Introduction to OpenMP

Lecture 6: Further topics in OpenMP

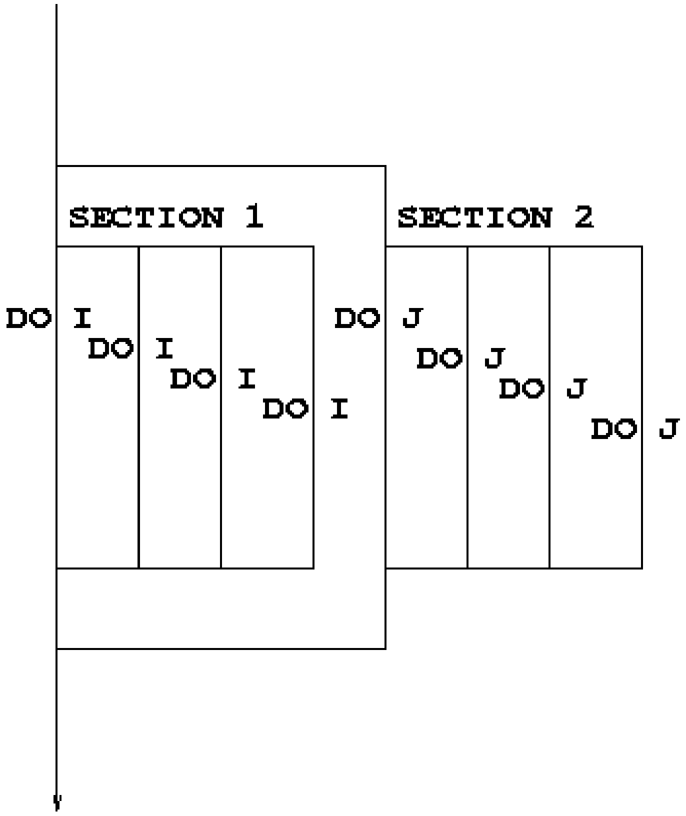
Nested parallelism

- Unlike most previous directive systems, nested parallelism is permitted in OpenMP.
- This is enabled with the **`OMP_NESTED`** environment variable or the **`OMP_SET_NESTED`** routine.
- If a PARALLEL directive is encountered within another PARALLEL directive, a new team of threads will be created.
- The new team will contain only one thread unless nested parallelism is enabled.

Nested parallelism (cont)

Example:

```
!$OMP PARALLEL
!$OMP SECTIONS
!$OMP SECTION
!$OMP PARALLEL DO
      do i = 1,n
         x(i) = 1.0
      end do
!$OMP SECTION
!$OMP PARALLEL DO
      do j = 1,n
         y(j) = 2.0
      end do
!$OMP END SECTIONS
!$OMP END PARALLEL
```



Nested parallelism (cont)

- Not often needed, but can be useful to exploit non-scalable parallelism (SECTIONS).

- Note: nested parallelism isn’t supported in some implementations (the code will execute, but as if OMP_NESTED is set to FALSE).
  - turns out to be hard to do correctly without impacting performance significantly.

NUMTHREADS clause

- One way to control the number of threads used at each level is with the NUM_THREADS clause:

```
!$OMP PARALLEL DO NUM_THREADS(4)
      DO I = 1,4
!$OMP PARALLEL DO NUM_THREADS(TOTALTHREADS/4)
         DO J = 1,N
            A(I,J) = B(I,J)
         END DO
      END DO
```

- The value set in the clause supersedes the value in the environment variable OMP_NUM_THREADS (or that set by **`omp_set_num_threads()`**)

Orphaned directives

- Directives are active in the *dynamic* scope of a parallel region, not just its *lexical* scope.
- Example:

```
!$OMP PARALLEL
      call fred()
!$OMP END PARALLEL

      subroutine fred()
!$OMP DO
      do i = 1,n
         a(i) = a(i) + 23.5
      end do
      return
      end
```

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

Data scoping rules

When we call a subroutine from inside a parallel region:

- Variables in the argument list inherit their data scope attribute from the calling routine.
- Global variables in C++ and COMMON blocks or module variables in Fortran are shared, unless declared THREADPRIVATE (see later).
- **`static`** local variables in C/C++ and **`SAVE`** variables in Fortran are shared.
- All other local variables are private.

Binding rules

- There could be ambiguity about which parallel region directives refer to, so we need a rule….

- DO/FOR, SECTIONS, SINGLE, MASTER and BARRIER directives always bind to the nearest enclosing PARALLEL directive.

Thread private global variables

- It can be convenient for each thread to have its own copy of variables with global scope (e.g. COMMON blocks and module data in Fortran, or file-scope and namespace-scope variables in C/C++).

- Outside parallel regions and in MASTER directives, accesses to these variables refer to the master thread's copy.

Thread private globals (cont)

Syntax:

Fortran: **!$OMP THREADPRIVATE (***list***)**

where list contains named common blocks (enclosed in slashes), module variables and SAVEd variables..

This directive must come after all the declarations for the common blocks or variables.

C/C++: **#pragma omp threadprivate (***list***)**

This directive must be at file or namespace scope, after all declarations of variables in *list* and before any references to variables in *list*. See standard document for other restrictions.

COPYIN clause

- Allows the values of the master thread's THREADPRIVATE data to be copied to all other threads at the start of a parallel region.

Syntax:

Fortran: **`COPYIN(`***list***`)`**

C/C++: **`copyin(`***list***`)`**

In Fortran the list can contain variables in THREADPRIVATE COMMON blocks.

COPYIN clause

Example:

```
      common /junk/ nx
      common /stuff/ a,b,c
!$OMP THREADPRIVATE (/JUNK/,/STUFF/)
      nx = 32
      c = 17.9
      . . .
!$OMP PARALLEL PRIVATE(NX2,CSQ) COPYIN(/JUNK/,C)
      nx2 = nx * 2
      csq = c*c
      . . .
```

Timing routines

OpenMP supports a portable timer:

- return current wall clock time (relative to arbitrary origin) with:

```
DOUBLE PRECISION FUNCTION OMP_GET_WTIME()
```

```
double omp_get_wtime(void);
```

- return clock precision with

```
DOUBLE PRECISION FUNCTION OMP_GET_WTICK()
```

```
double omp_get_wtick(void);
```

Using timers

```
DOUBLE PRECISION STARTTIME, TIME

STARTTIME = OMP_GET_WTIME()
......(work to be timed)
TIME = OMP_GET_WTIME()- STARTTIME
```

Note: timers are local to a thread: must make both calls on the same thread.

Also note: no guarantees about resolution!

Exercise

Molecular dynamics again

- Aim: use of orphaned directives.
- Modify the molecular dynamics code so by placing a parallel region directive around the iteration loop in the main program, and making all code within this sequential except for the forces loop.
- Modify the code further so that each thread accumulates the forces into a local copy of the force array, and reduce these copies into the main array at the end of the loop.