Shared Memory Programming with OpenMP

Work sharing directives

Work sharing directives

- Directives which appear inside a parallel region and indicate how work should be shared out between threads

  - Parallel for loops
  - Single directive
  - Master directive

Parallel for loops

- Loops are the most common source of parallelism in most codes. Parallel loop directives are therefore very important!

- A parallel 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:

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)

```
#pragma omp parallel
{
#pragma omp for
  for (int i=1;i<n;i++){
    b[i] = (a[i]*a[i-1])*0.5;
  }
}
```

Parallel for directive

- This construct is so common that there is a shorthand form which combines parallel region and worksharing loop directives:

C/C++:

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

Parallel loops (example)

```
#pragma omp parallel for
for (int i=1;i<n;i++){
    b[i] = (a[i]*a[i-1])*0.5;
}
```

Clauses

- **`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

- **`parallel for`** directive can take all clauses available for **`parallel`** directive.

- Beware! **`parallel for`** is not the same as **`for`** or the same as **`parallel`**

Parallel for loops (cont)

- With no additional clauses, the `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:

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.

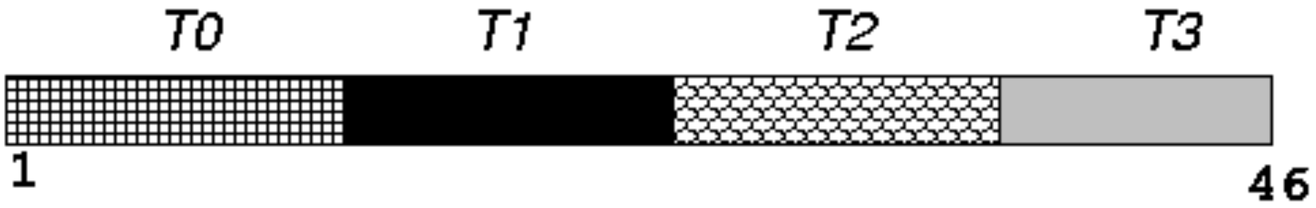
- e.g. **`#pragma omp for schedule(dynamic,4)`**

static schedule

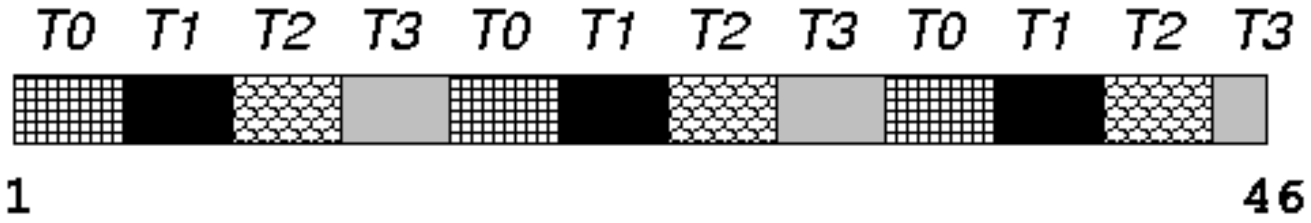
- With no *chunksize* specified, the iteration space is divided into (approximately) equal chunks, and one chunk is assigned to each thread in order (**block** schedule).

- 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.

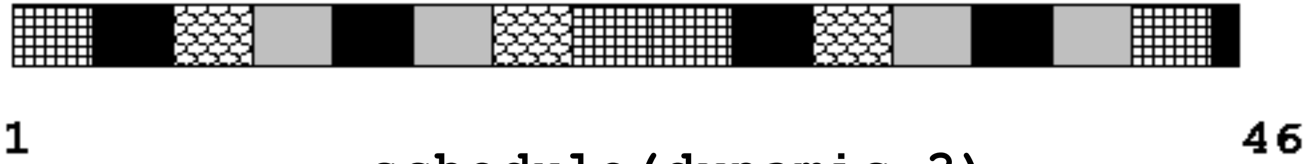
- i.e. as a thread finish a chunk, it is assigned the next chunk in the list.

