Lab 4:
OpenMP Part 1

Jan Izquierdo
jan.izquierdo@

Laura LLorente
laura.llorente@

17/05/2024

hello.c

1. *How many times will you see the "Hello world!" message if the program is executed with "./hello"?*
   The same number as the processors our machine contains (ours is set to 4 processors).

2. *Without changing the program, how to make it print 4 times the "Hello World!" message?*
   By using a machine with exactly 4 processors, like we are doing.

how_many.c
Assuming the OMP NUM THREADS variable is set to 8 with "export OMP NUM THREADS=8"

1. *How many "Hello world ..." lines are printed on the screen? Why?*
   At least 12, at most 15, it depends on the number of threads of the fifth parallel, which is determined by a random factor (**rand()%4+1**)

2. *Which mechanism overrides the number of threads already set by other mechanisms?*
   #**pragma omp parallel num_threads(3)**, as it sets its own number of threads

hello world.c : Assuming the OMP NUM THREADS variable is set to 8 with "export OMP NUM THREADS=8"

1. *Is the execution of the program correct? (i.e., prints a sequence of "(Thid) Hello (Thid) world!" being Thid the thread identifier) Which data sharing clause should be added to make it correct?*
   No, it should print:
   (0) Hello (0) world!
   (2) Hello (2) world!
   (1) Hello (1) world!
   (3) Hello (3) world!
   with the same number displaying the ID before the ‘hello’ and the ‘world’ (does not matter the order)
   Data sharing clause used : **#pragma omp parallel private(id)**

2. *Are the lines always printed in the same order? Could the messages appear intermixed?*
   Lines can be printed in different order since the output of different threads is being printed at the same time due to the concurrent execution of the threads.

data_sharing.c

1. *Which is the value of variable x after the execution of each parallel region with different data-sharing attributes (shared, private and first private)?*
   After the shared section x is 24
   After the private section x is 5
   After the first private section x is 71

2. *What needs to be changed/added/removed in the first directive to ensure that the value after the first parallel is always 8?*
   omp_set_num_threads (8)

parallel.c

1. *How many messages the program prints? Which iterations is each thread executing?*
   It prints 42 messages. Each thread is executing thread_id to N executions.

2. *Change the for loop to ensure that its iterations are distributed among all participating threads.*

```
    #pragma omp parallel for num_threads(NUM_THREADS)
    for (int i=0; i < N; i=i+1) {
            int id=omp_get_thread_num();
            printf("Thread ID %d Iter %d\n",id,i);
    }
```

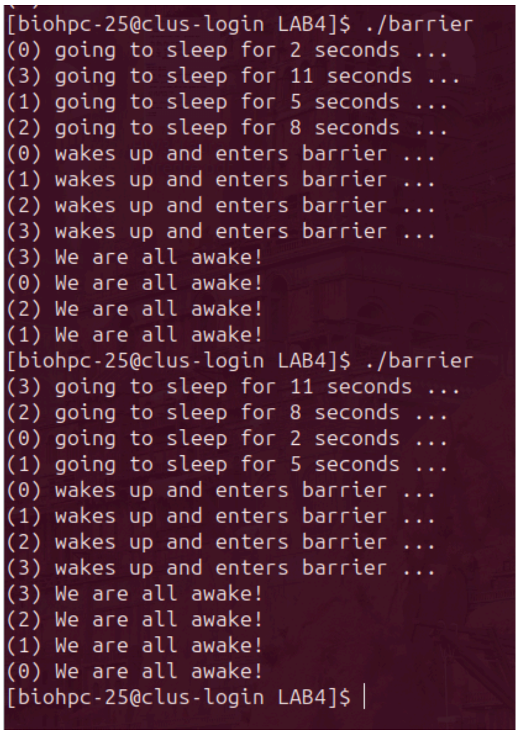
datarace.c (execute several times before answering the questions)

1. *Is the program always executing correctly?*
   No, the program does not always print the correct output

