OpenMP

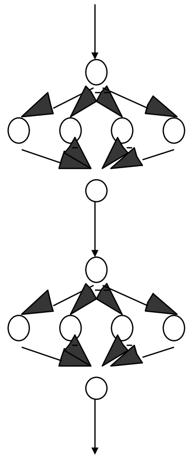
Steve Chapin

CSE 791

Fall 1999

What is OpenMP?

- A standard API for multithreaded programs (cf. MPI for message passing)
- C/C++ and Fortran definitions
- Uses fork-join model of parallelism



OpenMP Model

- One “master” thread enters parallel section
- Spawns “team” of threads
- All threads synchronize (barrier) at end
- Inside parallel section, program can use **directives** to guide compiler for OpenMP
- RTL allows control of parallel execution

OpenMP Components

- Directives
  - tell the compiler things
- Run-time library functions
  - control/query parallel environment
  - synchronize access to data
- Environment variables
  - control execution of parallel code

C/C++ Conditional Compilation

- _OPENMP macro name is defined to be the year/month of approved spec (yyyymm)

```
#ifdef _OPENMP
iam = omp_get_thread_num() + index;
#endif
```


Directives

- #pragma omp *directive-name [clauses]*
  - e.g. #pragma omp parallel
- Clauses
  - if (*scalar-expression*)
    - boolean guardian on directive
  - private(*list*), firstprivate(*list*), lastprivate(*list*)
    - variables in list are private to each thread in team

Clauses (cont.)

- shared(*list*)
  - causes variables in list to be shared among all threads in the team
- default(*shared* | *none*)
  - make each visible variable shared
- copyin(*list*)
  - initializes private variables

Clauses (III)

- reduction(*operator: list*)
  - specifies how to combine variables at end of parallel region
- schedule(kind[, chunk_size])
  - indicates how to divide loop iterations across the team (static, guided, dynamic, runtime)
  - chunk_size indicates how many iterations each thread gets

Work-sharing Constructs

- #pragma omp for [clauses]
  - specifies parallel for loop
- #pragma omp sections [clauses]
  - specifies non-iterative set of constructs that should be run in parallel
- #pragma simple [clauses]
  - only one thread does this
- Can combine with parallel directive
  - e.g. #pragma omp parallel for ...

Synchronization Constructs

- #pragma omp master
  - only the master thread does this
- #pragma omp critical [name]
  - defines critical section (by name)
  - all unnamed CS are considered the same
- #pragma omp barrier
  - waits for all threads to reach this spot
- #pragma omp atomic
  - ensures that expression is executed atomically

RTL Functions

- Execution environment functions
- Lock functions
- Some can only be called from serial portions of program
- Some can only be called from parallel portions of program
- Some can be called from either

Exec. Environment Functions

- omp_get_num_threads()
  - returns # of threads in current region
- omp_set_num_threads()
  - set # of threads for future regions (serial only)
- omp_get_max_threads()
  - maximum # that can come back from g_n_t
- omp_get_num_procs()
  - return maximum # of processors for prog

Exec. Env. Functions (cont.)

- omp_in_parallel()
  - returns true if in parallel region
- omp_set_dynamic()
  - enable/disable adjustment to # of threads
- omp_get_dynamic()
  - return true if dynamic enabled
- omp_set_nested()
  - enable/disable nested parallelism

Exec. Env. Functions III

- omp_get_nested()
  - return true if nested enabled

Lock Functions

- Type **`omp_lock_t`** is a simple lock
  - mutex, as you know it
- type **`omp_nest_lock_t`** is a nestable lock
  - mutex that a thread can get more than once
  - don't block if you own it
  - has a nesting count

Lock Functions (cont.)

- omp_init_lock(), omp_init_nest_lock()
  - initialize the lock (unblocked [, count 0])
- omp_destroy_lock(), omp_destroy_nest_lock()
  - delete the lock
- omp_set_lock(), omp_set_nest_lock()
  - block until lock available, set lock
  - if nested lock & I hold it, increment count and don't block

Lock Functions III

- omp_unset_lock(), omp_unset_nest_lock()
  - release ownership of simple lock
  - decrement counter, release if 0 for nested
- omp_test_lock(), omp_test_nest_lock()
  - ask "what if" question, non-blocking
  - get lock if available
  - simple lock: true/false
  - nested lock: nesting count/0

Environment Variables

- OMP_SCHEDULE
  - sets run-time schedule type and chunk size
- OMP_NUM_THREADS
  - number of threads to use during execution
- OMP_DYNAMIC
  - enables/disables dynamic adjustment of number of threads
- OMP_NESTED
  - enables/disables nested parallelism

Example I: Simple For Loop

```
#pragma omp parallel for
  for (I = 1; I < n; I++) {
    b[I] = (a[I] + a[I-1]) / 2.0;
  }
```


Example II: Using Parallel Regions

```
#pragma omp parallel shared(x, npoints)\
               private(iam, np, ipoints)
{
  iam = omp_get_thread_num();
  np = omp_get_num_threads();
  ipoints = npoints / np;
  dowork(x, iam, ipoints);
}
```

Example III: Parallel Sections

```
#pragma omp parallel sections
{
    #pragma omp section
      xaxis();
    #pragma omp section
      yaxis();
    #pragma omp section
      zaxis();
}
```


Example IV: Using Single Directives

```
#pragma omp parallel
{
    #pragma omp single
        printf("beginning work1.\n");
    work1();
   #pragma omp single
        printf("done with work1.\n");
}
```


Example V: Fixed Number of Threads

```
omp_set_dynamic(0);
omp_set_num_threads(16);
#pragma omp parallel
{
    if (omp_get_num_threads() != 16) abort ();
    /* rest of work that requires 16 threads…*/
}
```


Summary

- OpenMP defines a standard API for multithreaded programs
- Does not specify an implementation
- Several implementations around for Fortran (C/C++ ?)
- Can be combined with MPI for mixed-mode programming
- Combine with advisory library (cf. HOSS) to determine # of threads, etc.