6 OpenMP中的竞争条件和同步

- **OpenMP 中为了保证程序正确性而采用哪些机制**
  - barriers，Mutual Exclusion 互斥，Memory fence 内存屏障
- 同步
  - **什么是同步**： 管理共享资源的过程，以便无论线程如何调度，都以正确的顺序进行读取和写入
  - **同步的主要方式**：barriers，互斥，信号量……
  - **OpenMP的Barriers**
    - 一个同步点，线程团队中的每个成员必须在任何成员继续之前到达该同步点，也就是在还有其他线程未到达时，每个线程都将被阻塞
    - 语法：pragma omp barrier
      - 在工作共享结构的末尾自动插入
      - nowait 可以使其失效
  - nowait 子句告诉编译器在并行 for 循环或单个代码块结束时不需要屏障同步
- 互斥
  - 一种同步
  - 一次只允许单个线程或进程访问共享资源
  - 使用某种形式的锁定实现
  - 临界区（高级同步）：一次只有一个线程将执行关键部分内的结构化块
  - 锁（低级同步），粒度更细
- 条件竞争
  - 竞争条件是由两个或多个线程访问共享变量的时间引起的非确定性行为
  - 例如，假设线程 A 和线程 B 都在执行语句……A更新的数没来得及写回，B就读取了相同的数
  - 竞争带来的影响
    - 具有竞争条件的程序表现出不确定的行为：有时给出正确的结果，有时会给出错误的结果
    - 程序通常在琐碎的数据集和少量线程上正常工作
    - 当线程数和/或执行时间增加时更容易发生错误
    - 因此调试竞争条件可能很困难
  - **避免竞争**
    - 变量作用域线程私有化
      - 使用 OpenMP private子句
      - 在线程函数中声明的变量
      - 在线程的栈上分配（作为参数传递）

    - 控制关键区的共享访问
        - 互斥与同步
            - 确保一次只有一个线程引用或更新共享变量
            - 互斥，一种同步
            - 一次只允许单个线程或进程访问共享资源
            - 使用某种形式的锁定实现
- 锁机制
    - 以前的方法失败了，因为检查 flag 的值和设置它的值是两个不同的操作
    - 我们需要某种原子测试和设置
    - 操作系统提供了执行此操作的功能——锁
    - 用于控制对共享资源的访问的同步机制（一个通用术语）
- critical
    - critical section：一次只能执行一个线程的部分代码（互斥）
    - OpenMP 原语：pragma omp critical
    - 减少竞争
    - 串行执行
    - 自己识别定义

```
void AddHead(struct List* list, struct Node* node) {
#pragma omp critical
    {
        node->next = list->head;
        list->head = node;
    }
}
```

- 原子操作
    - critical section的特殊情况，以确保对内存位置进行原子更新
    - 仅适用于简单操作：
        - 前增量或后增量 (++)
        - 减量前或减量后 (--)
        - 使用二元运算符（标量类型）赋值
    - 适用于单个语句
        - #pragma omp atomic
        - counter+= 5;
    - **cirtical和atomic的对比**

```
#pragma omp parallel for
{
    for (i = 0; i < n; i++) {
#pragma omp critical
        x[index[i]] += WorkOne(i);
        y[i] += WorkTwo(i);
    }
}
```
A

```
#pragma omp parallel for
{
    for (i = 0; i < n; i++) {
#pragma omp atomic
        x[index[i]] += WorkOne(i);
        y[i] += WorkTwo(i);
    }
}
```
B

- Critical 确保一次只有一个线程执行结构化代码。 atomic 只能用在形如 x = 、x++、x--之类的临界区中，他比普通的临界区执行速度快。
- cirtical
  - cirtical保护
    - WorkOne(i)的调用过程，也就是在WorkOne(i)函数内部也是临界区
    - 从内存中找到index[i]值的过程
    - 将WorkOne(i)的值加到index[i]的过程
  - 将内存中index[i]更新的过程　后面跟的语句相当于串行执行
- atomic
  - atomic保护
    - 将WorkOne(i)的值加到index[i]
  - 更新内存中index[i]的值
  - 只将单一一条语句设置为临界区
  - 如果不同线程的index[i]非冲突的话仍然可以并行完成
  - 只有当两个线程的index[i]相等时才会触发使得这两个线程排队执行这条语句

- 总结
  - 同步（在 OpenMP 中）
    - barrier
      - 不同线程之间的语句排序
      - 屏障之后的任何语句都将在每个线程中屏障之前的语句之后执行
  - 互斥
    - 共享资源的访问顺序
    - 避免竞争条件的机制
  - memory fence
    - 同一线程中的数据相关语句排序
    - 内存围栏之前的任何与数据相关的语句都将在内存围栏之后的语句之前执行

以上内容整理于 幕布文档