Day7

Table of Contents

1. Agenda

- lastprivate
- critical section
- omp single
- omp master
- omp barrier
- parallel data creation

2. Lastprivate

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

```c
#define N 20
int main(){
    int a = 5;
    #pragma omp parallel for lastprivate(a) num_threads(T)
    for(int i = 0; i < N; i++){
        if(i == N - 1){
            continue;
        }
        a = i;
        printf("thread %d is changing a to %d\n", omp_get_thread_num(), a);
    }
    printf("a = %d\n", a);

    return 0;
}
```

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

```
./lastprivate.out
```

```
thread 1 is changing a to 2
thread 1 is changing a to 3
thread 4 is changing a to 8
thread 4 is changing a to 9
thread 2 is changing a to 4
thread 2 is changing a to 5
thread 9 is changing a to 18
thread 0 is changing a to 0
thread 0 is changing a to 1
thread 8 is changing a to 16
thread 8 is changing a to 17
thread 5 is changing a to 10
thread 5 is changing a to 11
thread 3 is changing a to 6
thread 3 is changing a to 7
thread 7 is changing a to 14
thread 7 is changing a to 15
thread 6 is changing a to 12
thread 6 is changing a to 13
a = 18
```

3. Lastprivate2

```c
#include<stdio.h>
#include<omp.h>
#define T 5
#define N 20
int main(){
    int a = 5;
    #pragma omp parallel for lastprivate(a) num_threads(T)
    for(int i = 0; i < N; i++){
        if(i > 16){
            continue;
        }
        a = i;
        printf("thread %d is changing a to %d\n", omp_get_thread_num(), a);
    }
    printf("a = %d\n", a);

    return 0;
}
```

```
gcc lastprivate2.c -o lastprivate2.out -fopenmp
```

```
./lastprivate2.out
```

```
thread 4 is changing a to 16
thread 0 is changing a to 0
thread 0 is changing a to 1
thread 0 is changing a to 2
thread 0 is changing a to 3
thread 2 is changing a to 8
thread 2 is changing a to 9
thread 2 is changing a to 10
thread 2 is changing a to 11
thread 1 is changing a to 4
thread 1 is changing a to 5
thread 1 is changing a to 6
thread 1 is changing a to 7
thread 3 is changing a to 12
thread 3 is changing a to 13
```

```
thread 3 is changing a to 14
thread 3 is changing a to 15
a = 16
```

4. Critical sections

```c
#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
```

5. You can use critical section for manual reduction

```c
#include<stdio.h>
#include<omp.h>
#define N 1000000000
```

```c
#define T 13
int main(){
    int chunksize = N / T;
    long long sum = 0;
    #pragma omp parallel num_threads(T)
    {
        int tid = omp_get_thread_num();
        long long localsum = 0;
        int start = tid * chunksize;
        int end = start + chunksize;
        if(tid == T - 1) end = N;
        for(int i = start; i < end; i++){
            localsum += i + 1;
        }
        #pragma omp critical
        {
            sum += localsum;
        }
    }

    printf("Calculated sum = %lld\n", sum);
    long long expectedSum = (N * ((N + 1) * 1L) / 2);
    printf("Expected sum = %lld\n", expectedSum);
    if(sum == expectedSum){
        printf("_____Passed_____\n");
    }
    else printf("_____Failed_____\n");

    return 0;
}
```

```
gcc criticalSection2.c -fopenmp -o criticalSection2.out
```

```
./criticalSection2.out
```

```
Calculated sum = 500000000500000000
Expected sum = 500000000500000000
_____Passed_____
```

6. Critical Section usage

```c
#include<stdio.h>
#include<omp.h>
#define N 1000000000
#define T 13
int main(){
    long long sum = 0;
    #pragma omp parallel num_threads(T)
    {
        int tid = omp_get_thread_num();
        long long localsum = 0;
        #pragma omp for
        for(int i = 0; i < N; i++){
            localsum += i + 1;
        }
        #pragma omp critical
        {
            sum+= localsum;
        }
    }

    printf("Calculated sum = %lld\n", sum);
    long long expectedSum = (N * ((N + 1) * 1L) / 2);
    printf("Expected sum = %lld\n", expectedSum);
    if(sum == expectedSum){
        printf("_____Passed_____\n");
    }
    else printf("_____Failed_____\n");

    return 0;
}
```

```
gcc criticalSection3.c -fopenmp -o criticalSection3.out
```

```
./criticalSection3.out
```

```
Calculated sum = 500000000500000000
Expected sum = 500000000500000000
_____Passed_____
```

7. Master

```c
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
#define N 10000
#define T 10
int main(){
    #pragma omp parallel num_threads(T)
    {
        #pragma omp master
        {
            printf("%d have an extra statement to execute\n", omp_get_thread_num());
        }
        printf("Hi, I am thread %d\n", omp_get_thread_num());
    }
    return 0;
}
```

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

```
./master.out
```

```
Hi, I am thread 9
Hi, I am thread 6
Hi, I am thread 5
Hi, I am thread 3
Hi, I am thread 4
0 have an extra statement to execute
Hi, I am thread 0
Hi, I am thread 8
Hi, I am thread 2
Hi, I am thread 1
Hi, I am thread 7
```

