NIKHIL N WAKODE 2022BCS0201

PARALLEL AND DISTRITUED COMPUTING

LAB-6 SET-6

Explore the following omp constructs:

1. parallel (different ways of creating threads)
2. single
3. master
4. worksharing using for
5. critical
6. barrier (explicit)

Code:

threads.cpp > main()

```cpp
#include <omp.h>
#include <iostream>
#include <iomanip>

using namespace std;

constexpr int N = 1000;

int main() {
    int arr[N];
    int tid = omp_get_thread_num();


    cout << "Parallel Region:" << endl;
    #pragma omp parallel
    {
        int tid = omp_get_thread_num();
        #pragma omp critical
        {

            cout << "Hello from thread " << tid << endl;
        }
    }
    cout << endl;
```

threads.cpp > main()

```cpp
int main() {
    #pragma omp parallel for
    for (int i = 0; i < N; i++) {
        arr[i] = i * 2;
    }
    cout << endl;
    #pragma omp master
        {
            cout << "Master thread (ID: " << tid << endl;
        }


    cout << "Sections Region" << endl;
    #pragma omp parallel sections
    {
        #pragma omp section
        {
            #pragma omp critical
            {

                cout << "Section 1 running on thread " << omp_get_thread_num() << endl;
            }
        }
        #pragma omp section
        {
            #pragma omp critical
            {
                cout << "Section 2 running on thread " << omp_get_thread_num() << endl;
            }
        }
    }
    cout << endl;
```

```cpp
    cout << "Tasks " << endl;
    #pragma omp parallel
    {
        #pragma omp single
        {
            #pragma omp task
            {
                #pragma omp critical
                {

                    cout << "Task 1 running on thread " << omp_get_thread_num() << endl;
                }
            }
            #pragma omp task
            {
                #pragma omp critical
                {
                    cout << "Task 2 running on thread " << omp_get_thread_num() << endl;
                }
            }
        }
    }
    cout << endl;



    return 0;
}
```

Output:

```
PS C:\Users\hp\Desktop\OpenMpLab6> g++ -fopenmp threads.cpp -o thread
PS C:\Users\hp\Desktop\OpenMpLab6> ./thread
Parallel Region:
Hello from thread 1
Hello from thread 2
Hello from thread 3
Hello from thread 5
Hello from thread 4
Hello from thread 6
Hello from thread 0
Hello from thread 7


Master thread ID: 0
Sections Region
Section 2 running on thread 1
Section 1 running on thread 4

Tasks
Task 1 running on thread 1
Task 2 running on thread 6

PS C:\Users\hp\Desktop\OpenMpLab6>
```

Codes:

```cpp
#include <omp.h>
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

const int N = 1000;

int main() {
    int data[N];
    int sum = 0;


    srand(time(nullptr));

    #pragma omp parallel
    {
        int thread_id = omp_get_thread_num();
        int num_threads = omp_get_num_threads();


        #pragma omp single
        {
            cout << "Initializing data with " << num_threads << " threads.\n";
            for (int i = 0; i < N; ++i) {
                data[i] = rand() % 100;
            }
        }
```

```cpp
int main() {
        }

        cout << " Critical section to prevent race conditions when updating sum" << endl;
        #pragma omp for
        for (int i = 0; i < N; ++i) {

            #pragma omp critical
            {
                sum += data[i];
            }
        }

        cout << "Explicit barrier to synchronize all threads before continuing" << endl;
        #pragma omp barrier



        cout << " only one thread prints the final result" << endl;
        #pragma omp single
        {
            cout << sum << "\n";
        }
    }

    return 0;
}
```

Output:

```
PS C:\Users\hp\Desktop\OpenMpLab6> ./code
Initializing data with 8 threads.
48255
PS C:\Users\hp\Desktop\OpenMpLab6>
```