



Threaded Programming

Lecture 3: Parallel Regions

- Code within a parallel region is executed by all threads.
- Syntax:

Fortran: **!\$OMP PARALLEL**

block

!\$OMP END PARALLEL

C/C++: **#pragma omp parallel**

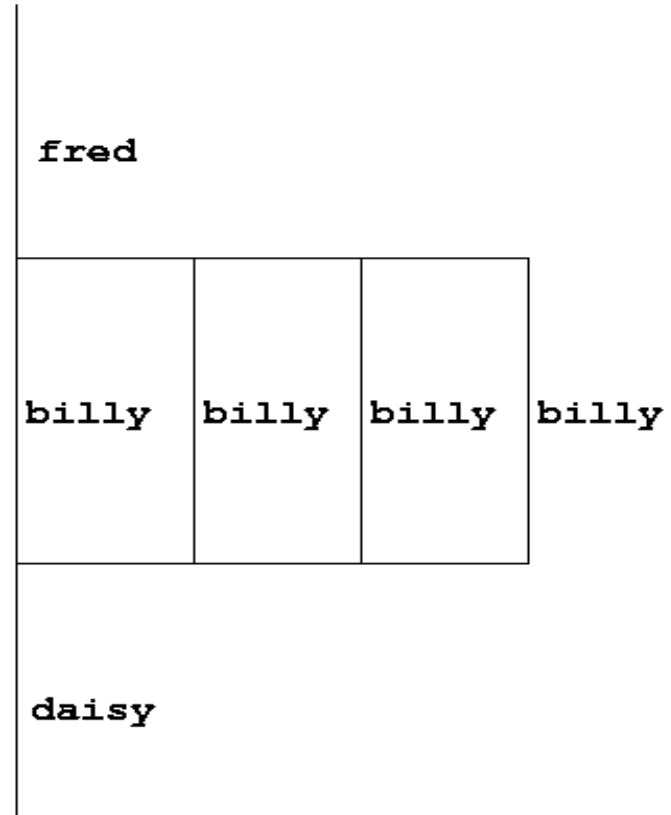
{

block

}

Example:

```
fred();  
#pragma omp parallel  
{  
    billy();  
}  
daisy();
```



- Often useful to find out number of threads being used.

Fortran:

```
USE OMP_LIB  
INTEGER FUNCTION OMP_GET_NUM_THREADS ()
```

C/C++:

```
#include <omp.h>  
int omp_get_num_threads(void);
```

- **Important note:** returns 1 if called outside parallel region!

- Also useful to find out number of the executing thread.

Fortran:

```
USE OMP_LIB
```

```
INTEGER FUNCTION OMP_GET_THREAD_NUM()
```

C/C++:

```
#include <omp.h>
```

```
int omp_get_thread_num(void)
```

- Takes values between 0 and `OMP_GET_NUM_THREADS () - 1`

- Specify additional information in the parallel region directive through *clauses*:

Fortran : **!\$OMP PARALLEL** [*clauses*]

C/C++: **#pragma omp parallel** [*clauses*]

- Clauses are comma or space separated in Fortran, space separated in C/C++.

- Inside a parallel region, variables can be either **shared** (all threads see same copy) or **private** (each thread has its own copy).
- Shared, private and default clauses

Fortran: **SHARED** (*list*)

PRIVATE (*list*)

DEFAULT (SHARED|PRIVATE|NONE)

C/C++: **shared** (*list*)

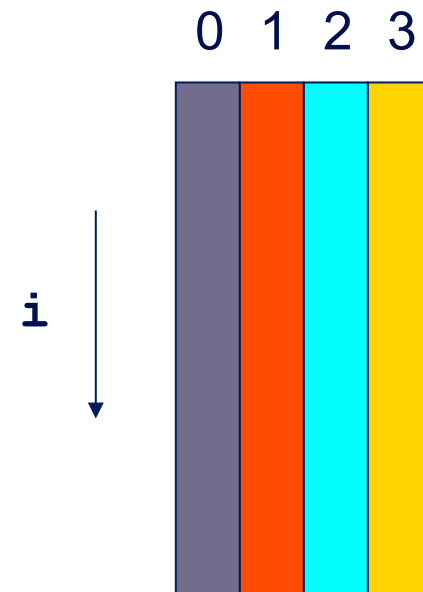
private (*list*)

default (shared|none)

- On entry to a parallel region, private variables are uninitialised.
- Variables declared inside the scope of the parallel region are automatically private.
- After the parallel region ends the original variable is unaffected by any changes to private copies.
- Not specifying a DEFAULT clause is the same as specifying DEFAULT(SHARED)
 - **Danger!**
 - Always use DEFAULT(NONE)

Example: each thread initialises its own column of a shared array:

```
!$OMP PARALLEL DEFAULT (NONE) , PRIVATE (I ,MYID) ,  
!  
!$OMP & SHARED (A ,N)  
  
    myid = omp_get_thread_num() + 1  
    do i = 1,n  
        a(i,myid) = 1.0  
    end do  
!  
!$OMP END PARALLEL
```



Multi-line directives

- Fortran: fixed source form

```
!$OMP PARALLEL DEFAULT(NONE) , PRIVATE(I,MYID) ,  
!$OMP& SHARED(A,N)
```

- Fortran: free source form

```
!$OMP PARALLEL DEFAULT(NONE) , PRIVATE(I,MYID) , &  
!$OMP SHARED(A,N)
```

- C/C++:

```
#pragma omp parallel default(none) \  
private(i,myid) shared(a,n)
```

- Private variables are uninitialised at the start of the parallel region.
- If we wish to initialise them, we use the `FIRSTPRIVATE` clause:

Fortran: **FIRSTPRIVATE** (*list*)

C/C++: **firstprivate** (*list*)

- Note: use cases for this are uncommon!

Example:

```
    b = 23.0;
    . . . . .
#pragma omp parallel firstprivate(b), private(i,myid)
{
    myid = omp_get_thread_num();
    for (i=0; i<n; i++){
        b += c[myid][i];
    }
    c[myid][n] = b;
}
```

- A *reduction* produces a single value from associative operations such as addition, multiplication, max, min, and, or.
- Would like each thread to reduce into a private copy, then reduce all these to give final result.
- Use REDUCTION clause:

Fortran: **REDUCTION** (*op: list*)

C/C++: **reduction** (*op: list*)

- Can have reduction arrays in Fortran, but not in C/C++

Reductions (cont.)

Example:

Value in original variable is saved

```
b = 10
```

Each thread gets a private copy of **b**, initialised to 0

```
!$OMP PARALLEL REDUCTION(+:b) ,
```

```
!$OMP& PRIVATE(I,MYID)
```

```
myid = omp_get_thread_num() + 1
```

```
do i = 1,n
```

All accesses inside the parallel region are to the private copies

```
  b = b + c(i,myid)
```

```
end do
```

```
!$OMP END PARALLEL
```

At the end of the parallel region, all the private copies are added into the original variable

```
a = b
```

Area of the Mandelbrot set

- Aim: introduction to using parallel regions.
- Estimate the area of the Mandelbrot set by Monte Carlo sampling.
 - Generate a grid of complex numbers in a box surrounding the set
 - Test each number to see if it is in the set or not.
 - Ratio of points inside to total number of points gives an estimate of the area.
 - Testing of points is independent - parallelise with a parallel region!

