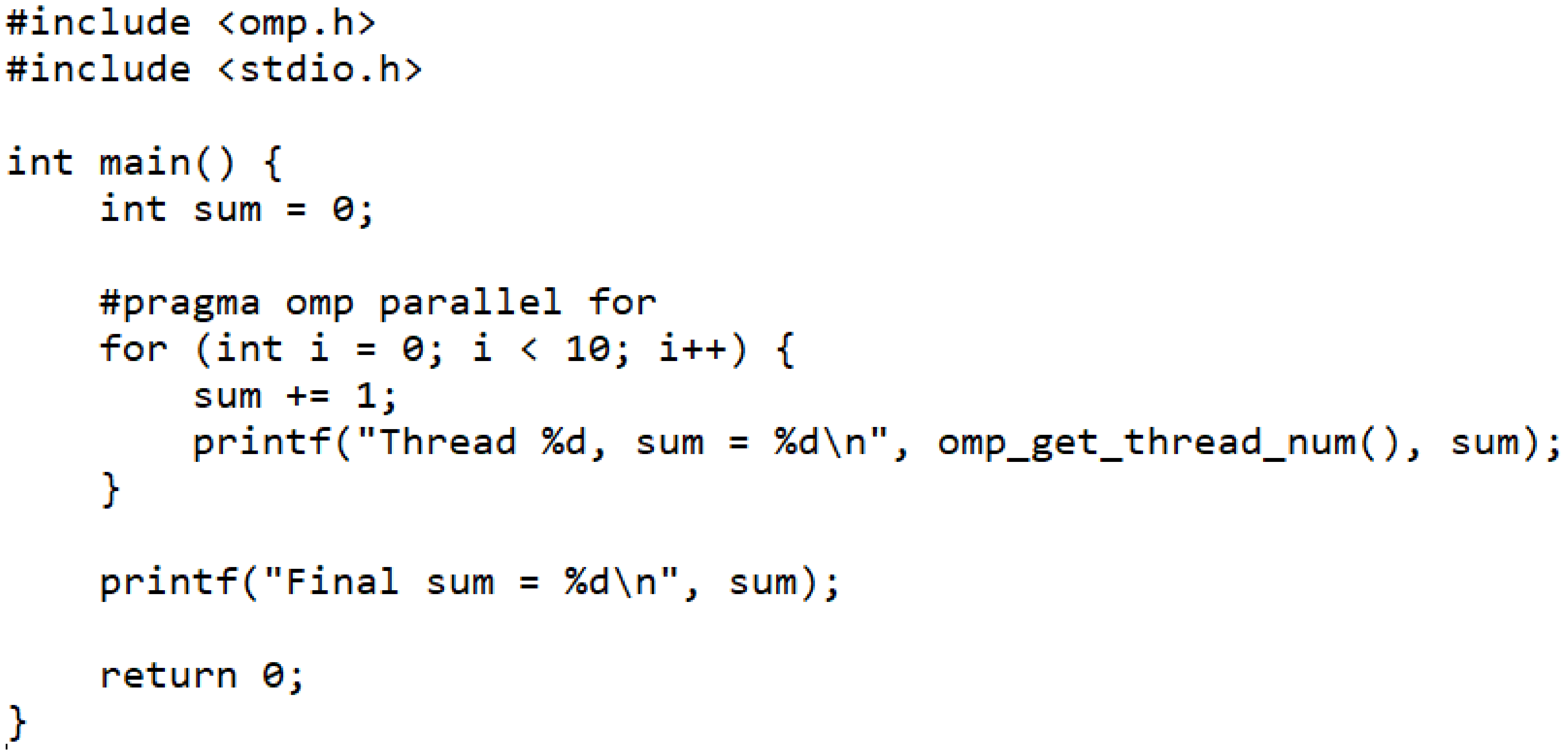
CS4172L: Parallel and Distributed Computing Lab

