CS 470 Spring 2017

Mike Lam, Professor

Advanced OpenMP

Atomics

- OpenMP provides access to highly-efficient hardware synchronization mechanisms
 - Use the **atomic** pragma to annotate a single statement
 - Statement must be a single increment/decrement or in the following form:
 - x <op>= <expr>; // <op> can be +, -, *, /, &, |, ^, <<, >>
 - Many processors provide a load/modify/store instruction
 - In x86-64, specified using the LOCK prefix
 - Far more efficient than using a mutex (i.e., critical)
 - This requires multiple function calls!

Locks

- OpenMP provides a basic locking system
 - Useful for protecting a data structure rather than a region of code
 - omp_lock_t: lock variable
 - Similar to pthread_mutex_t
 - omp_lock_init: initialize lock
 - Similar to pthread_mutex_init
 - omp_set_lock: acquire lock
 - Similar to pthread_mutex_lock
 - omp_unset_lock: release lock
 - Similar to pthread_mutex_unlock
 - omp_lock_destroy: clean up a lock
 - Similar to pthread_mutex_destroy

Thread safety

- Don't **mix** mutual exclusion mechanisms
 - #pragma omp critical
 - #pragma omp atomic
 - omp_set_lock()
- Don't nest mutual exclusion mechanisms
 - Nesting unnamed critical sections guarantees deadlock!
 - The thread cannot enter the second section because it is still in the first section, and unnamed sections "share" a name
 - If you must, use named critical sections or nested locks

Nested locks

- Simple vs. nested locks
 - omp_nest_lock_* instead of omp_lock_*
 - A nested lock may be acquired multiple times
 - Must be in the same thread
 - Must be released the same number of times
 - Allows you to write functions that call each other but need to acquire the same lock

Tasks

- OpenMP is most often used for data parallelism (parallel for)
- Newer versions (3.1+) have explicit task parallelism
 - #pragma omp parallel
 - Spawn worker threads
 - #pragma omp task
 - Create a new task (should be in a parallel block)
 - Task is assigned to an available worker by the runtime (may be deferred)
 - #pragma omp taskwait
 - Waits for all created tasks to finish (but doesn't destroy workers)

```
quicksort:
        <select pivot and partition>
```

main:

```
# pragma omp parallel
```

pragma omp single nowait
quick_sort(items, n);

// recursively sort each partition

- # pragma omp task
 quick_sort(items, p+1);
- # pragma omp task
 quick_sort(items+q, n-q);
- # pragma omp taskwait

Aside

- Often useful: multiple for-loops inside a parallel region
 - Many pragmas bind dynamically to any active parallel region
 - Less thread creation/joining overhead
 - Private variables can be re-used across multiple loops

```
# pragma omp parallel default(none) shared(n,m)
{
    int tid = omp_get_thread_num();
# pragma omp for
    for (int i = 0; i < n; i++) {
        // do something that requires tid
    }
# pragma omp for
    for (int j = 0; j < m; j++) {
        // do something else that requires tid
    }
}</pre>
```

Loop scheduling

- Use the schedule clause to control how parallel forloop iterations are allocated to threads
 - Modified by chunksize parameter
 - static: split into chunks before loop is executed
 - dynamic: split into chunks, dynamically allocated to threads (similar to thread pool or tasks)
 - guided: like dynamic, but chunk sizes decrease
 - The specified chunksize is the minimum
 - auto: allows the compiler or runtime to choose
 - runtime: allows specification using OMP_SCHEDULE

Loop scheduling

(static)

(static, 1)

Iteration 00 on thread 0 Iteration 01 on thread 0 Iteration 02 on thread 0 Iteration 03 on thread 0 Iteration 04 on thread 0 Iteration 05 on thread 0 Iteration 06 on thread 0 Iteration 07 on thread 0 Iteration 08 on thread 1 Iteration 09 on thread 1 Iteration 10 on thread 1 Iteration 11 on thread 1 Iteration 12 on thread 1 Iteration 13 on thread 1 Iteration 14 on thread 1 Iteration 15 on thread 1 Iteration 16 on thread 2 Iteration 17 on thread 2 Iteration 18 on thread 2 Iteration 19 on thread 2 Iteration 20 on thread 2 Iteration 21 on thread 2 Iteration 22 on thread 2 Iteration 23 on thread 2 Iteration 24 on thread 3 Iteration 25 on thread 3 Iteration 26 on thread 3 Iteration 27 on thread 3 Iteration 28 on thread 3 Iteration 29 on thread 3 Iteration 30 on thread 3 Iteration 31 on thread 3