2. *Add two alternative directives to make it correct. Explain why they make the execution correct.*
   #pragma omp critical (inside the for loop) Allows only one thread to update x each time.
   #pragma omp parallel private(i) reduction(+:x) Ads the sum of x from all threads into the x variable.

barrier.c

1. *Can you predict the sequence of messages in this program? Do threads exit from the barrier in any specific order?*
   No, there is no predetermined order when threads exit the barrier so we cannot predict the order.



```
[biohpc-25@clus-login LAB4]$ ./barrier
(0) going to sleep for 2 seconds ...
(3) going to sleep for 11 seconds ...
(1) going to sleep for 5 seconds ...
(2) going to sleep for 8 seconds ...
(0) wakes up and enters barrier ...
(1) wakes up and enters barrier ...
(2) wakes up and enters barrier ...
(3) wakes up and enters barrier ...
(3) We are all awake!
(0) We are all awake!
(2) We are all awake!
(1) We are all awake!
[biohpc-25@clus-login LAB4]$ ./barrier
(3) going to sleep for 11 seconds ...
(2) going to sleep for 8 seconds ...
(0) going to sleep for 2 seconds ...
(1) going to sleep for 5 seconds ...
(0) wakes up and enters barrier ...
(1) wakes up and enters barrier ...
(2) wakes up and enters barrier ...
(3) wakes up and enters barrier ...
(3) We are all awake!
(2) We are all awake!
(1) We are all awake!
(0) We are all awake!
[biohpc-25@clus-login LAB4]$
```

for.c

1. *How many and which iterations from the loop are executed by each thread? Which kind of schedule is applied by default?*
   2 iterations for each thread since N=16 and the number of threads is set to 8
   The default schedule → static

2. *Which directive should be added so that the first printf is executed only once by the first thread that finds it?.*
   the #**pragma omp single**

schedule.c

1. *Which iterations of the loops are executed by each thread for each schedule kind?*
   static scheduling: divides N=30 % 3 threads, so 10 iterations for each thread
   static scheduling with chunk size 2: assigns 2 consecutive iterations to each thread, assigned s in a Round-Robin fashion (also 10 total iterations per thread, but iterations are distributed differently)
   dynamic scheduling with chunk size 2: assigns 2 consecutive iterations to threads that request work until all iterations are assigned (unbalanced distribution can happen depending on how fast each thread finishes the work)
   guided scheduling with chunk size 2: assigns larger chunks to threads and decreases the size as the execution progresses, (no smaller than the specified size = 2)