Lab 3 - Parallel Loop with Shared Variable and Local Variable

Parallel Loop with Shared Variable

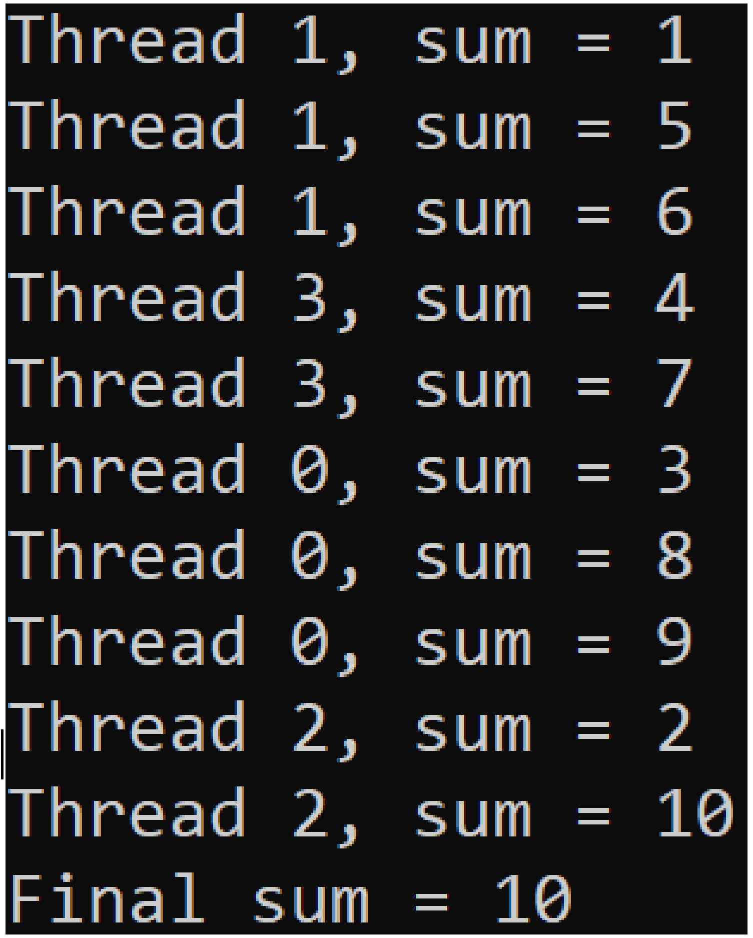


```c
#include <omp.h>
#include <stdio.h>

int main() {
    int sum = 0;

    #pragma omp parallel for
    for (int i = 0; i < 10; i++) {
        sum += 1;
        printf("Thread %d, sum = %d\n", omp_get_thread_num(), sum);
    }

    printf("Final sum = %d\n", sum);

    return 0;
}
```

Race Condition

- In this program, the variable sum is shared among all threads because it is declared outside the parallel region. Each thread updates the sum variable independently within the loop.
- However, since multiple threads are updating sum concurrently, **race conditions occur**, and the final value of sum may vary between different runs of the program.

Output

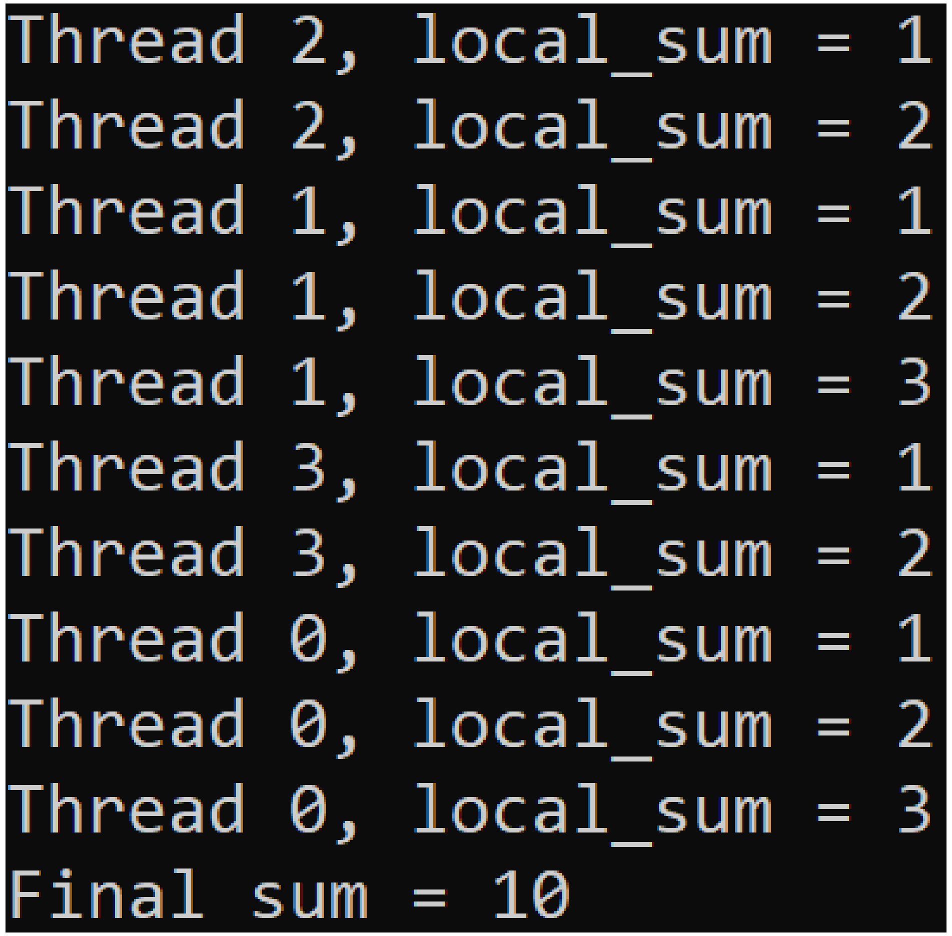


```
Thread 1, sum = 1
Thread 1, sum = 5
Thread 1, sum = 6
Thread 3, sum = 4
Thread 3, sum = 7
Thread 0, sum = 3
Thread 0, sum = 8
Thread 0, sum = 9
Thread 2, sum = 2
Thread 2, sum = 10
Final sum = 10
```

Parallel Loop with Local (private) Variable: 1. Explicit Summation

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

int main() {
    int sum = 0;

