Threaded Programming

Lecture 4: 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
  - Single directive
  - Master directive

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.

- The loop directive appears inside a parallel region and indicates that the work should be shared out between threads, instead of replicated

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

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 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
```

```
#pragma omp parallel

{

#pragma omp for

  for (int i=0;i<n;i++){

    b[i] = (a[i]*a[i-1])*0.5;

  }

}
```

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 , FIRSTPRIVATE and REDUCTION clauses which refer to the scope of the loop.
- 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.
- Beware! PARALLEL DO/FOR is not the same as DO/FOR or the same as PARALLEL

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

SCHEDULE clause

- The SCHEDULE clause gives a variety of options for specifying which loops iterations are executed by which thread.
- Syntax:

Fortran: `SCHEDULE (`*kind[, chunksize]*`)`

C/C++: `schedule (`*kind[, chunksize]*`)`

where *kind* is one of

`STATIC`, `DYNAMIC`, `GUIDED`, `AUTO` or `RUNTIME`

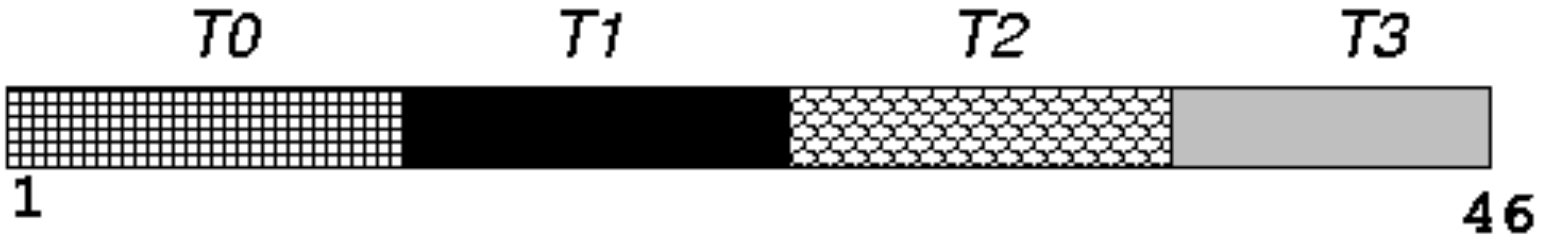
and *chunksize* is an integer expression with positive value.

- E.g. `!$OMP DO SCHEDULE(DYNAMIC,4)`

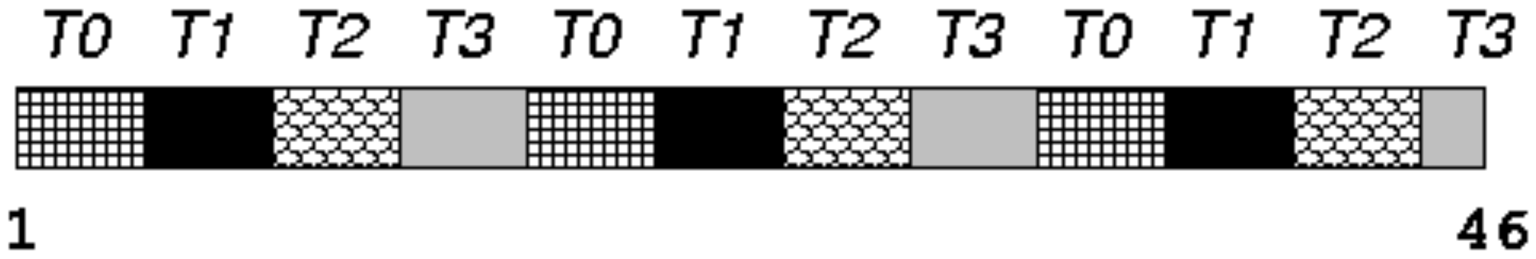
STATIC schedule

- With no *chunksize* specified, the iteration space is divided into (approximately) equal chunks, and one chunk is assigned to each thread in order (**block** schedule).
- If *chunksize* is specified, the iteration space is divided into chunks, each of *chunksize* iterations, and the chunks are assigned cyclically to each thread in order (**block cyclic** schedule)

STATIC schedule



SCHEDULE(STATIC)



SCHEDULE(STATIC,4)

DYNAMIC schedule

- DYNAMIC schedule divides the iteration space up into chunks of size *chunksize*, and assigns them to threads on a first-come-first-served basis.

- i.e. as a thread finish a chunk, it is assigned the next chunk in the list.

- When no *chunksize* is specified, it defaults to 1.

GUIDED schedule

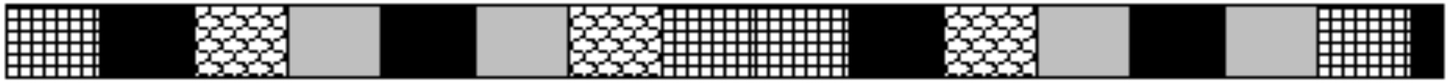
- GUIDED schedule is similar to DYNAMIC, but the chunks start off large and get smaller exponentially.

- The size of the next chunk is proportional to the number of remaining iterations divided by the number of threads.

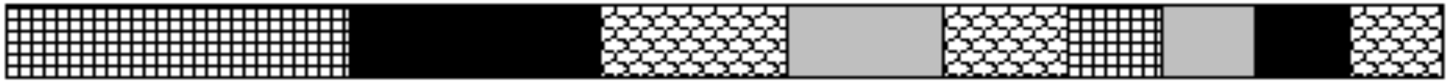
- The *chunksize* specifies the minimum size of the chunks.

- When no *chunksize* is specified it defaults to 1.

DYNAMIC and GUIDED schedules



SCHEDULE(DYNAMIC,3)



SCHEDULE(GUIDED,3)

AUTO schedule

- Lets the runtime have full freedom to choose its own assignment of iterations to threads
- If the parallel loop is executed many times, the runtime can evolve a good schedule which has good load balance and low overheads.

Choosing a schedule

When to use which schedule?

- STATIC best for load balanced loops - least overhead.
- STATIC,*n* good for loops with mild or smooth load imbalance, but can induce overheads.
- DYNAMIC useful if iterations have widely varying loads, but ruins data locality.
- GUIDED often less expensive than DYNAMIC, but beware of loops where the first iterations are the most expensive!
- AUTO may be useful if the loop is executed many times over

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
```

