TEXAS TECH UNIVERSITY

CS4379: Parallel and Concurrent Programming
CS5379: Parallel Processing

Lecture 17

Dr. Yong Chen

Associate Professor

Computer Science Department

Texas Tech University

Lecture Video

- Please view the lecture video either from Teams or from the below link:

- https://texastechuniversity.sharepoint.com/sites/CS4379-CS5379/Shared%20Documents/General/Lecture17.mp4

Course Info

- **Lecture Time**: TR, 12:30-1:50
- **Lecture Location**: ECE 217
- **Sessions**: CS4379-001, CS4379-002, CS5379-001, CS5379-D01
- **Instructor:** Yong Chen, Ph.D., Associate Professor
- **Email:** yong.chen@ttu.edu
- **Phone:** 806-834-0284
- **Office**: Engineering Center 315
- **Office Hours**: 2-4 p.m. on Wed., or by appointment
- **TA:** Mr. Ghazanfar Ali, Ghazanfar.Ali@ttu.edu
- **TA Office hours:** Tue. and Fri., 2-3 p.m., or by appointment
- **TA Office:** EC 201 A
- **More info:**
  - http://www.myweb.ttu.edu/yonchen
  - http://discl.cs.ttu.edu; http://cac.ttu.edu/; http://nsfcac.org

Announcements/Reminders

- Recorded lectures for the rest of semester – due to mixed sessions and distance section requires viewing lectures at a flexible time
- Remaining quizzes will be scheduled at class time and will be announced in advance
- Skype meeting link for my office hours (Wed. 2 – 4 p.m.)
  - https://meet.ttu.edu/yong.chen/60DC09T2
  - Join by Phone
    - +1 (806) 834-4888,, 64158017#
    - +1 (855) 834-4888 - Toll-free,, 64158017#
    - Conference ID: 64158017
- Please be safe and stay well

Outline

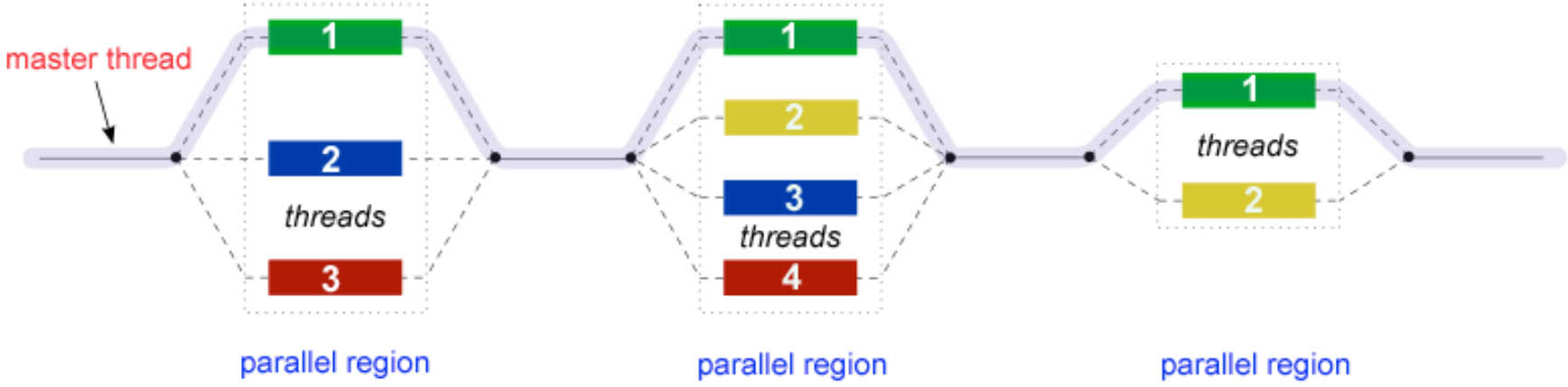
- Questions?
- OpenMP programming model
- Specifying concurrent tasks in OpenMP

OpenMP: a Standard for Directive Based Parallel Programming

- Pthreads still need explicit management of threads, low-level
- High-level constructs/directives desired
- OpenMP provides a directive-based API that can be used with FORTRAN, C, and C++ for programming shared address space machines.
  - OpenMP: Open Multi-Processing
- OpenMP directives provide support for concurrency, synchronization, and data handling while avoiding the need for explicitly setting up threads, distributing tasks, managing mutex locks, etc.