    #pragma omp parallel
    {
        int local_sum = 0;
        #pragma omp for
        for (int i = 0; i < 10; i++) {
            local_sum += 1;
            printf("Thread %d, local_sum = %d\n", omp_get_thread_num(), local_sum);
        }
        #pragma omp critical
        {
            sum += local_sum;
        }
    }

    printf("Final sum = %d\n", sum);

    return 0;
}
```

Output



```
Thread 2, local_sum = 1
Thread 2, local_sum = 2
Thread 1, local_sum = 1
Thread 1, local_sum = 2
Thread 1, local_sum = 3
Thread 3, local_sum = 1
Thread 3, local_sum = 2
Thread 0, local_sum = 1
Thread 0, local_sum = 2
Thread 0, local_sum = 3
Final sum = 10
```

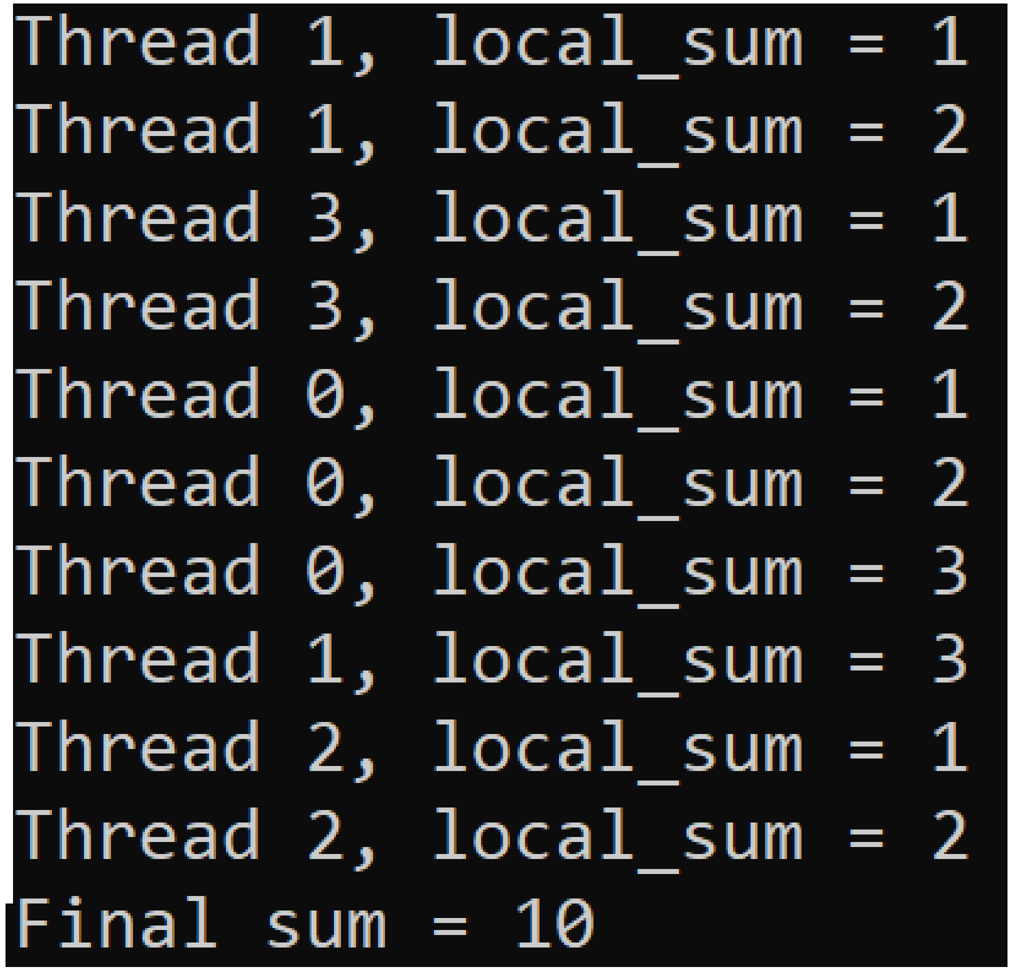
2. Reduce Operation: ‘reduction’

- With the **reduction(+:sum)** clause, each thread maintains its own private copy of sum, but at the end of the parallel loop, OpenMP combines these private copies using the **+** operator and assigns the result to the **sum** variable outside the loop. This way, you obtain the correct final value of sum.

‘reduction(+:sum)’

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

int main() {
    int sum = 0;

    #pragma omp parallel for reduction(+:sum)
    for (int i = 0; i < 10; i++) {
        sum += 1;
        printf("Thread %d, local_sum = %d\n", omp_get_thread_num(), sum);
    }

    printf("Final sum = %d\n", sum);

    return 0;
}
```

Output



```
Thread 1, local_sum = 1
Thread 1, local_sum = 2
Thread 3, local_sum = 1
Thread 3, local_sum = 2
Thread 0, local_sum = 1
Thread 0, local_sum = 2
Thread 0, local_sum = 3
Thread 1, local_sum = 3
Thread 2, local_sum = 1
Thread 2, local_sum = 2
Final sum = 10
```