Parallel Construct

ICHEC
Irish Centre for High-End Computing

ICHEC
Irish Centre for High-End Computing

OpenMP Directive Format

- Compiler Directives appear as pragmas in C/C+ and comments in Fortran, executed when the appropriate OpenMP flag is specified
  - Control
    - how is parallelism created?
    - what ordering is there between operations?
  - Synchronization
    - What operations are used to coordinate parallelism ?
    - What operations are atomic (indivisible)?
  - Data
    - What data is private or shared?
    - How is logically shared data accessed or communicated?

ICHEC
Irish Centre for High-End Computing

OpenMP Directive Format

C/C++:

```
#pragma omp directive-name [clause[clause]...]
{
...
}
```

Fortran free form:

```
!$omp directive-name [clause[clause]...]
...
!$omp end directive-name
```

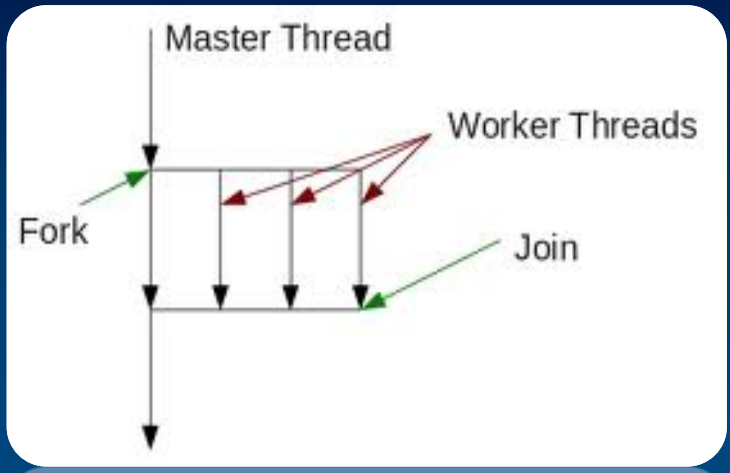
Fortran fixed form:

```
!$omp | c$omp | *$omp directive-name [clause[clause]...]
...
!$omp | c$omp | *$omp  end directive-name
```

ICHEC
Irish Centre for High-End Computing

Parallel Constructs

- The fundamental construct in OpenMP.
- Creates team of threads
- Every thread executes the same statements inside the parallel region at the end of the parallel region there is an implicit barrier



C/C++:

```
#pragma omp parallel [clauses]
{
    …
}
```

Fortran:

```
!$omp parallel [clauses]
   …
!$omp end parallel
```

ICHEC
Irish Centre for High-End Computing

Main Parallel Construct Clauses

- Clauses:

```
num_threads (integer-expression)
if (scalar_expression)
nowait
```

  - Data Clauses:

```
private (list)
shared (list)
default (shared | none)
firstprivate (list)
reduction (operator: list)
copyin (list)
```

ICHEC
Irish Centre for High-End Computing

How many threads?

The number of threads in a parallel region is determined by:

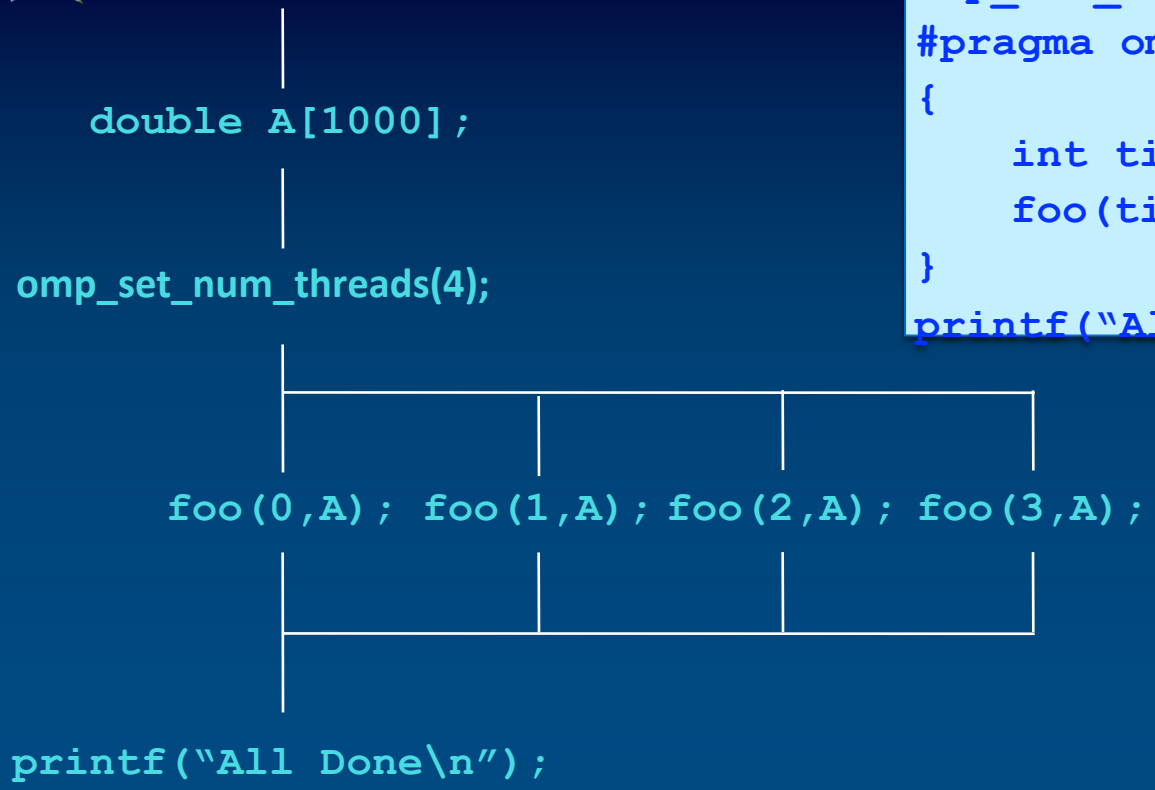
- Use of `num_threads(n)` clause.

```
#pragma omp parallel num_threads(4)
   {
      …
   }
```

