Shared Memory Programming

Work sharing directives

Work sharing directives

- Directives which appear inside a parallel region and indicate how work should be shared out between threads
  - Parallel do/for loops
  - Parallel sections
  - Fortran 90 array syntax
  - ‘One thread only’ directives

Parallel do loops

- Loops are the most common source of parallelism in most codes. Parallel loop directives are therefore very important!
- A parallel do/for loop divides up the iterations of the loop between threads.
- There is a synchronisation point at the end of the loop: all threads must finish their iterations before any thread can proceed
- We will just introduce the basic form here: more details will follow in the next session.

Parallel do/for loops (cont)

Syntax:

Fortran:

```
         !$OMP DO  [clauses]
             do loop
       [ !$OMP END DO  ]
```

C/C++:

```
         #pragma omp for  [clauses]
             for loop
```

Restrictions in C/C++

- Because the for loop in C is a general while loop, there are restrictions on the form it can take.

- It has to have determinable trip count - it must be of the form:

  `for (var = a; var` *logical-op* **`b;`** *incr-exp***`)`**

  where *logical-op* is one of **`<, <=, >, >=`**

  and *incr-exp* is **`var = var +/- incr`** or semantic equivalents such as **`var++`**.

  Also cannot modify **`var`** within the loop body.

Parallel do/for loops (cont)

- With no additional clauses, the DO/FOR directive will partition the iterations as equally as possible between the threads.

- However, this is implementation dependent, and there is still some ambiguity:

e.g. 7 iterations, 3 threads. Could partition as 3+3+1 or 3+2+2

Parallel do/for loops (cont)

- How can you tell if a loop is parallel or not?
- Useful test: if the loop gives the same answers if it is run in reverse order, then it is almost certainly parallel
- Jumps out of the loop are not permitted.

e.g.

```
do i=2,n
   a(i)=2*a(i-1)
end do
```

✗

Parallel do/for loops (cont)

2.

```
ix = base
do i=1,n
   a(ix) = a(ix)*b(i)
   ix  = ix + stride
end do
```

✗

3.

```
do i=1,n
    b(i)= (a(i)-a(i-1))*0.5
end do
```

✓

Parallel do loops (example)

Example:

```
!$OMP PARALLEL
!$OMP DO
      do i=1,n
         b(i) = (a(i)-a(i-1))*0.5
      end do
!$OMP END DO
!$OMP END PARALLEL
```

Parallel DO/FOR directive

- This construct is so common that there is a shorthand form which combines parallel region and DO/FOR directives:

Fortran:

```
  !$OMP PARALLEL DO [clauses]
     do loop
[ !$OMP END PARALLEL DO ]
```

C/C++:

```
#pragma omp parallel for [clauses]
     for loop
```

Clauses

- DO/FOR directive can take PRIVATE and FIRSTPRIVATE clauses which refer to the scope of the loop.

- Other clauses will be discussed in the next session.

- Note that the parallel loop index variable is PRIVATE by default
  - other loop indices are private by default in Fortran, but not in C.

- PARALLEL DO/FOR directive can take all clauses available for PARALLEL directive.

Parallel sections

- Allows separate blocks of code to be executed in parallel (e.g. several independent subroutines)
- There is a synchronisation point at the end of the blocks: all threads must finish their blocks before any thread can proceed
- Not scalable: the source code determines the amount of parallelism available.
- Rarely used, except with nested parallelism - see later!

Parallel sections (cont)

Syntax:

Fortran:

```
  !$OMP SECTIONS [clauses]
[ !$OMP SECTION ]
        block
[ !$OMP SECTION
        block ]
     .  .  .
 !$OMP END SECTIONS
```

Parallel sections (cont)