OpenMP programming model

- OpenMP uses the fork-join model of parallel execution:
  - Begin as a single thread: the master thread and executes sequentially until the first parallel region construct is encountered
  - FORK: the master thread then creates a team of parallel threads
    - The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the various team threads.
  - JOIN: When the team threads complete, they synchronize and terminate, leaving only the master thread.



OpenMP Programming Model

- OpenMP directives in C and C++ are based on the `#pragma` compiler directives

- A directive consists of a directive name followed by clauses.

```
#pragma omp directive [clause list]
```

- OpenMP programs execute serially until they encounter the `parallel` directive, which creates a group of threads.

```
#pragma omp parallel [clause list]
/* structured block */
```

- The main thread that encounters the `parallel` directive becomes the *master* of this group of threads and is assigned the thread id 0 within the group.

OpenMP Programming Model

- The clause list is used to specify conditional parallelization, number of threads, and data handling.
  - **Conditional Parallelization:** The clause `if (scalar expression)` determines whether the parallel construct results in creation of threads.
  - **Degree of Concurrency:** The clause `num_threads(integer expression)` specifies the number of threads that are created.

OpenMP Programming Model

- The clause list is used to specify conditional parallelization, number of threads, and data handling (cont.)
  - **Data Handling:** The clause `private (variable list)` indicates variables local to each thread.
  - The clause `firstprivate (variable list)` is similar to the `private`, except values of variables are initialized to corresponding values before the parallel directive.
  - The clause `shared (variable list)` indicates that variables are shared across all the threads.
  - The default state of a variable is specified by the clause `default`, e.g. `default(shared)`, or `default(private)`

OpenMP Programming Model

```
int a, b;
main() {
  // serial segment
  #pragma omp parallel num_threads (8) private (a) shared (b)
  {
    // parallel segment
  }
  // rest of serial segment
}
```

Sample OpenMP program

Code inserted by the OpenMP compiler

```
int a, b;
main() {
  // serial segment

  for (i = 0; i < 8; i++)
      pthread_create (......., internal_thread_fn_name, ...);

  for (i = 0; i < 8; i++)
      pthread_join (.......);

  // rest of serial segment

}

void *internal_thread_fn_name (void *packaged_argument) [
    int a;

  // parallel segment

}
```

Corresponding Pthreads translation

- A sample OpenMP program along with its Pthreads translation that might be performed by an OpenMP compiler.

OpenMP Programming Model

```
#pragma omp parallel if (is_parallel == 1) num_threads(8) \
   private (a) shared (b) firstprivate(c) {
 /* structured block */
}
```

- If the value of the variable `is_parallel` equals one, eight threads are created.
- Each of these threads gets private copies of variables a and c, and shares a single value of variable b.
- The value of each copy of c is initialized to the value of c before the parallel directive.

Reduction Clause in OpenMP

- The `reduction` clause specifies how multiple local copies of a variable at different threads are combined into a single copy at the master when threads exit.

- The usage of the `reduction` clause is `reduction (operator: variable list)`.
- The variables in the list are implicitly specified as being private to threads.
- The `operator` can be one of `+, *, -, &, |, ^, &&,` and `||`.

```
#pragma omp parallel reduction(+: sum) num_threads(8) {
/* compute local sums here */
}
/*sum here contains sum of all local instances of sums */
```

OpenMP Programming: Example

```
/* ******************************************************
An OpenMP version of a threaded program to compute Pi.
****************************************************** */
#pragma omp parallel default(private) shared (npoints) \
   reduction(+: sum) num_threads(8)
{
   num_threads = omp_get_num_threads();
   sample_points_per_thread = npoints / num_threads;
   sum = 0;
   for (i = 0; i < sample_points_per_thread; i++) {
      rand_no_x =(double)(rand_r(&seed))/(double)((2<<14)-
         1);
      rand_no_y =(double)(rand_r(&seed))/(double)((2<<14)-
         1);
      if ((rand_no_x * rand_no_x +
         rand_no_y * rand_no_y) < 1)
         sum++;
   }
}
```

Outline

- Questions?
- OpenMP programming model
- Specifying concurrent tasks in OpenMP

Specifying Concurrent Tasks in OpenMP

- The `parallel` directive can be used in conjunction with other directives to specify concurrency across iterations and tasks.
- OpenMP provides two directives: `for` and `sections` - to specify concurrent iterations and tasks.