- Setting as an environment variable.
  - csh/tcsh: `setenv OMP_NUM_THREADS n`
  - ksh/sh/bash: `export OMP_NUM_THREADS=n`
- Use of the `omp_set_num_threads(n)` library function.
  - C/C++: Add `#include<omp.h>`
  - Fortran: Add `use omp_lib`
- The implementation default - usually the number of CPUs/cores on a node

Threads are numbered from 0 (master thread) to n-1 n: the total number of threads.

ICHEC
Irish Centre for High-End Computing




```
double A[1000];
omp_set_num_threads(4);
#pragma omp parallel
{
    int tid = omp_get_thread_num();
    foo(tid,A);
}
printf("All Done\n");
```

- `omp_get_thread_num()` - returns the id of this thread
- `omp_get_num_threads()` - returns the current number of threads

ICHEC
Irish Centre for High-End Computing

Hello World

C - Serial:

```
#include<stdio.h>

int main(int argc, char**argv)
{
    printf("Hello world!\n");
}
```

C - Parallel:

```
#include<stdio.h>
#include<omp.h>
int main(int argc, char**argv)
{
    #pragma omp parallel
    printf("Hello from thread %d out of %d\n", omp_get_thread_num(),
                                               omp_get_num_threads());
}
```

ICHEC
Irish Centre for High-End Computing

Hello World

Fortran - Serial:

```
program hello

print *, 'Hello world!'

end program hello
```

Fortran - Parallel:

```
program hello
use omp_lib

!$omp parallel
print *, 'Hello from thread', omp_get_thread_num(),'out of', &
       omp_get_num_threads()
!$omp end parallel

end program hello
```

ICHEC
Irish Centre for High-End Computing

Compile and Run

```
$ gcc -fopenmp hello.c -o hello
$ gfortran -fopenmp hello.f90 -o hello
$ export OMP_NUM_THREADS=12
$ ./hello
```

```
Hello from thread 0 out of 12
Hello from thread 8 out of 12
Hello from thread 10 out of 12
Hello from thread 11 out of 12
Hello from thread 4 out of 12
Hello from thread 9 out of 12
Hello from thread 7 out of 12
Hello from thread 5 out of 12
Hello from thread 3 out of 12
Hello from thread 6 out of 12
Hello from thread 1 out of 12
Hello from thread 2 out of 12
```

ICHEC
Irish Centre for High-End Computing

If Clause:

- Used to make the parallel region directive itself conditional.
- Only execute in parallel if expression is true.

C/C++:

```
#pragma omp parallel if(n>100)
{
    …
}
```

Fortran:

```
!$omp parallel if(n>100)
    ...
!$omp end parallel
```

nowait Clause:

- allows threads that finish earlier to proceed without waiting

```
#pragma omp parallel nowait
{
    …
}
```

```
!$omp parallel
    ...
!$omp end parallel nowait
```

ICHEC
Irish Centre for High-End Computing

Main Data Clauses

- Used in conjunction with several directives to control the scoping of enclosed variables.
  - `default(shared|none)` : The default scope for all of the variables; private for Fortran
  - `shared(list)` : Variable is shared by all threads in the team. All threads can read or write to that variable.

    C/C++:
    ```
    #pragma omp parallel default(none) shared(n)
    ```
    Fortran:
    ```
    !$omp parallel default(none) shared(n)
    ```
  - `private(list)` : Each thread has a private copy of variable. It can only be read or written by its own thread.

