OpenMP 常用语句（指令对）

（1）!$OMPDO/!$OMP END DO

该指令对使得最外层的 do 循环并行执行，即将 do 循环分散到不同的线程，每个线程仅计算其中一部分。例如：

```
!$OMP DO
do i = 1, 1000
...
end do
!$OMP END DO
```

（2）!$OMP SECTIONS/!$OMPEND SECTIONS

该指令对给每个线程分派完全不同的任务，每段代码仅被一个线程执行一次，指令对格式如下：

```
!$OMP SECTIONS clause1 clause2 ...
!$OMP SECTION
...
!$OMP SECTION
...
...
!$OMP END SECTIONS end_clause
```

其中 clause1，clause2 等是其他修饰语句（子句）。

应用实例如下：

```
!$OMP SECTIONS
!$OMP SECTION
write(*,*) "Hello"
!$OMP SECTION
write(*,*) "Hi"
!$OMP SECTION
write(*,*) "Bye"
!$OMP END SECTIONS
```

“Hello”、“Hi”、“Bye”这三个信息只各在屏幕上出现一次，由三个不同线程同步执行。

（3）!$OMP SINGLE/!$OMP END SINGLE

该指令对包含的代码仅由其中一条线程执行，也就是最先到达!$OMP SINGLE 指令的线程。指令对格式如下：

```
!$OMP SINGLE clause1 clause2 ...
...
!$OMP END SINGLE end_clause
```

应用实例如下：

```
!$OMP SINGLE
write(*,*) "Hello"
!$OMP END SINGLE
```

“Hello”只在屏幕出现一次，其余线程并不执行指令对内的代码，而是在关闭指令处闲置、等待。

（4）!$OMPMASTER/!$OMPENDMASTER

指令对中的代码只被主线程执行，其他线程继续执行它们的工作。

```
!$OMPMASTER
write(*,*) "Hello"
!$OMP ENDMASTER
```

（5）!$OMPCRITICAL/!$OMP END CRITICAL

每次只有一条线程进入内部代码块，以确保内部代码正确执行。

```
!$OMPCRITICAL name
...
!$OMP END CRITICAL name
```

可选参数 name 用于识别临界区域。name 虽然不是强制的，但仍建议为每个区域添加。当一条线程到达临界区域的开始部分时，它会在此等待，直到没有其他线程执行其中的代码为止。有相同名字的不同临界区域被看作共同的临界区域，一次只有一个线程在里面。此外，所有的未命名临界区都将看作同一个区域，这也是建议命名 name 的原因。

下面演示从键盘或文件读取数据更新变量，由于只需要一个线程读取，所以使用!$OMPCRITICAL 适合该操作：

```
!$OMPCRITICAL write_file
write(1,*) data
!$OMP END CRITICAL write_file
```

（6）!$OMP BARRIER

某个线程遇到该指令时将处于等待状态，直到所有线程都到达该指令，这一过程也称为显式同步。

（7）!$OMP ATOMIC

如果使用的变量能被所有线程修改，该指令能够确保只有一个线程在修改变量的内存地址，否则会引起程序错误。

（8）数据作用域属性子句：私有（PRIVATE）和共享（SHARED）变量

前面介绍的许多指令都可以通过添加子句（上文中的 clause1、clause2 等）的形式改变工作方式。可以定义两种不同的子句：数据作用域属性子句和其他子

句。我们首先来看数据作用域属性子句。

首先是 PRIVATE(list)子句，顾名思义，表示对于各个线程私有的变量，例如：

```
!$OMP PARALLEL PRIVATE(a, b)
...
!$OMP END PARALLEL
```

表示在!$OMP PARALLEL/!$OMP END PARALLEL 指令对内，变量 *a*、*b* 在不同线程中拥有不同的值，对于各个线程而言，它是局部变量。

而对于所有线程共享的变量，可以通过 SHARED(list)子句定义，例如：

```
!$OMP PARALLEL SHARED(c, d)
...
!$OMP END PARALLEL
```