Specifying Concurrent Tasks in OpenMP

- The `for` directive is used to split parallel iteration spaces across threads. The general form of a for directive is as follows:

```
#pragma omp for [clause list]
   /* for loop */
```

- The clauses that can be used in this context include: `private, firstprivate, reduction, schedule, nowait.`

Specifying Concurrent Tasks in OpenMP: for Directive Example (compute Pi)

```
#pragma omp parallel default(private) shared (npoints) \
   reduction(+: sum) num_threads(8)
{
   sum = 0;
   #pragma omp for
   for (i = 0; i < npoints; i++) {
      rand_no_x =(double)(rand_r(&seed))/(double)((2<<14)-1);
      rand_no_y =(double)(rand_r(&seed))/(double)((2<<14)-1);
      if ((rand_no_x * rand_no_x +
         rand_no_y * rand_no_y) < 1)
         sum++;
   }
}
```

Simple OpenMP programming: only two directives

Assigning Iterations to Threads

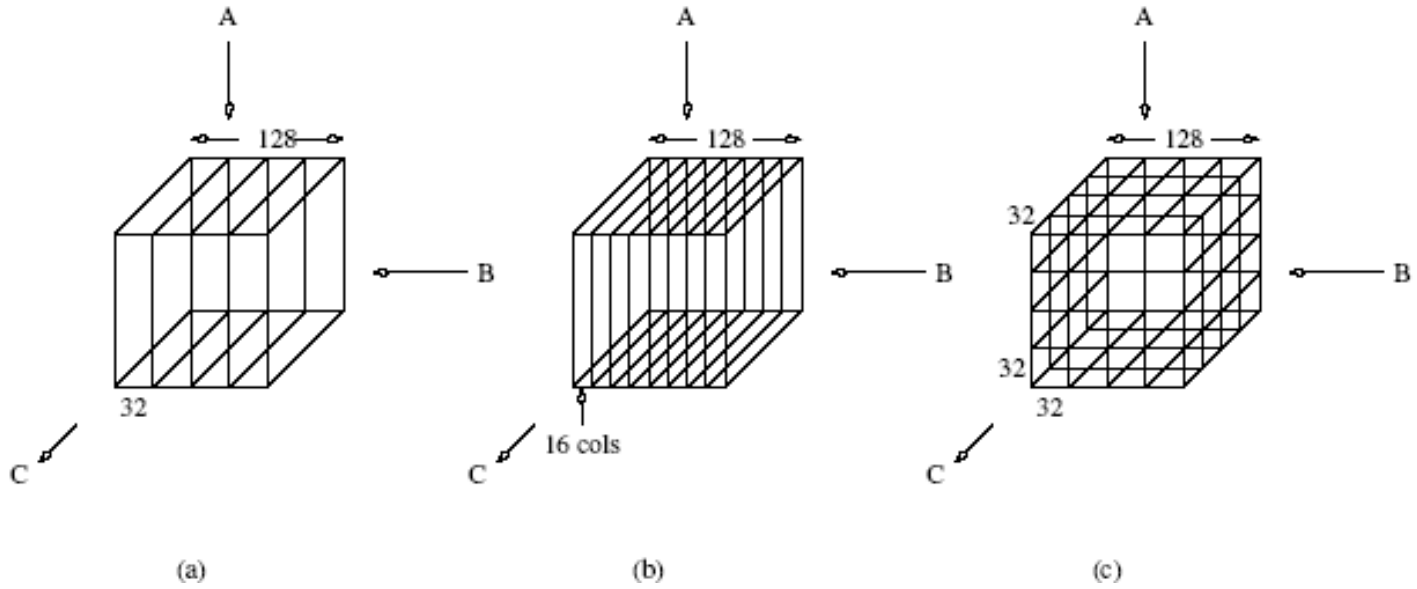
- The `schedule` clause of the `for` directive deals with the assignment of iterations to threads.
- The general form of the `schedule` directive is `schedule(scheduling_class[, parameter]).`
- OpenMP supports four scheduling classes: `static, dynamic, guided`, and `runtime`.

Assigning Iterations to Threads: Example

```
/* static scheduling of matrix multiplication loops */
#pragma omp parallel default(private) shared (a, b, c, dim) \
   num_threads(4)
   #pragma omp for schedule(static)
   for (i = 0; i < dim; i++) {
      for (j = 0; j < dim; j++) {
         c(i,j) = 0;
         for (k = 0; k < dim; k++) {
            c(i,j) += a(i, k) * b(k, j);
         }
      }
   }
```

`schedule(static[, chunk-size])` splits the iteration space into equal chunks of size chunk-size and assigns them to threads in a round-robin fashion

Assigning Iterations to Threads: Example



`schedule(static)` `schedule(static, 16)` Each for loop is parallelized and `schedule(static)`

- Three different schedules using the static scheduling class of OpenMP.