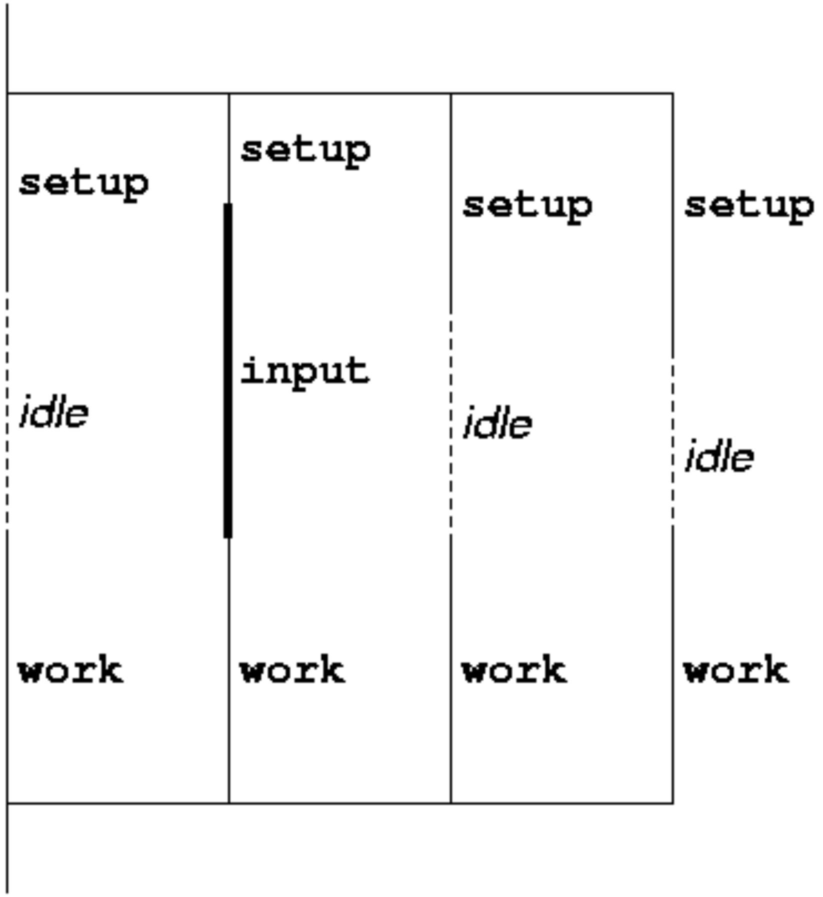
C/C++:

```
#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.

MASTER directive (cont)

Syntax:

Fortran:

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

C/C++:

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

Exercise

- Redo the Mandelbrot example using a worksharing do/for directive.