nowait.c

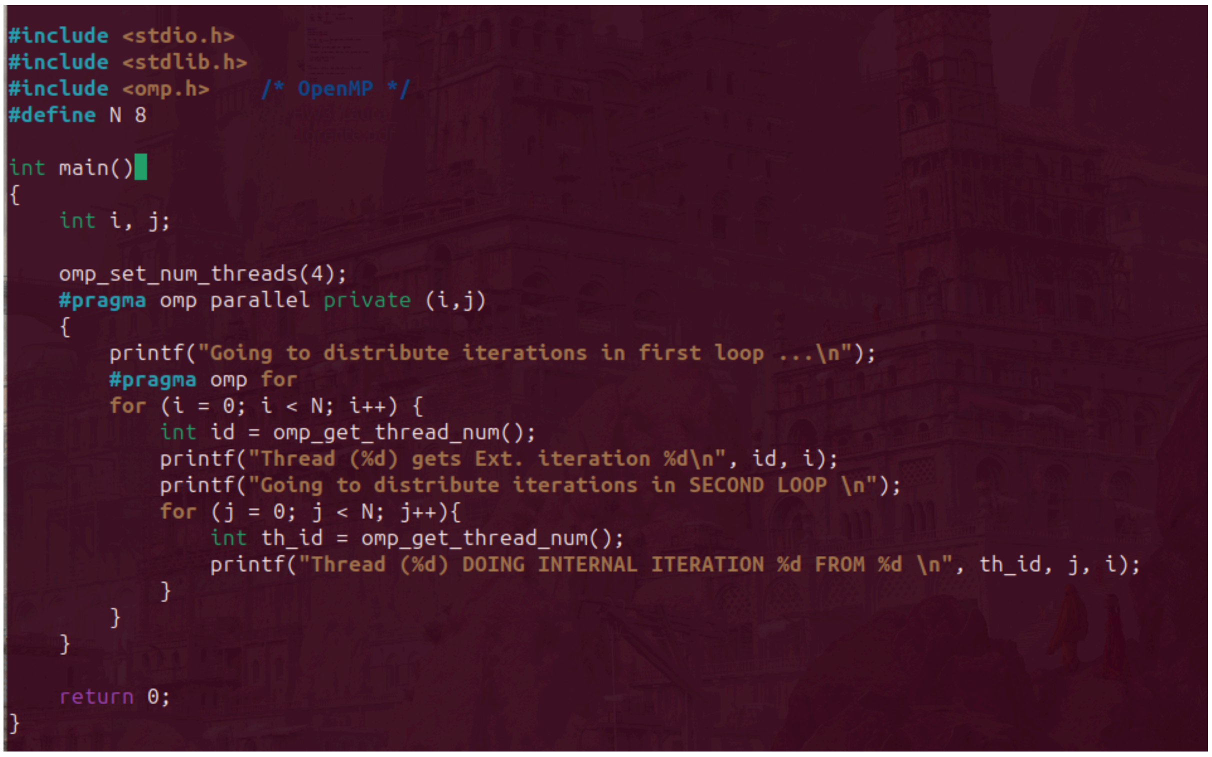
1. *How does the sequence of printf change if the nowait clause is removed from the first for directive?*
   Without the nowait in the first loop the threads will wait until the end of loop 1 before starting loop 2.
   The printf statement of loop 1 will always be before any printf of loop 2

2. *If the nowait clause is removed in the second for directive, will you observe any difference?*
   Since at the end of a **#pragma omp parallel** there is an implicit barrier, the absence of the nowait won't change the result

nested_for.c

1. *How many loop iterations are executed in the program by each thread? Variables i and j are private. Why?*
   Since i and j are private, each thread has its copies of these variables, leading to the execution of all iterations of both loops independently = NxN executions = 64

