Day9

Table of Contents

1. Agenda

- omp atomic
- omp sections
- nested parallelism

2. test

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

int main() {
    #pragma omp parallel num_threads(2)
    {
        #pragma omp sections
        {
            #pragma omp section
            {
                printf("Task 1 executed by thread %d\n", omp_get_thread_num());
                #pragma omp parallel for
                for (int i = 0; i < 5; i++) {
                    printf("  Task 1: Loop iteration %d executed by thread %d\n", i, omp_get_thread_num());
                }
            }
            #pragma omp section
            {
                printf("Task 2 executed by thread %d\n", omp_get_thread_num());
                #pragma omp parallel for
                for (int i = 0; i < 5; i++) {
                    printf("  Task 2: Loop iteration %d executed by thread %d\n", i, omp_get_thread_num());
                }
            }
        }
    }
    return 0;
}
```

```
gcc test.c -fopenmp
```

```
export OMP_NESTED=TRUE
./a.out
```

```
Task 1 executed by thread 1
Task 2 executed by thread 0
  Task 1: Loop iteration 1 executed by thread 1
  Task 1: Loop iteration 4 executed by thread 4
  Task 1: Loop iteration 0 executed by thread 0
  Task 2: Loop iteration 1 executed by thread 1
  Task 2: Loop iteration 0 executed by thread 0
  Task 2: Loop iteration 2 executed by thread 2
  Task 1: Loop iteration 3 executed by thread 3
  Task 2: Loop iteration 4 executed by thread 4
  Task 2: Loop iteration 3 executed by thread 3
  Task 1: Loop iteration 2 executed by thread 2
```

3. Critical sections

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
#define N 10000
#define T 10
int main(){
    int sum = 0;
    #pragma omp parallel for num_threads(10)
    for(int i = 0; i < N; i++){
        #pragma omp critical
        {
            sum+= i + 1;
        }
    }
    printf("Sum = %d\n", sum);

    return 0;
}
```

```
gcc criticalSection.c -fopenmp -o criticalSection.out
```

```
./criticalSection.out
```

```
Sum = 50005000
```

4. Atomic

- read
- write
- update
- capture

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
#define N 100000000
#define T 10
```

```c
int main(){
    long long  csum = 0, asum = 0;
    double startCritical = omp_get_wtime();
    #pragma omp parallel for num_threads(10)
    for(long long i = 0; i < N; i++){
        #pragma omp critical
        {
            csum+= i + 1;
        }
    }
    double endCritical = omp_get_wtime();

    double startAtomic = omp_get_wtime();
    #pragma omp parallel for num_threads(10)
    for(long long i = 0; i < N; i++){
        #pragma omp atomic
        asum+= i + 1;
    }
    double endAtomic = omp_get_wtime();
    printf("Time taken by critical section : %lf\n", endCritical - startCritical);
    printf("Critical sum : %lld\n", csum);
    printf("Time taken by atomic: %lf\n", endAtomic - startAtomic);
    printf("Atomic sum : %lld\n", asum);

    return 0;
}
```

```
gcc atomic.c -fopenmp -o atomic.out
```

```
./atomic.out
```

```
Time taken by critical section : 8.678401
Critical sum : 5000000050000000
Time taken by atomic: 3.726399
Atomic sum : 5000000050000000
```

5. Nested parallelism

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

```
#include<omp.h>
int main(){
    //omp_set_nested(1); //using this function you can enable desable nested parallelism
    #pragma omp parallel num_threads(2)
    {
        printf("Level 1 : Id %d\n", omp_get_thread_num());
        #pragma omp parallel num_threads(2)
        {
            printf("Level 2 : Id %d\n", omp_get_thread_num());
        }
    }
}
```

```
gcc nested.c -o nested.out -fopenmp
```

```
export OMP_NESTED=TRUE
./nested.out
```

```
Level 1 : Id 1
Level 1 : Id 0
Level 2 : Id 1
Level 2 : Id 0
Level 2 : Id 1
Level 2 : Id 0
```

6. Sections (task parallelism)

```
#include<stdio.h>
#include<omp.h>
#define N 10000
#define T 10
int main(){
    #pragma omp parallel num_threads(3)
    {
        #pragma omp sections
        {
            #pragma omp section
            {
                printf("This section is executed by thread %d\n", omp_get_thread_num());
            }
            #pragma omp section
            {
                printf("This section is executed by thread %d\n", omp_get_thread_num());
            }
            #pragma omp section
            {
                printf("This section is executed by thread %d\n", omp_get_thread_num());
            }
            #pragma omp section
            {
                printf("This section is executed by thread %d\n", omp_get_thread_num());
            }
        }
    }
}
```