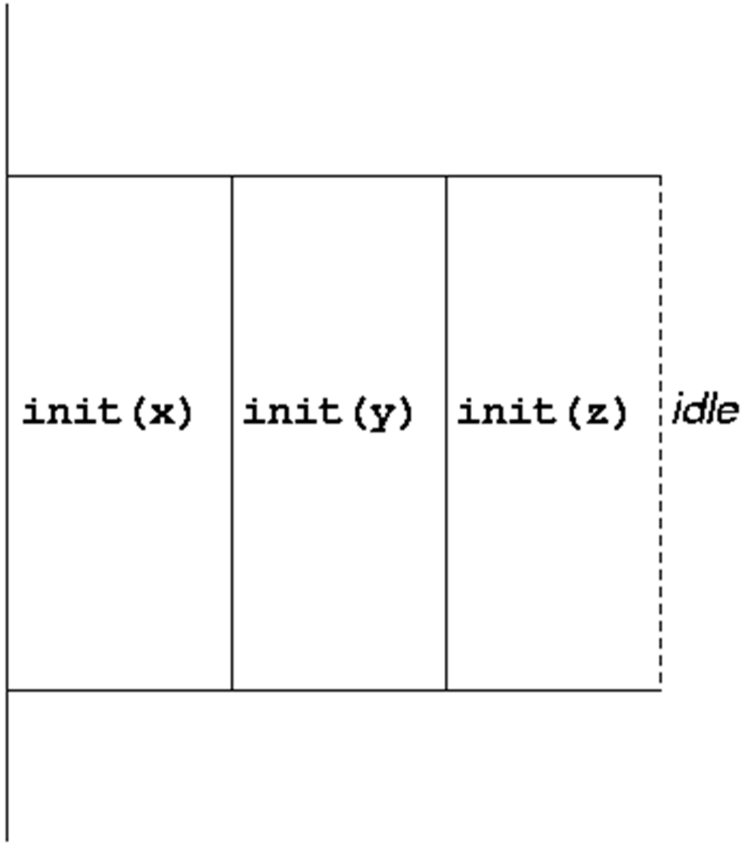
C/C++:

```
#pragma omp sections [clauses]
{
 [ #pragma omp section ]
        structured-block
 [ #pragma omp section
        structured-block
    . . . ]
 }
```

Parallel sections (cont)

Example:

```
!$OMP PARALLEL
!$OMP SECTIONS
!$OMP SECTION
      call init(x)
!$OMP SECTION
      call init(y)
!$OMP SECTION
      call init(z)
!$OMP END SECTIONS
!$OMP END PARALLEL
```



Parallel sections (cont)

- SECTIONS directive can take PRIVATE, FIRSTPRIVATE, LASTPRIVATE (see later) and clauses.

- Each section must contain a structured block: cannot branch into or out of a section.

Parallel section (cont)

Shorthand form:

Fortran:

```
!$OMP PARALLEL SECTIONS [clauses]

. . .

!$OMP END PARALLEL SECTIONS
```

C/C++:

```
#pragma omp parallel sections [clauses]
{
 . . .
 }
```

Workshare directive

- A worksharing directive (!) which allows parallelisation of Fortran 90 array operations, WHERE and FORALL constructs.

- Syntax:

```
!$OMP WORKSHARE
             block
!$OMP END WORKSHARE [NOWAIT]
```

Workshare directive (cont.)

- Simple example

```
REAL A(100,200), B(100,200), C(100,200)
...
!$OMP PARALLEL
!$OMP WORKSHARE
       A=B+C
!$OMP END WORKSHARE
!$OMP END PARALLEL
```

- N.B. No schedule clause: distribution of work units to threads is entirely up to the compiler!

- There is a synchronisation point at the end of the workshare: all threads must finish their work before any thread can proceed

Workshare directive (cont.)

- Can also contain array intrinsic functions, WHERE and FORALL constructs, scalar assignment to shared variables, ATOMIC and CRITICAL directives.
- No branches in or out of block.
- No function calls except array intrinsics and those declared ELEMENTAL.
- Combined directive:

```
!$OMP PARALLEL WORKSHARE
             block
!$OMP END PARALLEL WORKSHARE
```

Workshare directive (cont.)