Iteration 00 on thread 0 Iteration 01 on thread 1 Iteration 02 on thread 2 Iteration 03 on thread 3 Iteration 04 on thread 0 Iteration 05 on thread 1 Iteration 06 on thread 2 Iteration 07 on thread 3 Iteration 08 on thread 0 Iteration 09 on thread 1 Iteration 10 on thread 2 Iteration 11 on thread 3 Iteration 12 on thread 0 Iteration 13 on thread 1 Iteration 14 on thread 2 Iteration 15 on thread 3 Iteration 16 on thread 0 Iteration 17 on thread 1 Iteration 18 on thread 2 Iteration 19 on thread 3 Iteration 20 on thread 0 Iteration 21 on thread 1 Iteration 22 on thread 2 Iteration 23 on thread 3 Iteration 24 on thread 0 Iteration 25 on thread 1 Iteration 26 on thread 2 Iteration 27 on thread 3 Iteration 28 on thread 0 Iteration 29 on thread 1 Iteration 30 on thread 2 Iteration 31 on thread 3

(static, 2)

Iteration 00 on thread 0 Iteration 01 on thread 0 Iteration 02 on thread 1 Iteration 03 on thread 1 Iteration 04 on thread 2 Iteration 05 on thread 2 Iteration 06 on thread 3 Iteration 07 on thread 3 Iteration 08 on thread 0 Iteration 09 on thread 0 Iteration 10 on thread 1 Iteration 11 on thread 1 Iteration 12 on thread 2 Iteration 13 on thread 2 Iteration 14 on thread 3 Iteration 15 on thread 3 Iteration 16 on thread 0 Iteration 17 on thread 0 Iteration 18 on thread 1 Iteration 19 on thread 1 Iteration 20 on thread 2 Iteration 21 on thread 2 Iteration 22 on thread 3 Iteration 23 on thread 3 Iteration 24 on thread 0 Iteration 25 on thread 0 Iteration 26 on thread 1 Iteration 27 on thread 1 Iteration 28 on thread 2 Iteration 29 on thread 2 Iteration 30 on thread 3 Iteration 31 on thread 3

(dynamic, 2)

Iteration 00 on thread 1 Iteration 01 on thread 1 Iteration 02 on thread 3 Iteration 03 on thread 3 Iteration 04 on thread 2 Iteration 05 on thread 2 Iteration 06 on thread 0 Iteration 07 on thread 0 Iteration 08 on thread 3 Iteration 09 on thread 3 Iteration 10 on thread 3 Iteration 11 on thread 3 Iteration 12 on thread 3 Iteration 13 on thread 3 Iteration 14 on thread 3 Iteration 15 on thread 3 Iteration 16 on thread 2 Iteration 17 on thread 2 Iteration 18 on thread 3 Iteration 19 on thread 3 Iteration 20 on thread 2 Iteration 21 on thread 2 Iteration 22 on thread 1 Iteration 23 on thread 1 Iteration 24 on thread 3 Iteration 25 on thread 3 Iteration 26 on thread 1 Iteration 27 on thread 1 Iteration 28 on thread 1 Iteration 29 on thread 1 Iteration 30 on thread 0 Iteration 31 on thread 0

(guided)

Iteration 00 on thread 2 Iteration 01 on thread 2 Iteration 02 on thread 2 Iteration 03 on thread 2 Iteration 04 on thread 2 Iteration 05 on thread 2 Iteration 06 on thread 2 Iteration 07 on thread 2 Iteration 08 on thread 0 Iteration 09 on thread 0 Iteration 10 on thread 0 Iteration 11 on thread 0 Iteration 12 on thread 0 Iteration 13 on thread 0 Iteration 14 on thread 1 Iteration 15 on thread 1 Iteration 16 on thread 1 Iteration 17 on thread 1 Iteration 18 on thread 1 Iteration 19 on thread 3 Iteration 20 on thread 3 Iteration 21 on thread 3 Iteration 22 on thread 3 Iteration 23 on thread 2 Iteration 24 on thread 2 Iteration 25 on thread 2 Iteration 26 on thread 2 Iteration 27 on thread 2 Iteration 28 on thread 2 Iteration 29 on thread 1 Iteration 30 on thread 1 Iteration 31 on thread 3