表示在!$OMP PARALLEL/!$OMP END PARALLEL 指令对内，所有线程都可以使用变量 *c*、*d*。

当指令对中多数变量属性为共享或私有，将他们全部都写入子句中显得很笨重。针对这种情况，可以指定 DEFAULT 设置：

```
!$OMP PARALLEL DEFAULT(PRIVATE) SHARED(a)
```

表示除了变量 *a* 为共享属性，其余均为私有属性。当然也可以：

```
!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(a)
```

表示除了变量 *a* 为私有属性，其余均为共享属性。如果没有指定 DEFAULT 子句，默认为共享属性。

私有变量在指令对开头具有不确定的值，但有时候这些局部变量在指令对开始部分可以拥有初始值。将变量包含进 FIRSTPRIVATE 子句：

```
a=2
b=1
!$OMP PARALLEL PRIVATE(a) FIRSTPRIVATE(b)
```

此例中 *a* 具有不确定值，而 *b* 的初值为 1。如果在指令对的开始部分将变量包含进 FIRSTPRIVATE 子句，则变量自动具有私有属性，而不需要再次声明其私有属性。

同样的，私有变量在指令对结尾处具有不确定值，很多时候非常不方便。如将其置于 LASTPRIVATE 子句中，原变量将会被最后的值更新，如同在串行程序里一样。

```
!$OMP DO PRIVATE(i) LASTPRIVATE(a)
do i = 1, 1000
a=i
enddo
```

```
!$OMP END DO
```

指令对完成后，*a* 的值为 1000。

此外，对于共享变量，需要确保同时仅有一个线程访问或者更新其值，否者将出现不可预料的结果：

```
!$OMP DO
do i = 1, 1000
a=a+i
enddo
!$OMP END DO
```

期望值为循环变量 *i* 的总和，但实际可能并不如此。子句 REDUCTION 可解决这个问题，该子句保证同时只有一个线程更新 *a* 的值，可确保结果的正确性。上例可改为：

```
!$OMP DO REDUCTION(+:a)
do i = 1, 1000
a=a+i
enddo
!$OMP END DO
```

REDUCTION 子句的语法：在列表中变量前有一操作符，操作符与变量同时出现。

判断变量是私有还是共享，可依据一个简单原则。除了以下三种情况外，并行区域中的所有变量都是共享的：1）并行区域中定义的变量；2）循环变量；3）PRIVATE、FIRSTPRIVATE、LASTPRIVATE 或 REDUCTION 子句修饰的变量。

（9）其他子句

特定并行区需要使用固定数量的线程，如!$OMP SECTIONS 指令中的任务。NUM_THREADS 子句定义线程数目，下例使用了 4 个线程：

```
!$OMP PARALLEL NUM_THREADS(4)
```

指定的线程数仅影响当前并行区。

某些情况下，并不需要所有线程同时结束工作，因为接下来的操作不依赖于之前的结果。这种情况下，应该在指令对结尾处添加 NOWAIT 子句，以避免隐式同步。下例：

```
!$OMP PARALLEL
!$OMP DO
do i = 1, 1000
a=i
enddo
!$OMP END DO NOWAIT
!$OMP DO
do i = 1, 1000
```

```
b=i
enddo
!$OMP END DO
!$OMP END PARALLEL
```

如果第二个循环与前一个无关，在第一个并行循环结束的地方，并不需要等待所有线程结束。因此在!$OMP END DO 的关闭指令添加 NOWAIT 子句，禁用最近的同步，提高计算速度。

当循环中有需要顺序执行的语句时，如下：

```
do i = 1, 1000
A(i) = 2 * A(i-1)
End do
```

!$OMP ORDERED/!$OMP END ORDERED 指令对用于指定顺序区。在!$OMP DO 打开指令后强制添加 ORDERED，告诉编译器存在顺序区：

```
!$OMP DO ORDERED
do i = 1, 1000
!$OMP ORDERED
A(i) = 2 * A(i-1)
!$OMP END ORDERED
enddo
!$OMP END DO
```