- Example:

```
!$OMP PARALLEL WORKSHARE
      A = B + C
      WHERE (D .ne. 0) E = 1/D
!$OMP ATOMIC
      t = t + SUM(F)
      FORALL (i=1:n, X(i)=0) X(i)= 1
!$OMP END PARALLEL WORKSHARE
```

SINGLE directive

- Indicates that a block of code is to be executed by a single thread only.
- The first thread to reach the SINGLE directive will execute the block
- There is a synchronisation point at the end of the block: all the other threads wait until block has been executed.

SINGLE directive (cont)

Syntax:

Fortran:

```
!$OMP SINGLE [clauses]
      block
!$OMP END SINGLE
```

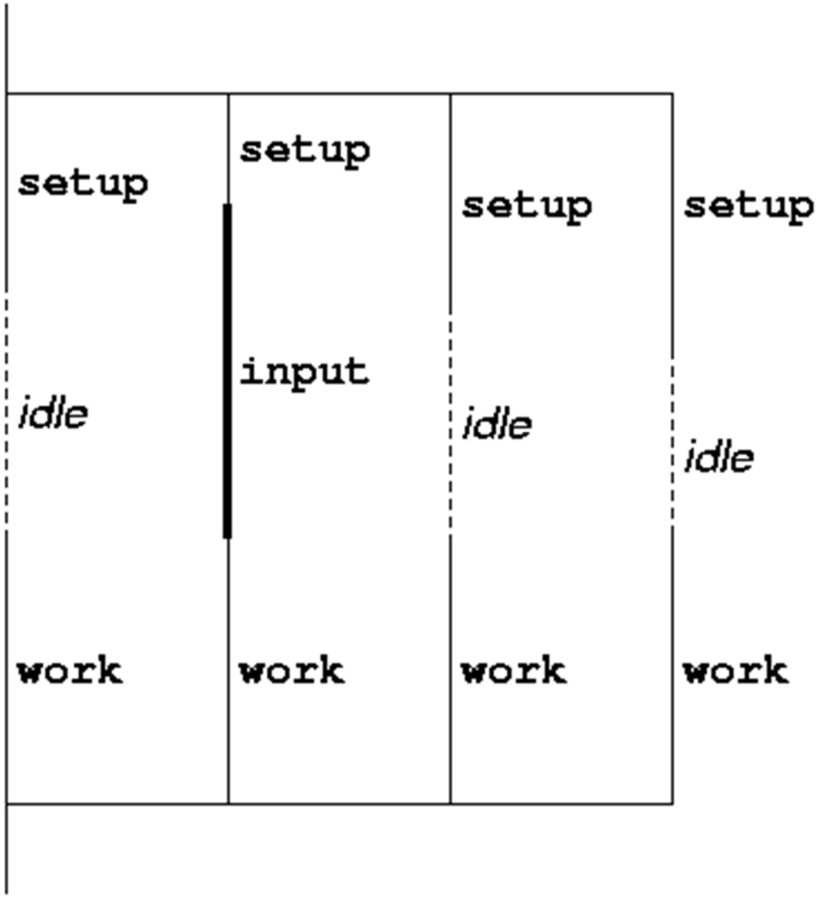
C/C++:

```
#pragma omp single [clauses]
      structured block
```

SINGLE directive (cont)

Example:

```
#pragma omp parallel
{
   setup(x);
#pragma omp single
  {
      input(y);
  }
   work(x,y);
}
```



SINGLE directive (cont)

- SINGLE directive can take PRIVATE and FIRSTPRIVATE clauses.
- Directive must contain a structured block: cannot branch into or out of it.

MASTER directive

- Indicates that a block of code should be executed by the master thread (thread 0) only.

- There is no synchronisation at the end of the block: other threads skip the block and continue executing: N.B. different from SINGLE in this respect. This generally means you need to combine it with a barrier

MASTER directive (cont)

Syntax:

Fortran:

```
!$OMP MASTER
        block
!$OMP END MASTER
```

C/C++:

```
#pragma omp master
        structured block
```

Practical session

Image processing

- Aim: Introduction to the use of parallel do/for loops.
- Simple image processing algorithm to reconstruct and image from an edge-detected version.
- Use PARALLEL DO/FOR directives to run it in parallel