Scheduling Strategies

- Dynamic: assigned to threads when they become idle
  - `schedule(dynamic[, chunk-size]).`
  - Default to single iteration per chunk
- Guided: chunk size is dynamically changed toward the completion of the computation
  - To avoid load imbalancing and idling
- Runtime: delay scheduling decision until runtime, not in the code
  - Environment variable OMP_SCHEDULE determines the scheduling class and chunk size

Synchronization Across Multiple for Directives

- It is often desirable to have a sequence of `for`-directives within a parallel construct that do not execute an implicit barrier at the end of each `for` directive.

- OpenMP provides a clause - `nowait`, which can be used with a for directive.

Synchronization Across Multiple for Directives

```
#pragma omp parallel
{
  #pragma omp for nowait
    for (i = 0; i < nmax; i++)
      if (isEqual(name, current_list[i])
        processCurrentName(name);
  #pragma omp for
    for (i = 0; i < mmax; i++)
      if (isEqual(name, past_list[i])
        processPastName(name);
}
```

The `sections` Directive

- OpenMP supports non-iterative parallel task assignment using the `sections` directive.
- The general form of the `sections` directive is as follows:

```
#pragma omp sections [clause list]
{
   [#pragma omp section
     /* structured block */
   ]
   [#pragma omp section
     /* structured block */
   ]
   ...
}
```

The `sections` Directive: Example

```
#pragma omp parallel
{
   #pragma omp sections
   {
      #pragma omp section
      {
         taskA();
      }
      #pragma omp section
      {
         taskB();
      }
      #pragma omp section
      {
         taskC();
      }
   }
}
```

Merging Directives

- Without parallel directive specified, the for and sections directives would execute serially
- Consequently, `for` and `sections` directives are generally preceded by the parallel directive

- OpenMP allows the programmer to merge the parallel directives to `parallel for` and `parallel sections`, respectively
- The clause list for the merged directive can be from the clause lists of either the parallel or for / sections directives.

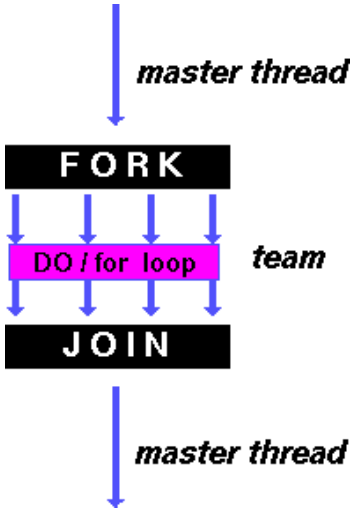
Merging Directives Example

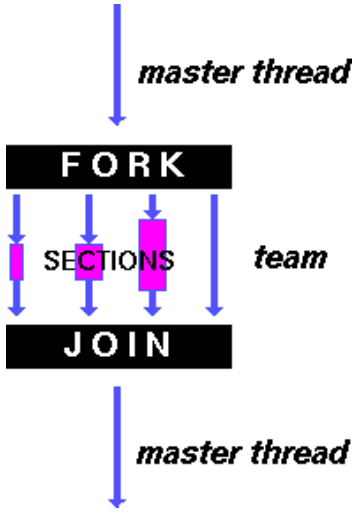
```
1    #pragma omp parallel default (private) shared (n)
2    {
3        #pragma omp for
4        for (i = 0 < i < n; i++) {
5        /* body of parallel for loop */
6        }
7    }
```

is identical to:

```
1    #pragma omp parallel for default (private) shared (n)
2    {
3        for (i = 0 < i < n; i++) {
4        /* body of parallel for loop */
5        }
6    }
```

Merging Directives





`parallel for`

`parallel sections`

Readings

- Reference book ITPC – Chapter 7, 7.10
- OpenMP Programming, by Blaise Barney, Lawrence Livermore National Laboratory: https://computing.llnl.gov/tutorials/openMP/
- OpenMP 5.0 Complete Specifications
  - https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf

Questions?

Questions/Suggestions/Comments are always welcome!

Write me: yong.chen@ttu.edu
Call me: 806-834-0284
See me: ENGCTR 315

If you write me an email for this class, please start the email subject with [CS4379] or [CS5379].