TEXAS TECH UNIVERSITY

CS4379: Parallel and Concurrent Programming
CS5379: Parallel Processing

Lecture 18

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/Lecture18.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

Outline

- Questions?

- Synchronization constructs in OpenMP
- OpenMP runtime library functions
- Environment variables
- Compiling, running, and demos

Synchronization Constructs in OpenMP

- OpenMP provides a variety of synchronization constructs:

```
#pragma omp barrier

#pragma omp single [clause list]
  structured block

#pragma omp master
  structured block

#pragma omp critical [(name)]
  structured block

#pragma omp atomic
  expression
```

Synchronization Point: The barrier Directive

- One of the most frequently used synchronization primitives

```
#pragma omp barrier
```

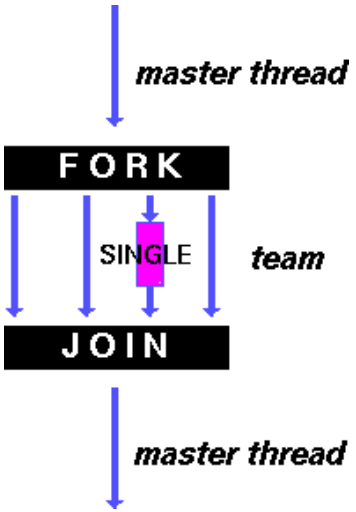
- On encountering this directive, all threads in a team wait until others have caught up, and then release
- When used with nested parallel directives, the barrier directive binds to the closest parallel directive

Single Thread Executions: The single and master Directives

- A single directive specifies a structured block that is executed by a single (arbitrary) thread.

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

- E.g. one thread compute the mean by dividing this global sum by the number of entries in the list



- The master directive is a specialization of the single directive in which only the master thread executes the structured block

```
#pragma omp master
        structured block
```

Critical Sections: The critical Directive

- OpenMP provides a critical directive for implementing critical regions

```
#pragma omp critical [(name)]
   structured block
```

- The use of name can identify a critical region and allows different threads to execute different critical regions

- OpenMP can also have explicit lock management

Critical Sections: The atomic Directive

- Often, a critical section consists simply of an update to a single memory location, e.g. incrementing or adding to an integer
- OpenMP provides another directive, atomic, for such atomic update to a memory location.
- Applied only to the single statement that immediately follows it

```
#pragma omp atomic
        count++;
```

- All atomic directives can be replaced by critical directives
- However, the availability of atomic hardware instructions may optimize the performance of the program

OpenMP Runtime Library Functions

- In addition to directives, the OpenMP standard defines an API for library calls that control the execution of threaded programs
  - Query the number of threads/processors, set number of threads to use
  - General purpose locking routines
  - Portable wall clock timing routines
  - Set execution environment functions: nested parallelism, dynamic adjustment of threads.

- It is necessary to specify the include file omp.h

The OpenMP specification is your best friend and reference.

Controlling Number of Threads and Processors

- Functions relate to the concurrency
- `void omp_set_num_threads(int num_threads);`
  - Sets the number of threads that will be used in the next parallel region (a positive integer)
  - This routine can only be called from the serial portions of the code
  - This call has precedence over the OMP_NUM_THREADS environment variable
- `int omp_get_num_threads();`
  - Returns the number of threads that are currently in the team executing the parallel region from which it is called.
  - If this call is made from a serial portion of the program, it will return 1

Example

- $ gcc -fopenmp omp_setnumthreads.c
- $ ./a.out

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

int main()
{
    printf("%d\n", omp_get_num_threads( ));
    omp_set_num_threads(4);
    printf("%d\n", omp_get_num_threads( ));
    #pragma omp parallel
        #pragma omp master
        {
            printf("%d\n", omp_get_num_threads( ));
        }

    printf("%d\n", omp_get_num_threads( ));

    #pragma omp parallel num_threads(3)
        #pragma omp master
        {
            printf("%d\n", omp_get_num_threads( ));
        }

