!$omp parallel

- Creates a team of threads that execute a region
- Clauses
  - **num_threads(***integer***)**
    determines the number of threads in the parallel region
  - **private(***list***)** / **shared(***list***)**
    determines whether each thread has its own copy of the variables in list or whether they all access the same copy
  - **reduction(***identifier : list***)**
    used to perform sums or compute minimax/maxima across all threads

Parallel regions

Example

```
! only one thread here

!$omp parallel [clause]
! team of threads here
!$omp end parallel

! only one thread here
```

!$omp parallel do

Parallel loops

- Distributes the iterations of a loop to a team of threads
- Clauses
  - Same clauses as **`!$omp parallel`**
  - **`collapse(`***integer*`)`** distributes the iterations of more than one loop
  - **`schedule(`***kind*`)`** determines how the iteration space is distributed to the threads
- The loop index is automatically private

Example

```
!$omp parallel do [clause]
do i = 1, 100
  a(i) = b(i) + c(i)
end do
!$omp end parallel do
```

Warning

Not all loops can be parallelized. It is your responsibility to decide whether parallelization is legal and beneficial.

!$omp critical

- All threads execute the region, but the region is only executed by a single thread at a time.
- Often used to avoid race conditions, where multiple threads write to the same memory location

One thread at a time

Example

```
count = 0

!$omp parallel do
do i = 1, 100

  ! do some work
  a(i) = b(i) + c(i)

  !$omp critical
  count = count + 1
  !$omp end critical

end do
!$omp end parallel do
```

!$omp single / master

- Only one thread executes the region.
- Often used to do operations such as writing to standard output or reading files from disk.

Only one thread

Example

```
!$omp parallel

! do some parallel work

!$omp single
print *, 'I am rank ', omp_get_thread_num()
!$omp end single

! do some parallel work

!$omp master
print *, 'I am master'
!$omp end master

!$omp end parallel do
```

!$omp barrier

- Threads are only allowed to continue once all threads of the team have reached this statement.

Wait for all threads

Example

```
!$omp parallel

! do some parallel work
call work()

! wait for all threads to finish
!$omp barrier

!$omp end parallel do
```

OpenMP Runtime Library

- **`call omp_set_num_threads(integer)`**
  Determines the number of threads for subsequent **`!$omp parallel`** constructs (without a **`num_threads`** clause).

- ***`integer`*** **`omp_get_num_threads()`**
  Returns the number of threads in the current team. Returns 1 if called from sequential part of program (outside of parallel region).

- ***`integer`*** **`omp_get_thread_num()`**
  Returns the thread number of the calling threads, within the current team.

!$

- Statements are only executed if compiled with OpenMP support.
- Often used to call OpenMP Runtime Library routines.

Conditional execution

Example

```
!$omp parallel
!$omp master
! only executed if compiled with OpenMP support
!$ num_threads = omp_get_num_threads()
!$ write(*,*) '#threads = ', num_threads
!$omp end master
!$omp end parallel
```

Compiling and running with OpenMP

- Compilers need to be instructed with a flag to compile using OpenMP

Example

```
ftn -h omp -o test.exe test.F90
gfortran -openmp -o test.exe test.F90
```

- Set the number of threads with an environment variables before running your executable.

```
OMP_NUM_THREADS=4 ./test.exe

export OMP_NUM_THREADS=4
./test.exe
```