    C/C++:
    ```
    #pragma omp parallel default(shared) private(tid)
    ```
    Fortran:
    ```
    !$omp parallel default(shared) private(tid)
    ```

ICHEC
Irish Centre for High-End Computing

Example

C:

```
#include<stdio.h>
#include<omp.h>
int main()
{
    int tid, nthreads;
    #pragma omp parallel private(tid), shared(nthreads)
    {
        tid = omp_get_thread_num();
        nthreads = omp_get_num_threads();
        printf("Hello from thread %d out of %d\n", tid, nthreads);
    }
}
```

ICHEC
Irish Centre for High-End Computing

Example

Fortran:

```
program hello
use omp_lib
implicit none
integer tid, nthreads

!$omp parallel private(tid), shared(nthreads)
tid = omp_get_thread_num()
nthreads = omp_get_num_threads()
print*, 'Hello from thread',tid,'out of',nthreads
!$omp end parallel

end program hello
```

ICHEC
Irish Centre for High-End Computing

- How do we decide which variables should be shared and which private?
  - Loop indices - private
  - Loop temporaries - private
  - Read-only variables - shared
  - Main arrays - shared
- Most variables are shared by default
  - **C/C++:** File scope, static variables
  - **Fortran:** COMMON blocks, SAVE, MODULE variables
  - **Both:** dynamically allocated variables
- Variables declared in parallel region are always private

ICHEC
Irish Centre for High-End Computing

Additional Data Clauses

- **`firstprivate(list)`**: pre-initialize private vars with value of variable with same name before parallel construct.
- **`lastprivate(list)`**: On exiting the parallel region, this gives private data the value of last iteration (if sequential)
- **`threadprivate(list)`**: Used to make global file scope variables (C/C++) or common blocks (Fortran) private to thread.
- **`copyin(list)`**: Copies a value from master thread to all threadprivate variables of a thread team.

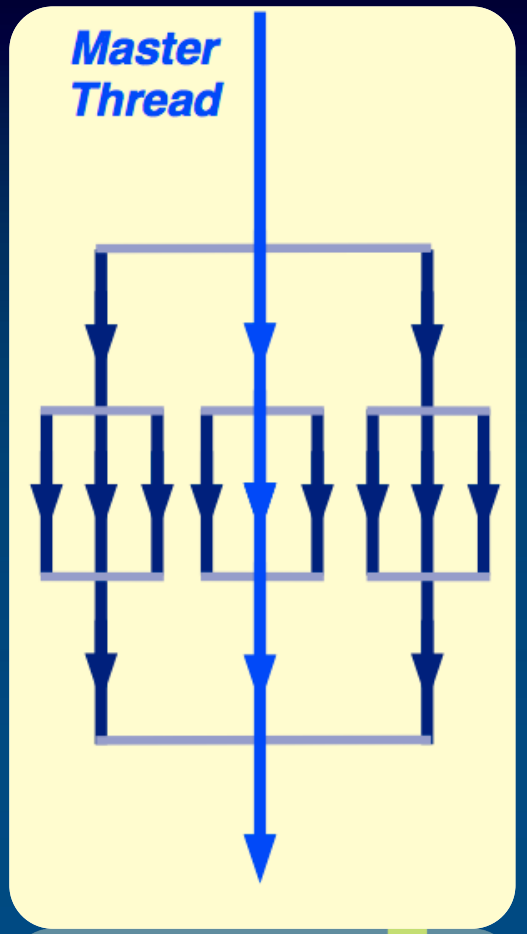
```
j = jstart;
#pragma omp parallel for firstprivate(j)
{
     for(i=1; i<=n; i++)
     {
       j = j + 1;
       a[i] = a[i] + j;
}
}
```

```
#pragma omp parallel for lastprivate(x)
{
     for(i=1; i<=n; i++)
     {
          x = sin( pi * dx * (float)i );
a[i] = exp(x);
     }
}
lastx = x;
```

ICHEC
Irish Centre for High-End Computing

Nested parallel regions:

- If a parallel directive is encountered within another parallel directive, a new team of threads will be created.
- `omp_set_nested()`, `OMP_NESTED`, `omp_get_nested()`
- Num threads affects the new regions
- New threads with one thread unless nested parallelism is enabled
- `num_threads(n)` clause or dynamic threading for different num threads



ICHEC
Irish Centre for High-End Computing

Dynamic threads:

- Used to create a parallel region with a variable number of threads
- `omp_set_dynamic(), OMP_DYNAMIC, omp_get_dynamic()`
- OpenMP runtime will decide the number of threads

```
omp_set_dynamic(0);
omp_set_num_threads(10);
#pragma omp parallel
printf("Num threads in non-dynamic region = %d\n",  omp_get_num_threads());

omp_set_dynamic(1);
omp_set_num_threads(10);
#pragma omp parallel
printf("Num threads in dynamic region is = %d\n", omp_get_num_threads());
```