8. Single

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

```c
#define N 10000
#define T 10
int main(){
    #pragma omp parallel num_threads(T)
    {
        #pragma omp single
        {
            printf("%d have an extra statement to execute\n", omp_get_thread_num());
        }
        printf("Hi, I am thread %d\n", omp_get_thread_num());
    }
    return 0;
}
```

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

```
./single.out
```

```
9 have an extra statement to execute
Hi, I am thread 7
Hi, I am thread 6
Hi, I am thread 4
Hi, I am thread 8
Hi, I am thread 9
Hi, I am thread 0
Hi, I am thread 2
Hi, I am thread 3
Hi, I am thread 5
Hi, I am thread 1
```

9. Data creation inside parallel region using single

```c
#include<stdio.h>
#include<omp.h>
#define N 1000000
#define T 13
int main(){
    long long sum = 0;
    long long a[N];
```

```
    #pragma omp parallel  num_threads(T)
    {
        #pragma omp single
        {
            for(int i = 0; i < N; i++) a[i] = i + 1;
        }
        #pragma omp for reduction( + : sum )
        for(int i = 0; i < N; i++){
            sum += a[i];
        }
    }

    printf("Calculated sum = %lld\n", sum);
    long long expectedSum = (N * ((N + 1) * 1L) / 2);
    printf("Expected sum = %lld\n", expectedSum);
    if(sum == expectedSum){
        printf("_____Passed_____\n");
    }
    else printf("_____Failed_____\n");
    return 0;
}
```

```
gcc dc1.c -fopenmp -o dc1.out
```

```
./dc1.out
```

```
Calculated sum = 500000500000
Expected sum = 500000500000
_____Passed_____
```

10. Barrier

```
#include<stdio.h>
#include<unistd.h>
#include<omp.h>
#define T 5
int main(){
    #pragma omp parallel num_threads(T)
    {
```

```
        printf("Before barrier\n");
        #pragma omp barrier
        printf("After barrier\n");
    }

    return 0;
}
```

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

```
./barrier.out
```

```
Before barrier
Before barrier
Before barrier
Before barrier
Before barrier
After barrier
After barrier
After barrier
After barrier
After barrier
```

11. Data creation inside parallel region using master

```
#include<stdio.h>
#include<omp.h>
#define N 1000000
#define T 13
int main(){
    long long sum = 0;
    long long a[N];
    #pragma omp parallel  num_threads(T)
    {
        #pragma omp master
        {
            for(int i = 0; i < N; i++) a[i] = i + 1;
        }
        #pragma omp barrier
```

```
        #pragma omp for reduction( + : sum )
        for(int i = 0; i < N; i++){
            sum += a[i];
        }
    }

    printf("Calculated sum = %lld\n", sum);
    long long expectedSum = (N * ((N + 1) * 1L) / 2);
    printf("Expected sum = %lld\n", expectedSum);
    if(sum == expectedSum){
        printf("_____Passed_____\n");
    }
    else printf("_____Failed_____\n");
    return 0;
}
```

```
gcc dc2.c -fopenmp -o dc2.out
```

```
./dc2.out
```

```
Calculated sum = 500000500000
Expected sum = 500000500000
_____Passed_____
```

12. Barrier data creation

```
#include<stdio.h>
#include<omp.h>
#include<stdlib.h>
#define N 1000000
#define T 13

int main(){
    long long *a, sum=0;
    a= malloc(sizeof(long long) * N);

    #pragma omp parallel num_threads(T)
    {
        #pragma omp master
```

```
        {
            for(int i=0; i<N; i++){
                a[i] = i+1;
            }
        }
        #pragma omp for reduction(+:sum)
        for(int i=0; i<N; i++){
            sum += a[i];
        }
    }
    printf("sum = %lld\n", sum);

    return 0;
}
```

```
gcc dc3.c -fopenmp -o dc3.out
```

```
./dc3.out
```

```
sum = 2958689350
```

13. Data creation parallelly

```
#include<stdio.h>
#include<omp.h>
#define N 1000000
#define T 13
int main(){
    long long sum = 0;
    long long a[N];
    #pragma omp parallel  num_threads(T)
    {
        #pragma omp for
        for(int i = 0; i < N; i++){
            a[i] = i + 1;
        }
        #pragma omp for reduction( + : sum )
        for(int i = 0; i < N; i++){
            sum += a[i];
```

```
        }
    }

    printf("Calculated sum = %lld\n", sum);
    long long expectedSum = (N * ((N + 1) * 1L) / 2);
    printf("Expected sum = %lld\n", expectedSum);
    if(sum == expectedSum){
        printf("_____Passed_____\n");
    }
    else printf("_____Failed_____\n");
    return 0;
}
```

```
gcc dc4.c -fopenmp -o dc4.out
```

```
./dc4.out
```

```
Calculated sum = 500000500000
Expected sum = 500000500000
_____Passed_____
```

Author: Abhishek Raj
Created: 2024-12-20 Fri 09:58