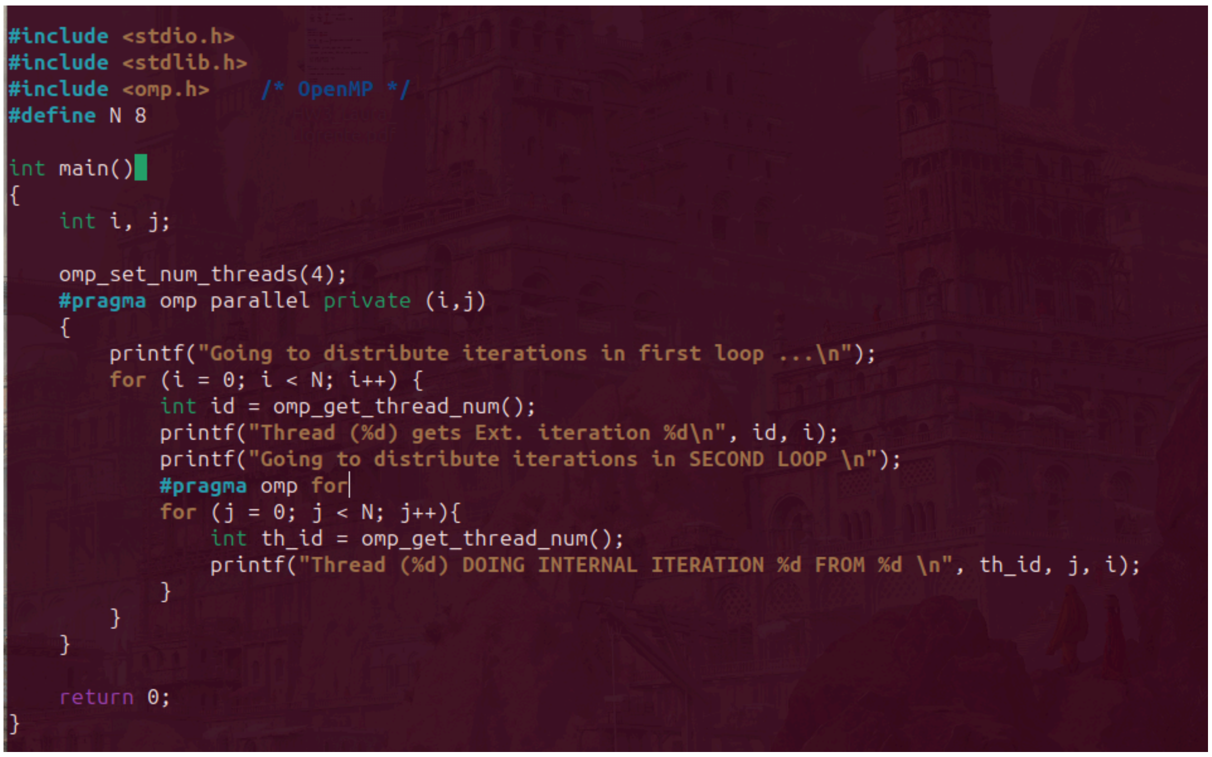
2. *Modify the program to distribute only the iterations of the outermost loop. Which loop iterations are executed by each thread now?*



```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>     /* OpenMP */
#define N 8

int main()
{
    int i, j;

    omp_set_num_threads(4);
    #pragma omp parallel private (i,j)
    {
        printf("Going to distribute iterations in first loop ...\n");
        #pragma omp for
        for (i = 0; i < N; i++) {
            int id = omp_get_thread_num();
            printf("Thread (%d) gets Ext. iteration %d\n", id, i);
            printf("Going to distribute iterations in SECOND LOOP \n");
            for (j = 0; j < N; j++){
                int th_id = omp_get_thread_num();
                printf("Thread (%d) DOING INTERNAL ITERATION %d FROM %d \n", th_id, j, i);
            }
        }
    }

    return 0;
}
```

we added the #pragma omp for before the internal iteration
The result is distributed iterations of the outermost loop i among the threads but each thread still will execute all iterations of loop j

3. *Modify the program to distribute only the innermost iterations. Which loop iterations are executed by each thread now?*



```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>     /* OpenMP */
#define N 8

int main()
{
    int i, j;

    omp_set_num_threads(4);
    #pragma omp parallel private (i,j)
    {
        printf("Going to distribute iterations in first loop ...\n");
        for (i = 0; i < N; i++) {
            int id = omp_get_thread_num();
            printf("Thread (%d) gets Ext. iteration %d\n", id, i);
            printf("Going to distribute iterations in SECOND LOOP \n");
            #pragma omp for
            for (j = 0; j < N; j++){
                int th_id = omp_get_thread_num();
                printf("Thread (%d) DOING INTERNAL ITERATION %d FROM %d \n", th_id, j, i);
            }
        }
    }

    return 0;
}
```

Each thread executes all iterations of the outer loop i but different iterations of the inner loop j ffor each value of i.

