

OpenMP并行程序设计(一)

在win11中, 使用CLion配置 OpenMP 环境, 需要在 CMakeLists.txt 中配置如下内容:

```
1  # openMP 配置
2  FIND_PACKAGE(OpenMP REQUIRED)
3  if (OPENMP_FOUND)
4      message("OPENMP FOUND")
5      set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${OpenMP_C_FLAGS}")
6      set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${OpenMP_CXX_FLAGS}")
7  endif ()
```

然后直接引用 `#include<omp.h>` 即可

先看一个简单的OpenMP程序, 展现线程顺序

```
1  #include <iostream>
2  #include "omp.h"
3
4  int main() {
5
6  #pragma omp parallel for
7      for (int i = 0; i < 10; ++i) {
8          std::cout << "第" << i << "个线程的 " << "Hello, World!" << std::endl;
9      }
10
11      return 0;
12  }
13
```

执行结果:

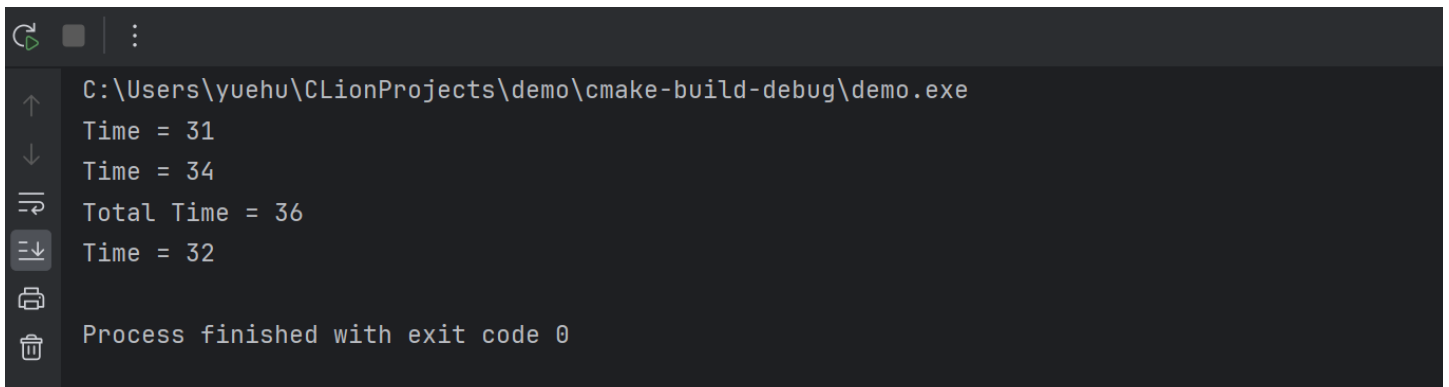
```
C:\Users\yuehu\CLionProjects\demo\cmake-build-debug\demo.exe
0
2
5
4
8
7
9
6
3
1
Process finished with exit code 0
```

可以看到默认有 9 个不同顺序的线程（不同的机器是不一样的，这里其实也表明了**并行是不分先后顺序的**）

测试引入OpenMP到 for 循环，变成并行执行，效率是否提升

```
1 //执行1亿次的测试案例
2 void test()
3 {
4     int a = 0;
5     clock_t t1 = clock();
6     for (int i = 0; i < 100000000; ++i) {
7         a = a+1;
8     }
9     clock_t t2 = clock();
10    cout << "Time = " << t2 - t1 << endl;
11
12 }
13
14 //主函数
15 int main() {
16
17     clock_t t1 = clock();
18    #pragma omp parallel for default(none) //这里不加default(none) 会警告, 原因是必须加 子句
19    for (int i = 0; i < 2; ++i) {
20        test();
21    }
22    clock_t t2 = clock();
23
24    cout << "Total Time = " << t2 - t1 << endl;
25
26    test();
27
28    return 0;
29 }
30
```

执行结果



```
C:\Users\yuehu\CLionProjects\demo\cmake-build-debug\demo.exe
Time = 31
Time = 34
Total Time = 36
Time = 32
Process finished with exit code 0
```

分析：

执行两次 1 亿 次的 `test()` 函数，时间仅为 36s，比单独执行一次 `test()` 仅多了 4s，提升了几乎一倍；

量还是太小了，显示不出多线程的太大优势，**因为引入 OpenMP 指令是需要额外开销的**