```
gcc section.c -fopenmp -o section.out
```

```
./section.out
```

```
This section is executed by thread 1
This section is executed by thread 1
This section is executed by thread 1
This section is executed by thread 0
```

7. task parallelism

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
#define N 1000000
#define T 10
int main(){
    long long *a;
    long long sum = 0, sumSquare = 0;
    a = (long long*) malloc(sizeof(long long) * N);
    for(int i = 0; i < N; i++){
        a[i] = i + 1;
    }
    double startTime = omp_get_wtime();
    #pragma omp parallel num_threads(T)
    {
        #pragma omp sections
        {
            #pragma omp section
            {
                for(int i = 0; i < N; i++){
                    sum+= a[i];
                }
            }
            #pragma omp section
            {
                for(int i = 0; i < N; i++){
                    sumSquare += a[i] * a[i];
                }
            }
        }
    }
    double endTime = omp_get_wtime();
    printf("Sum = %lld\n", sum);
    printf("Sum of Squares = %lld\n", sumSquare);
    printf("Execution time = %lf\n", endTime - startTime);
    free(a);
}
```

```
gcc taskParallelism.c -fopenmp -o taskParallelism.out
```

```
./taskParallelism.out
```

```
Sum = 500000500000
Sum of Squares = 333333833333500000
Execution time = 0.001971
```

8. task parallelism1

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

```c
#include<stdlib.h>
#define N 1000000
#define T 10
int main(){
    long long *a;
    long long sum = 0, sumSquare = 0;
    a = (long long*) malloc(sizeof(long long) * N);
    for(int i = 0; i < N; i++){
        a[i] = i + 1;
    }
    omp_set_nested(1);
    double startTime = omp_get_wtime();
    #pragma omp parallel num_threads(T)
    {
        #pragma omp sections
        {
            #pragma omp section
            {
                #pragma omp parallel for reduction(+ : sum) num_threads(T)
                for(int i = 0; i < N; i++){
                    sum+= a[i];
                }
            }
            #pragma omp section
            {
                #pragma omp parallel for reduction(+ : sumSquare) num_threads(T)
                for(int i = 0; i < N; i++){
                    sumSquare += a[i] * a[i];
                }
            }
        }
    }
    double endTime = omp_get_wtime();
    printf("Sum = %lld\n", sum);
    printf("Sum of Squares = %lld\n", sumSquare);
    printf("Execution time = %lf\n", endTime - startTime);
    free(a);
}
```

```
gcc taskParallelism1.c -fopenmp -o taskParallelism1.out
```

```
./taskParallelism1.out
```

```
Sum = 500000500000
Sum of Squares = 333333833333500000
Execution time = 0.001393
```

9. task parallelism1

```c
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
#define N 1000000
#define T 10
int main(){
    long long *a;
    long long sum = 0, sumSquare = 0;
    a = (long long*) malloc(sizeof(long long) * N);
    for(int i = 0; i < N; i++){
        a[i] = i + 1;
    }
    for(int i = 0; i < N; i++){
            sum+= a[i];
    }
    for(int i = 0; i < N; i++){
            sumSquare += a[i] * a[i];
    }
    printf("Sum = %lld\n", sum);
    printf("Sum of Squares = %lld\n", sumSquare);
    free(a);
}
```

```
gcc taskParallelism1.c -fopenmp -o taskParallelism1.out
```

```
./taskParallelism1.out
```

```
Sum = 500000500000
Sum of Squares = 333333833333500000
Execution time = 0.001499
```

10. Locks

```
#include <omp.h>
#include <stdio.h>
omp_lock_t lock;

void solve(int thread_id) {
    omp_set_lock(&lock);
    printf("Thread %d is taking the lock\n", thread_id);

    printf("Thread %d is releasing the lock\n", thread_id);
    omp_unset_lock(&lock);
}

int main() {
    omp_init_lock(&lock);
    #pragma omp parallel
    {
        int thread_id = omp_get_thread_num();
        solve(thread_id);
    }
    omp_destroy_lock(&lock);
    return 0;
}
```

```
gcc locks.c -o locks.out -fopenmp
```

```
./locks.out
```

```
Thread 11 is taking the lock
Thread 11 is releasing the lock
Thread 8 is taking the lock
Thread 8 is releasing the lock
Thread 9 is taking the lock
Thread 9 is releasing the lock
Thread 10 is taking the lock
Thread 10 is releasing the lock
Thread 0 is taking the lock
Thread 0 is releasing the lock
Thread 3 is taking the lock
Thread 3 is releasing the lock
Thread 2 is taking the lock
Thread 2 is releasing the lock
Thread 1 is taking the lock
Thread 1 is releasing the lock
Thread 4 is taking the lock
Thread 4 is releasing the lock
Thread 6 is taking the lock
Thread 6 is releasing the lock
Thread 7 is taking the lock
Thread 7 is releasing the lock
Thread 5 is taking the lock
Thread 5 is releasing the lock
```

Author: Abhishek Raj

Created: 2025-01-03 Fri 12:04