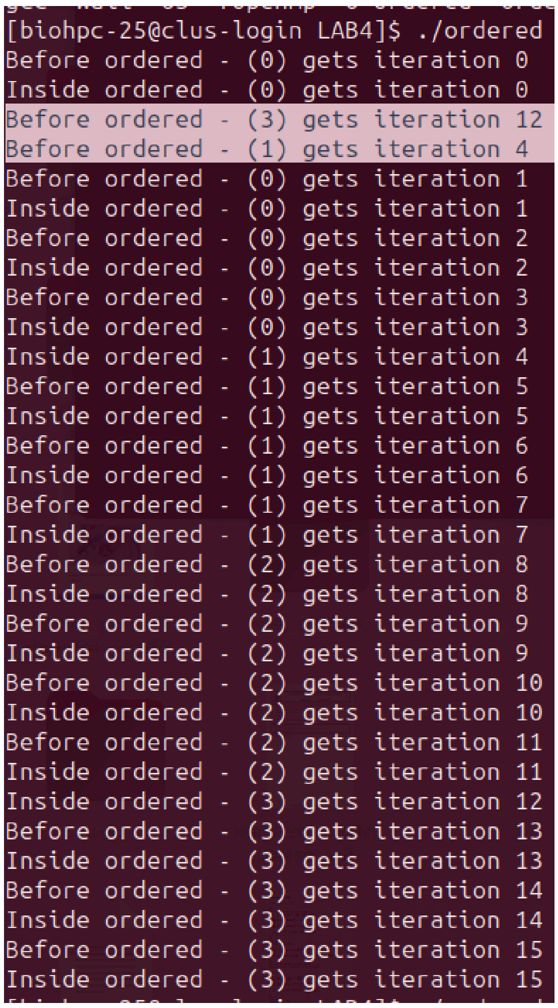
collapse.c

1. *Which iterations of the loop are executed by each thread when the collapse clause is used?*
   Collapse clause treats the nested loop as one with the outer, iterations = NxN = 36, distributed by dynamic scheduling
2. *Is the execution correct if the collapse clause is removed? Which clause (different than collapse) should be added to make it correct?.*
   if collapse is removed, the #pragma omp parallel for will apply to the outer loop only, so the result will be wrong.
   Another way of having the corrected result would be to set the type of scheduling to dynamic with a: #pragma omp parallel for schedule(dynamic)

ordered.c

1. *Can you explain the order in which printf appear?*
   #pragma omp ordered directive ensures that a specific section of the code inside the parallel loop executes in the order of loop iterations.
   In the 'before ordered', the printf statements can be executed in any order because each thread can run iterations of the loop independently, in the

‘inside ordered’ the printf is executed in the exact order of the loop iterations (from 0 to N-1= 15):



```
[biohpc-25@clus-login LAB4]$ ./ordered
Before ordered - (0) gets iteration 0
Inside ordered - (0) gets iteration 0
Before ordered - (3) gets iteration 12
Before ordered - (1) gets iteration 4
Before ordered - (0) gets iteration 1
Inside ordered - (0) gets iteration 1
Before ordered - (0) gets iteration 2
Inside ordered - (0) gets iteration 2
Before ordered - (0) gets iteration 3
Inside ordered - (0) gets iteration 3
Inside ordered - (1) gets iteration 4
Before ordered - (1) gets iteration 5
Inside ordered - (1) gets iteration 5
Before ordered - (1) gets iteration 6
Inside ordered - (1) gets iteration 6
Before ordered - (1) gets iteration 7
Inside ordered - (1) gets iteration 7
Before ordered - (2) gets iteration 8
Inside ordered - (2) gets iteration 8
Before ordered - (2) gets iteration 9
Inside ordered - (2) gets iteration 9
Before ordered - (2) gets iteration 10
Inside ordered - (2) gets iteration 10
Before ordered - (2) gets iteration 11
Inside ordered - (2) gets iteration 11
Inside ordered - (3) gets iteration 12
Before ordered - (3) gets iteration 13
Inside ordered - (3) gets iteration 13
Before ordered - (3) gets iteration 14
Inside ordered - (3) gets iteration 14
Before ordered - (3) gets iteration 15
Inside ordered - (3) gets iteration 15
```

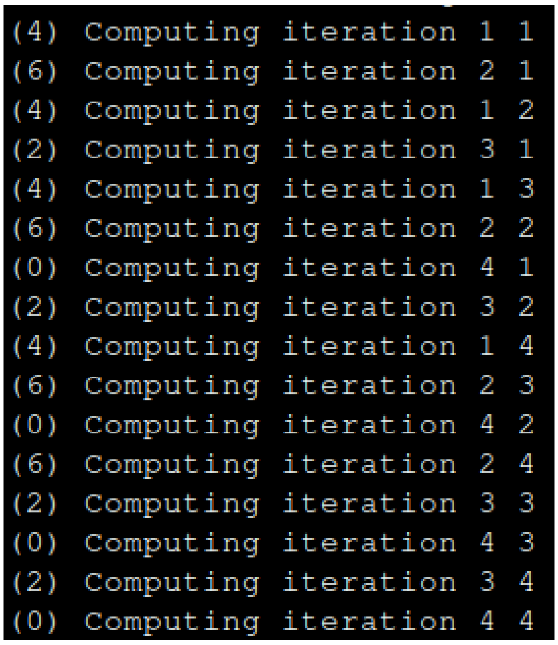
The before ordered does not follow the loop iterations order

