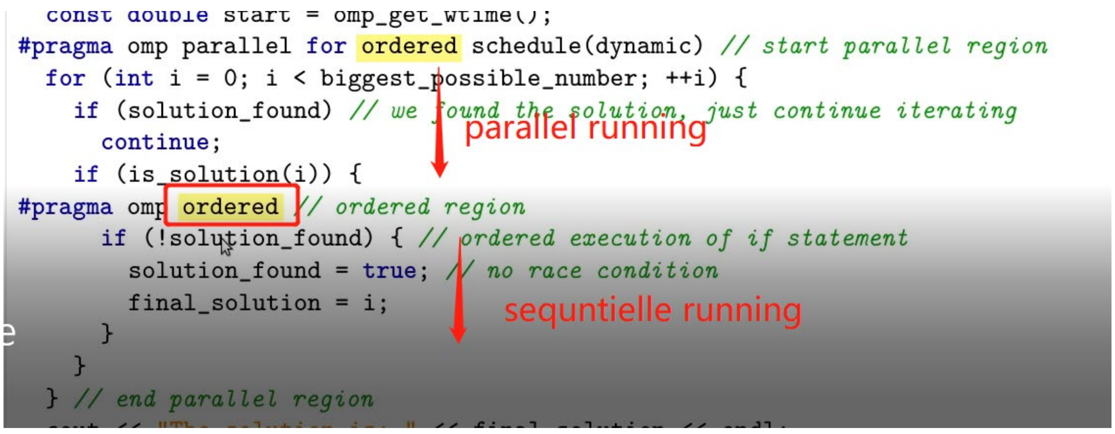
Exam Assignments V03

chengchengguo 183090

1. How does the **ordered clause** in OpenMP work in conjunction with a parallel for loop?

Different threads execute concurrently until they encounter the **ordered region,** which is then **executed sequentially** in the **same order as** it would get executed in a **serial loop**.



```
  const double start = omp_get_wtime();
#pragma omp parallel for ordered schedule(dynamic) // start parallel region
  for (int i = 0; i < biggest_possible_number; ++i) {
    if (solution_found) // we found the solution, just continue iterating
      continue;
    if (is_solution(i)) {
#pragma omp ordered // ordered region
      if (!solution_found) { // ordered execution of if statement
        solution_found = true; // no race condition
        final_solution = i;
      }
    }
  } // end parallel region
```

2. What is the **collapse clause** in OpenMP good for?

We can **parallelize nested for loops** with the collapse clause.
it is good for balancing the work of nested for loops.
if not use collapse , only use #pragma omp parallel for , only the outer for loop would run parallel

3. Explain how **reductions** work internally in OpenMP.

Reduction is an **associative** and **commutative operation.** It is used in parallel programming to **reduce many values** into **a single result.**

- A local copy of each ***list*** variable is made and initialized depending on the ***op*** (0 for **+**)
- Updates occur on the local copy
- Local copies are reduced into a single value and combined with the original global value

4. What is the purpose of a **barrier** in parallel computing?

A barrier means that **any thread must stop** at **this point** and **cannot proceed until all other threads reach this barrier.**
To avoid conflicting access to shared data, we use barrier to divide a program into phases, ensuring that shared data is mutated in a phase in which no other thread accesses it. A *barrier* divides a program into phases by requiring all threads to reach it before any of them can proceed. Code that is executed after a barrier cannot be concurrent with code executed before the barrier.

5. Explain the differences between the library routines

- omp_get_num_threads() *// number of threads*
  The **omp_get_num_threads** routine returns the number of threads in the team executing the parallel region to which the routine region binds. If called from the sequential part of a program, this routine returns 1.
- omp_get_num_procs() *// number of logical cores*
  e.g. A system with two E5420 Xeon's has 2 packages, 2 processors per package, 2 cores per processor, 0 hardware threads per core. omp_get_num_procs should return 8.[1]
- omp_get_max_threads(). *// maximum number of threads in a parallel region*
  *The value returned by **omp_get_max_threads** is the value of the first element of the* nthreads-var *ICV of the current task. This value is also an upper bound on the number of threads that could be used to form a new team if a parallel region without a num_threadsclause were encountered after execution returns from this routine.*[2]

6. Clarify how the storage attributes **private** and **firstprivate** differ from each other.

private *// create **uninitialized** copy of the variable for each thread*
firstprivate *// create **initialized** one-to-one copy of the variable for each thread*

The **private** clause declares the variables in the list to be private to each thread in a team. The **firstprivate** clause provides a superset of the functionality provided by the private clause. The private variable is initialized by the original value of the variable when the parallel construct is encountered.[3]

[1] OMP_GET_MAX_THREADS vs OMP_GET_NUM_PROCS - Intel Communities
[2] c++ - OpenMP omp_get_num_threads() V.S. omp_get_max_threads() - Stack Overflow
[3] Shared and private variables in a parallel environment - IBM Documentation

private
1. private variables are <u>undefined</u> on entry and exit of the parallel region.即
private
2. The value of the original variable (before the parallel region) is undefined after the parallel region!

3. A private variable within the parallel region has no storage association with the same variable outside of the region.

firstprivate
Firstprivate(list):All variables in the list are initialized with the value the original object had before entering the parallel construct.

7. Do the **coding warmup** on **slide 18**.

   Write in **<u>pseudo code</u>** how the **computation of** pi can be **parallelized** with **simple threads**.

```
fun thunk:
  sum_local = 0
  do i = thread_id; i < num_steps; i += num_threads
        calculate x = midpoint
        sum_local += new hight
  end do
  ---barrier---
  sum+= sum_local

fun main
    set num_steps, width of rectangle
    start the timer
            get amount of logical cores
            set a vector threads, threads.reserve(num_threads)
            run function thunk on each thread

            run function thunk on master thread
            join threads

            calculate pi : pi = sum * 4 * width
    finish the timer
```

```
num_points = 100000000; // amount of points
in_circle_count = 0
  ☟ (this part can run in parallel)
do i = 1→ num_points
    generate 2 random number between 0 and 1
    if (x, y) is in clrcle:
        in_circle_count+ =1
end do   //☜ end parallel part
pi = 4 * in_circle_count / num_points
```