Threaded Programming

Lecture 3: Parallel Regions

epcc

THE UNIVERSITY OF EDINBURGH

Parallel region directive

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

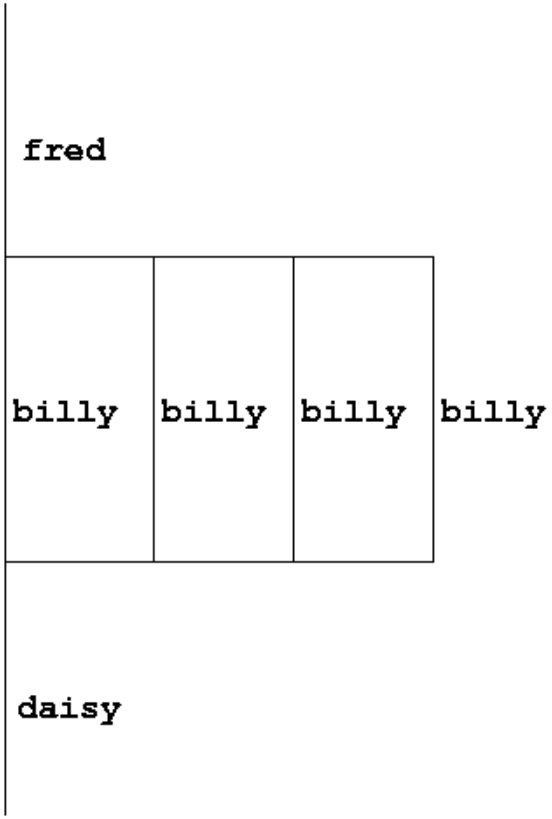
Fortran:
```
!$OMP PARALLEL
      block
!$OMP END PARALLEL
```

C/C++:
```
#pragma omp parallel
{
   block
}
```

Parallel region directive (cont)

Example:

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



Useful functions

- 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!

Useful functions (cont)

- 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

Clauses

- 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.

Shared and private variables

- 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)`

Shared and private (cont.)

- 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.
- In C++ private objects are created using the default constructor
- Not specifying a DEFAULT clause is the same as specifying DEFAULT(SHARED)
  - Danger!
  - Always use DEFAULT(NONE)

Reductions

- 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
- In C/C++, can use a special OpenMP syntax for array sections

Reductions (cont.)

Example:

```
      b = 10;
#pragma omp parallel reduction(+:b)
  {
      int myid = omp_get_thread_num();
      for (int i=0; i<n; i++){
         b += c[myid][i];
      }
  }
      a = b;
```

Value in original variable is saved

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

All accesses inside the parallel region are to the private copies

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

Reductions (cont.)

Example:

```
      b = 10
!$OMP PARALLEL REDUCTION(+:b),
!$OMP& PRIVATE(I,MYID)
      myid = omp_get_thread_num() + 1
      do i = 1,n
         b = b + c(i,myid)
      end do
!$OMP END PARALLEL
      a = b
```

Value in original variable is saved

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

All accesses inside the parallel region are to the private copies

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

ATOMIC directive

- Used to protect a single update to a shared scalar variable of basic type.
- Applies only to a single statement.
- Syntax:

Fortran: **!$OMP ATOMIC**

statement

where *statement* must have one of these forms:

x = *x op expr*, *x* = *expr op x*, *x* = *intr* (*x, expr*) or

x = *intr* (*expr, x*)

op is one of **+**, *****, **-**, **/**, **.and.**, **.or.**, **.eqv.**, or **.neqv.**

intr is one of **MAX**, **MIN**, **IAND**, **IOR** or **IEOR**

ATOMIC directive (cont)

C/C++: **`#pragma omp atomic`**
statement

where *statement* must have one of the forms:

x binop **=** *expr*, *x***++**, **++***x*, *x***−−**, or **−−***x*

and *binop* is one of **+**, *****, **−**, **/**, **&**, **^**, **<<**, or **>>**

- Note that the evaluation of *expr* is not atomic.
- May be more efficient than using CRITICAL directives, e.g. if different array elements can be protected separately.
- No interaction with CRITICAL directives

Critical sections

- A critical section is a block of code which can be executed by only one thread at a time.
- Can be used to protect updates to shared variables.

CRITICAL directive

- Syntax:

Fortran:
```
!$OMP CRITICAL
        block
!$OMP END CRITICAL
```

C/C++:
```
#pragma omp critical
        structured block
```

Exercise

Area of the Mandelbrot set

- Aim: introduction to using parallel regions.
- Estimate the area of the Mandelbrot set.
  - 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!