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?*
   We can specify the number of consecutive iterations by adding: **#pragma omp for schedule(static, 2) ordered** and another **printf("Inside ordered - (%d) gets iteration %d\n", id, i+1)**, so each thread handles two consecutive iterations and order is preserved.

doacross.c

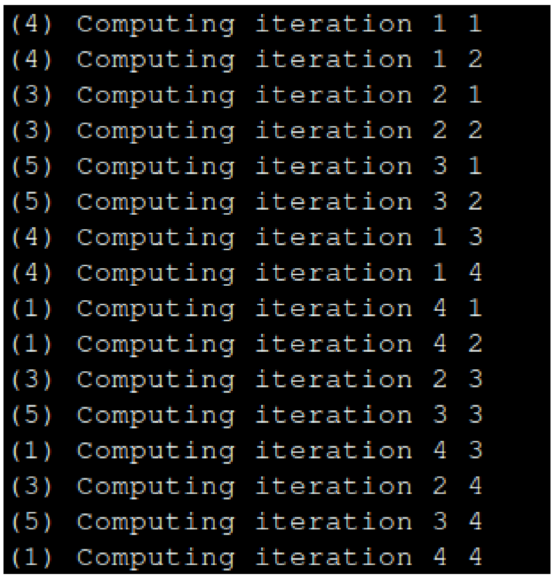
1. *In which order are the ”Outside” and ”Inside” messages printed?*
   Outside messages are printed in any order because of the dynamic scheduling. Inside messages are printed respecting the #pragma omp ordered depend(sink: i-3), which just prints the Inside message for iteration i after iteration i-3 has been completed.

2. *In which order are the iterations in the second loop nest executed?*
   The second loop iterates with dependencies to the first i loop, i,j depends on (i-1,j) and (i, j-1) (except on 1,1), this is a wavefront pattern. This pattern ensures that all computations are done respecting dependencies, so that there are no computation errors.



```
(4) Computing iteration 1 1
(6) Computing iteration 2 1
(4) Computing iteration 1 2
(2) Computing iteration 3 1
(4) Computing iteration 1 3
(6) Computing iteration 2 2
(0) Computing iteration 4 1
(2) Computing iteration 3 2
(4) Computing iteration 1 4
(6) Computing iteration 2 3
(0) Computing iteration 4 2
(6) Computing iteration 2 4
(2) Computing iteration 3 3
(0) Computing iteration 4 3
(2) Computing iteration 3 4
(0) Computing iteration 4 4
```

3. *What would happen if you remove the invocation of sleep(1). Is the order of the iterations modified with respect to the previous case? Execute several times to answer in the general case.*
   After removing the sleep(1) buffer the iteration order changes due to some execution times being faster than others, this alters the timing of executions which breaks the would-be wavefront pattern. Because of this, the matrix is calculated by rows in ascending 1 to M-1 order in the i and j loops.



```
(4) Computing iteration 1 1
(4) Computing iteration 1 2
(3) Computing iteration 2 1
(3) Computing iteration 2 2
(5) Computing iteration 3 1
(5) Computing iteration 3 2
(4) Computing iteration 1 3
(4) Computing iteration 1 4
(1) Computing iteration 4 1
(1) Computing iteration 4 2
(3) Computing iteration 2 3
(5) Computing iteration 3 3
(1) Computing iteration 4 3
(3) Computing iteration 2 4
(5) Computing iteration 3 4
(1) Computing iteration 4 4
```