

# **CS 292**

# **Introduction to Parallel Computing**

**Spring 2008**

VANDERBILT



School of Engineering

# Announcements

- Programming assignment #1
  - Two programs using Pthreads
  - Due Friday, 2/29/08
- Questions??

# Chapter 7 Topic Overview

Shared memory programming:

- Thread Basics ✓
- The POSIX Thread API ✓
- Synchronization Primitives in Pthreads ✓
- Controlling Thread and Synchronization Attributes ✓
- Composite Synchronization Constructs ✓
- OpenMP: a Standard for Directive Based Parallel Programming ← wrapping up today

# Synchronization Constructs in OpenMP

- OpenMP provides a variety of synchronization constructs.
- These constructs provide:
  - Synchronization points
  - Single threaded execution
  - Critical regions
- These are provided via high-level, easy-to-use APIs.

# Synchronization Constructs in OpenMP

- As seen in the Pthreads discussion, barriers are a synchronization point which hold threads until all threads have reached that point.
  - A frequently used synchronization primitive
- The OpenMP directive is:  

```
#pragma omp barrier
```
- Use barriers when data is update asynchronously and data integrity is at risk
  - Between sections of code that read/write the same variables or arrays
  - After each timestep in an iterative solver

# Synchronization Constructs in OpenMP

- In Pthreads, we used locks to protect critical sections.
  - These were necessary to prevent race conditions updating a shared variable
- OpenMP provides a corresponding directive.
- The OpenMP directive is:

```
#pragma omp critical [(name)]  
    structured block
```
- Only one thread will execute the block at a time.
- The block must be structured; no jumps in or out.
- Recall that critical sections serialize that block of code.
  - Will affect scalability and overall performance.

# Synchronization Constructs in OpenMP

- Critical sections are commonly used to protect the update of a single variable.
- OpenMP provides a special directive for this case:  
`#pragma omp atomic`  
`<single update statement>`
- This is a special, lightweight form of a critical section
- The form of the update statement is limited to

`x <binop>= exp`      `x--`

`x++`                      `--x`

`++x`

# Synchronization Constructs in OpenMP

- Sometimes a parallel section will contain a piece of code that needs to be performed by just one thread.
- OpenMP provides two directives for this case:

```
#pragma omp single [clause list]
    structured block

#pragma omp master
    structured block
```
- The `single` directive causes the block to be executed by a single (arbitrary) thread.
  - There is an implied barrier at the end of the block (unless the `nowait` clause is used), so it generates a scalar region
  - Useful for computing global data, initialization, or performing I/O
- The `master` directive specifies that the master thread does the execution (and no implied barrier)

# Memory Consistency

- OpenMP provides a mechanism to ensure all threads have a consistent view of objects in memory:  
`#pragma omp flush [(list)]`
- Generates a memory fence that causes all shared data to be written to or read from memory
  - All write operations must be committed to memory
  - All read operations must be satisfied from memory
  - Does not affect local/private data
- Several OpenMP directives have an implied flush, including a `barrier`, `exit` of `critical` sections, and `end of parallel`, `for`, and `sections` blocks

# Data Handling

- How threads manipulate data is an important factor affecting performance
  - Make data items `private` if it is safe for each thread to initialize and use their own copy.
  - Make data items `firstprivate` if they are read often and are initialized prior to the parallel region.
  - If multiple threads manipulate a single item, see if they can be done locally and then the results combined via a `reduction`.
  - If multiple threads manipulate different parts of a large data structure, see if it can be broken into smaller pieces that can be made `private`.
  - If none of the above apply, it is likely the data item must be `shared`. Be sure to consider race conditions.

# OpenMP Library Functions

- In addition to directives, OpenMP also supports a number of functions that allow a programmer to control the execution of threaded programs:

```
#include <omp.h>
```

```
/* thread and processor count functions */
```

```
void omp_set_num_threads (int num_threads);
```

Sets the default number of threads for when not explicitly stated.

Requires that dynamic adjustment of threads is enabled.

```
int omp_get_num_threads ();
```

```
int omp_get_max_threads ();
```

```
int omp_get_thread_num ();
```

```
int omp_get_num_procs ();
```

```
int omp_in_parallel();
```

# OpenMP Library Functions

- These functions allow a programmer to set and monitor thread creation:

```
#include <omp.h>
```

```
/* controlling and monitoring thread creation */  
void omp_set_dynamic (int dynamic_threads_flag);  
int omp_get_dynamic ();  
void omp_set_nested (int nested_flag);  
int omp_get_nested ();
```

- As stated last lecture, nested parallelism is only allowed if that capability has been enabled.

# OpenMP Library Functions

- These functions allow a programmer to create explicit locks, rather than relying on other synchronization constructs provided by OpenMP:

```
#include <omp.h>

/* mutual exclusion */
void omp_init_lock (omp_lock_t *lock);
void omp_set_lock (omp_lock_t *lock);
void omp_unset_lock (omp_lock_t *lock);
int omp_test_lock (omp_lock_t *lock);
void omp_destroy_lock (omp_lock_t *lock);
```

- In addition, all lock routines also have a nested lock counterpart for recursive mutexes.

# Environment Variables in OpenMP

- `OMP_NUM_THREADS`: This environment variable specifies the default number of threads created upon entering a parallel region.
- `OMP_SET_DYNAMIC`: Determines if the number of threads can be dynamically changed.
- `OMP_NESTED`: Turns on nested parallelism.
- `OMP_SCHEDULE`: Scheduling of for-loops if the clause specifies runtime

# Explicit Threads versus Directive Based Programming

- Directives layered on top of threads facilitate a variety of thread-related tasks.
- A programmer is rid of the tasks of initializing attributes objects, setting up arguments to threads, partitioning iteration spaces, etc.
- There are some drawbacks to using directives as well.
- An artifact of explicit threading is that data exchange is more apparent. This helps in alleviating some of the overheads from data movement, false sharing, and contention.
- Explicit threading also provides a richer API in the form of condition waits, locks of different types, and increased flexibility for building composite synchronization operations.
- Finally, since explicit threading is used more widely than OpenMP, tools and support for Pthreads programs are easier to find.