ICHEC
Irish Centre for High-End Computing

Work-Sharing Constructs

- Work-sharing construct divides work among the thread team; what kind of work to be parallelised.
- Specify inside the parallel region.
- No new threads created. Construct must be encountered by all threads in the team.
- No implied barrier on entry to a work-sharing construct; Yes at end of construct.
- Work-sharing Constructs:
  - Loop
  - Sections
  - Single
  - Workshare

ICHEC
Irish Centre for High-End Computing

Sections Construct

- Specifies enclosed sections that are distributed over the threads in the team; implicit barrier at the end
- Each section is executed by one thread.
- Functional parallelism

C/C++:

```
#pragma omp parallel [clauses]
{
#pragma omp sections [clauses] nowait
{
    #pragma omp section
    …
    #pragma omp section
    …
}
}
```

Fortran:

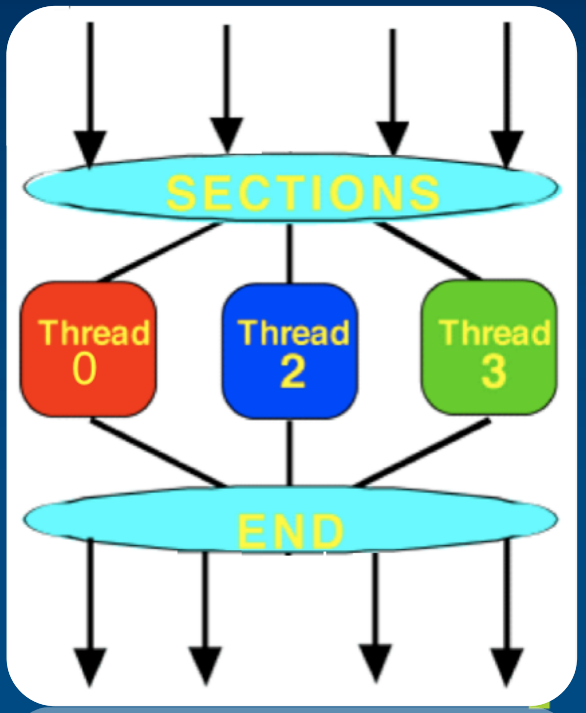
```
!$omp parallel [clauses]
!$omp sections [clauses]
   !$omp section
   ...
   !$omp section
   ...
!$omp end sections
[nowait]
!$omp end parallel
```

ICHEC
Irish Centre for High-End Computing

```
$export OMP_NUM_THREADS=4
```

```
#include <stdio.h>
#include <omp.h>
int main()
{
    int tid;
    #pragma omp parallel private(tid)
    {
        tid = omp_get_thread_num();
        #pragma omp sections
        {
            #pragma omp section
            printf("Hello from thread %d \n", tid);
            #pragma omp section
            printf("Hello from thread %d \n", tid);
            #pragma omp section
            printf("Hello from thread %d \n", tid);
        }
    }
}
```

```
Hello from thread 0
Hello from thread 2
Hello from thread 3
```



ICHEC
Irish Centre for High-End Computing

Sections Construct Main Clauses

- Clauses:

```
private (list)
firstprivate (list)
lastprivate (list)
reduction (operator: list)
nowait
```

ICHEC
Irish Centre for High-End Computing

Single Construct

- Specifies a block of code that is executed by only one of the threads in the team.
- Rest of threads wait at the end of enclosed code block
- May be useful when dealing with sections of code that are not thread-safe.

C/C++:

```
#pragma omp parallel [clauses]
{
    #pragma omp single [clauses]
    …
}
```

Fortran:

```
!$omp parallel [clauses]
   !$omp single [clauses]
   ...
   !$omp end single
!$omp end parallel
```

- `copyprivate(list)`: used to broadcast private variable values from a single thread to all instances of the private variables in the other threads.

ICHEC
Irish Centre for High-End Computing

Workshare construct

- Fortran only
- Divides the execution of the enclosed structured block into separate units of work
- Threads of the team share the work
- Each unit is executed only once by one thread
- Allows parallelisation of
  - array and scalar assignments
  - WHERE statements and constructs
  - FORALL statements and constructs
  - parallel, atomic, critical constructs

```
!$omp workshare
    ...
!$omp end workshare
[nowait]
```

ICHEC
Irish Centre for High-End Computing

```
program wshare
use omp_lib
implicit none

integer :: i
real :: a(10), b(10), c(10)
do i=1,10
   a(i)=i
   b(i)=i+1
enddo

!$omp parallel shared(a, b, c)
!$omp workshare
   c=a+b
!$omp end workshare nowait
!$omp end parallel

end program wshare
```