    printf("%d\n", omp_get_num_threads( ));
}
```

Controlling Number of Threads and Processors

- `int omp_get_max_threads();`
  - Returns the maximum number of threads that could possibly be created
  - May be called from both serial and parallel regions of code
- `int omp_get_thread_num();`
  - Returns the thread number of the thread, within the team
  - This number will be between 0 and OMP_GET_NUM_THREADS-1. The master thread of the team is thread 0
  - If called from a nested parallel region, or a serial region, return 0
- `int omp_get_num_procs();`
  - Returns the number of processors that are available to the program.
- `int omp_in_parallel();`
  - Determine if the section of code which is executing is parallel (non-zero) or not (zero)

Controlling and monitoring thread creation

- `void omp_set_dynamic(int dynamic_threads);`
  - Enables or disables dynamic adjustment of the number of threads available for execution of parallel regions
  - If the value evaluates to zero, disabled, otherwise it is enabled
  - Has precedence over the OMP_DYNAMIC environment variable.
  - Must be called from a serial section of the program
- `int omp_get_dynamic();`
  - Query if dynamic thread adjustment is enabled (non-zero) or not
- `void omp_set_nested(int nested);`
  - Enable (non-zero) or disable (zero) nested parallelism
  - Has precedence over the OMP_NESTED environment variable
- `int omp_get_nested();`
  - Determine if nested parallelism is enabled or not.

Mutual Exclusion

- `void omp_init_lock(omp_lock_t *lock);`
  - Initialize lock before using it
  - The lock data structure in OpenMP is of type omp_lock_t
- `void omp_destroy_lock(omp_lock_t *lock);`
- `void omp_set_lock(omp_lock_t *lock);`
  - Forces the executing thread to wait until the specified lock is available
  - A thread is granted ownership of a lock when it becomes available
- `void omp_unset_lock(omp_lock_t *lock);`
- `int omp_test_lock(omp_lock_t *lock);`
  - Attempt to set a lock but does not block if the lock is unavailable
  - If the function returns a non-zero value, the lock has been successfully set, otherwise the lock is currently owned by another thread

Timing Routines

- Get wall-clock time: a double precision value equal to the number of seconds since the initial value of the OS real-time clock.
  - `double omp_get_wtime(void)`
  - Provides a portable wall clock timing routine
  - Usually used in "pairs" with the value of the first call subtracted from the value of the second call to obtain the elapsed seconds
- Get wall-clock time precision:
  - `double omp_get_wtick(void)`
  - Get timer precision
  - Returns a double-precision floating point value equal to the number of seconds between successive clock ticks

Environment Variables in OpenMP

- OMP_NUM_THREADS:
  - This environment variable specifies the default number of threads created upon entering a parallel region
  - csh: setenv OMP_NUM_THREADS 8
  - Bash: export OMP_NUM_THREADS=8
- OMP_DYNAMIC:
  - Enables or disables dynamic adjustment of the number of threads
  - csh: setenv OMP_DYNAMIC TRUE
- OMP_NESTED
  - Enables or disables nested parallelism
- OMP_SCHEDULE:
  - Applies only to for, parallel for directives which schedule set to RUNTIME
  - setenv OMP_SCHEDULE "dynamic”
- All environment variable names are uppercase. The values assigned to them are not case sensitive

Explicit Threads versus Directive Based Programming

- Directives layered on top of threads facilitate a variety of thread-related tasks.
- Ease the tasks of creating/joining threads, setting up arguments to threads, partitioning iteration spaces, etc.
- There are some drawbacks to using directives as well.
  - Explicit threading makes data exchange more apparent, alleviating some of the overheads from data movement.
  - 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.
  - Since explicit threading is used more widely than OpenMP, tools and support for Pthreads programs are easier to find.

Outline

- Questions?

- Synchronization Constructs in OpenMP
- OpenMP runtime library functions
- Environment variables
- Compiling, running, and demos

Compiling OpenMP programs

- Load GNU Compiler module
  - `$ module load gnu7/7.3.0`
- Compiling
  - `$ gcc –o hello-openmp –fopenmp hello-openmp.c`
- Automate compiling using Makefile
  - Edit a Makefile
  - `$ make`

Running OpenMP programs

- Create job submission script, e.g.

```
#!/bin/sh
#$ -V
#$ -cwd
#$ -S /bin/bash
#$ -N OpenMP_Test_Job
#$ -o $JOB_NAME.o$JOB_ID
#$ -e $JOB_NAME.e$JOB_ID
#$ -q omni
#$ -pe sm 36
#$ -l h_vmem=5.3G
#$ -l h_rt=48:00:00
#$ -P quanah

./hello-openmp
```

- Submit job
  - **qsub** <job submission script>
- Check job status
  - Command: **qstat**

Checking Output and Debugging Failed Jobs

- Job output
  - Standard: $JOB_NAME.o$JOB_ID
  - Error: $JOB_NAME.e$JOB_ID

- When debugging:
  - Check the output files for errors
  - Check the output of qacct –j <job_ID>
    - failed
    - exit_status
    - maxvmem
    - start_time & end_time (<runtime limit)
    - low

Demos

- Source code samples: Please checkout source code samples with executing the following command:

```
git clone
https://discl.cs.ttu.edu/gitlab/yongchen/cs4379
cs5379.git
```

- If you have already checked out a copy of the repo earlier, you can run the following command to update to the latest source code repo:

```
git pull
```

More Resources: OpenMP Standard

- Standard developed from 1997, with initial members of Intel, IBM, Compaq, HP, SGI, Sun Microsystems, DOE
- OpenMP website: openmp.org
  - API specifications, FAQ, presentations, discussions, media releases, calendar, membership application and more...
- OpenMP specifications
  - https://www.openmp.org/specifications/
- OpenMP 5.0 Complete Specifications
  - https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf
  - Useful reference
- Book: “Using OpenMP: Portable Shared Memory Parallel Programming”

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].