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

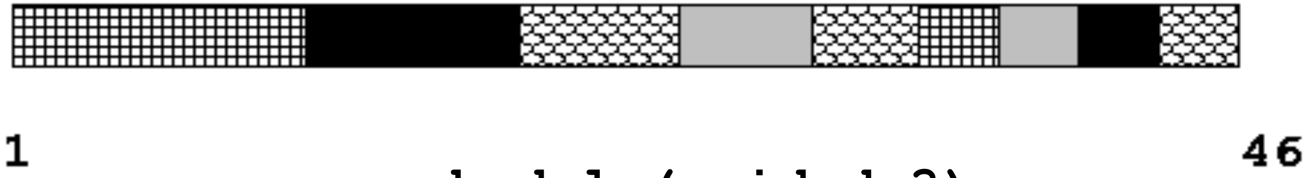
guided schedule

- `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.

runtime schedule

- Allows the schedule to be set using the environment variable **`OMP_SCHEDULE`**
  - e.g. **`export OMP_SCHEDULE="dynamic,1"`**
- Convenient for experimenting with schedules and chunksizes without having to recompile.

Choosing a schedule

When to use which schedule?

- **`static`** usually best for load balanced loops - least overhead.
- **`static,n`** good for loops with mild or smooth load imbalance, but can induce overheads for small chunksizes.
- **`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`** allows compiler-specific options

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:

C/C++:

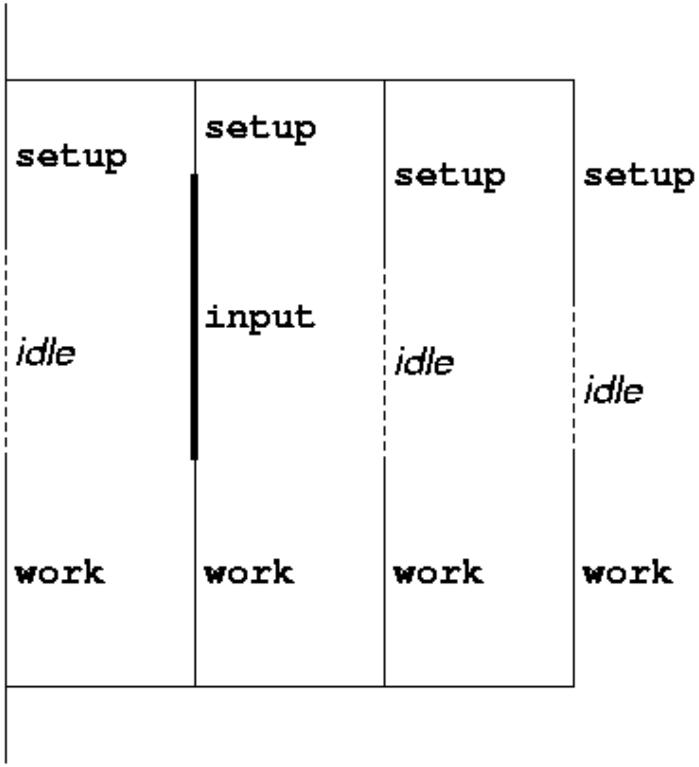
```
#pragma omp single [clauses]
        structured block
```

- Construct must contain a structured block: cannot branch into or out of it.

single directive (cont)

Example:

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



nowait clause

- The implicit barrier synchronization at the end of worksharing directive (`for` or `single`) can be removed by adding a `nowait` clause.
  - Use with care! Easy to introduce race conditions…

C/C++:

```
#pragma omp for nowait
   for loop

#pragma omp single nowait
   structured block
```

master directive

- Indicates that a block of code should be executed by the master thread (thread 0) only.
- Technically this isn’t a worksharing directive(!)
- 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.
- Latest versions of OpenMP have deprecated the name and replaced it with **`masked`**.

master directive (cont)

Syntax:

C/C++:

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

Orphaned directives

- Directives can be present in functions called from inside parallel regions

Example:

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

void fred() {
#pragma omp for
   for (int i=0; i<N; i++) {
      a[i] += 23.5;
   }
}
```

Orphaned directives (cont)

- This is very useful, as it allows a modular programming style….

- But it can also be rather confusing if the call tree is complicated (what happens if **fred** is also called from outside a parallel region? - the worksharing loop is all executed by the master thread)

- There are some extra rules about data scope attributes….

Data scoping rules

When we call a subroutine/function from inside a parallel region:

- Variables passed by reference/address in the argument list inherit their data scope attribute from the calling routine.
- Global variables in C/C++ are shared unless declared `threadprivate`
- `static` local variables in C/C++ are shared.
- All other local variables are private.

Exercise

- Redo the Mandelbrot example using a worksharing `for` directive.

Reusing this material

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

https://creativecommons.org/licenses/by-nc-sa/4.0/

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Acknowledge EPCC as follows: “© EPCC, The University of Edinburgh, www.epcc.ed.ac.uk”

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.