Deliverable **Alejandro Gallego Rodriguez**

Par1203 **Oriol Catalan**

3.1 OpenMP Questionnaire

A) Parallel regions

1.hello.c

1. How many times will you see the "Hello world!" message if the program is executed with "./1.hello"?

- 24 times, one for each thread.

2. Without changing the program, how to make it to print 4 times the "Hello World!" message?

- Executing the following command: OMP_NUM_THREADS=4 ./1.hello

2.hello.c

1. Is the execution of the program correct? (i.e., prints a sequence of "(Thid) Hello (Thid) world!" being Thid the thread identifier). If not, add a data sharing clause to make it correct?

- The execution is not correct; we make it correct by adding the clause "private(id)"

2. Are the lines always printed in the same order? Why the messages sometimes appear intermixed? (Execute several times in order to see this).

- No, synchronization is random and a thread can finish earlier than another thread.

3.how_many.c

1. How many "Hello world ..." lines are printed on the screen?

- 20, 8 from the first parallel printf, 5 in the for loop, 4 in the third parallel printf and 3 in the last one.

2. What does omp get num threads return when invoked outside and inside a parallel region?

- Outside returns the number 1, inside returns the current number of threads declared.

4.data_sharing.c

1. Which is the value of variable x after the execution of each parallel region with different data-sharing attribute (shared, private, firstprivate and reduction)? Is that the value you would expect? (Execute several times if necessary)

- Shared: [109, 120], because of no synchronization and last iterations of the loop are not executed sometimes.

- Private: 5, because x is independent in each thread.

- Firstprivate: 5, same reason as private.

- Reduction: 125, a local variable is used for each thread and each variable of each thread is added and stored in the original variable at the end of the parallelism.

B) Loop Parallelism

1.schedule.c

1. Which iterations of the loops are executed by each thread for each schedule kind?

- Static: Each thread executes 3 iterations in order.

- Static, 2: Each thread executes 2, due to the chunk, iterations in order.

- Dynamic: Each thread executes 2 iterations when it the thread is free.

- Guided: Each thread executes 2 iterations minimum.

2.nowait.c

1. Which could be a possible sequence of printf when executing the program?

- Loop 1: thread (0) gets iteration 0

  Loop 1: thread (2) gets iteration 1

  Loop 2: thread (1) gets iteration 2

  Loop 2: thread (3) gets iteration 3

2. How does the sequence of printf change if the nowait clause is removed from the first for directive?

- Loop 1: thread (0) gets iteration 0

  Loop 1: thread (1) gets iteration 1

  Loop 2: thread (1) gets iteration 3

  Loop 2: thread (0) gets iteration 2

3. What would happen if dynamic is changed to static in the schedule in both loops? (keeping the nowait clause)

- Loop 1: thread (0) gets iteration 0

  Loop 1: thread (1) gets iteration 1

  Loop 2: thread (0) gets iteration 2

  Loop 2: thread (1) gets iteration 3

3.collapse.c

1. Which iterations of the loop are executed by each thread when the collapse clause is used?

- Each thread executes 3 iterations except for one that executes 4. The collapse clause merges 2 loops into 1 and divides the work into the threads.

2. Is the execution correct if the collapse clause is removed? Which clause (different than collapse) should be added to make it correct?

- The division of tasks is unbalanced; we can correct this with the “ordered” clause.

C) Synchronization

1.datarace.c

1. Is the program always executing correctly?

-No, when the threads execute big iterations, chunk is bigger than iterations remaining and they don't execute them.

2. Add two alternative directives to make it correct. Explain why they make the execution correct.

- One solution is with guided schedule with chunk size 1, this clause makes the size of every chunk proportional to the number of unassigned iterations divided by the number of threads. Because of that we do not need to worry about the size of the chunk being too big and missing iterations

- The other one is with static schedule with no chunk size, the iteration space is divided into chunks at the beginning of the execution.

2.barrier.c

1. Can you predict the sequence of messages in this program? Do threads exit from the barrier in any specific order?

- No, we can't predict it because the exit is random. The only messages we can predict is the ones entering in the barrier due to the milliseconds that are specified.

3.ordered.c

1. Can you explain the order in which the ”Outside” and ”Inside” messages are printed?

- The outside messages are printed in parallel, and the inside are sequential because of the ordered clause.

2. How can you ensure that a thread always executes two consecutive iterations in order during the execution of the ordered part of the loop body?

- Putting the ordered clause before the two printfs.

D) Tasks

1.single.c

1. Can you explain why all threads contribute to the execution of instances of the single worksharing construct? Why are those instances appear to be executed in bursts?

- Because of the “nowait” clause.

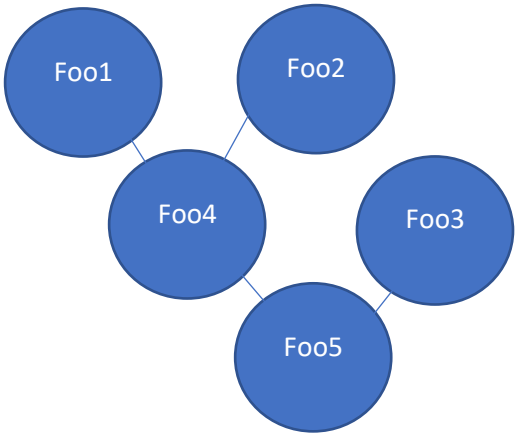
2.fibtasks.c

1. Why all tasks are created and executed by the same thread? In other words, why the program is not executing in parallel?

- There is no parallel clause.

3.synchtasks.c

1. Draw the task dependence graph that is specified in this program



2. Rewrite the program using only taskwait as task synchronisation mechanism (no depend clauses allowed)

```
int main(int argc, char *argv[]) {

   #pragma omp parallel
```

```
    #pragma omp single
    {
      printf("Creating task foo1\n");
      #pragma omp task
      foo1();
      printf("Creating task foo2\n");
      #pragma omp task
      foo2();
      printf("Creating task foo3\n");
      #pragma omp task
      foo3();
      printf("Creating task foo4\n");
      #pragma omp taskwait
      foo4();
      printf("Creating task foo5\n");
      #pragma omp taskwait
      foo5();
    }
    return 0;
}
```

4.taskloop.c

1. Find out how many tasks and how many iterations each task executes when using the grainsize and num tasks clause in a taskloop.

- Grainsize (5) = 12/5 tasks, 5 iterations per task

- Num_tasks (5) = 5 tasks, iterations per task 12/52.

2. What does occur if the nogroup clause in the first taskloop is uncommented?

- The taskloop construct creates an implicit taskgroup region, which waits until the threads finish to continue the execution. With the nogroup clause there is no creation of taskgroup and the execution does not wait until the threads are over.

3.2 Observing overheads

…