Entregable de Laboratori 2: Brief tutorial on OpenMP programming model

GUILLEM GRÀCIA ANDREU, GERARD MADRID MIRÓ

TARDOR 2021-22

PAR3109

Dia 1: Regions paral·leles i tasques implícites.

1.hello.c

1. How many times will you see the "Hello world!" message if the program is executed with "./1.hello"?
Un cop executat el primer arxiu, veiem el missatge dues vegades, una per cada thread assignat (per defecte, 2).

2. Without changing the program, how to make it print 4 times the "Hello World!" message?
Com sabem que el número de threads és el causant de les impressions, haurem d'ampliar quants threads treballen aquest codi sense modificar-lo amb la comanda linux següent:

```
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?

L'execució no és correcta. Per tal de solucionar-ho hem afegit la següent comanda pragma al codi:

```
#PRAGMA OMP private (id)
```

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

En ocasions, les línies impresses apareixen entrellaçades i no tenen l'ordre correcte. Això és degut a que els threads no s'esperen a que acabi l'anterior per acabar ells. Ho podem solucionar fàcilment afegint al codi la sentencia següent:

```
#PRAGMA OMP critical
```

3.how many.c:

Assuming the OMP NUM THREADS variable is set to 8 with "OMP NUM THREADS=8 ./3.how many"

1. What does omp get num threads return when invoked outside and inside a parallel region?
Fora de la regió paral·lela, la funció retorna un 1, mentre que dins de la mateixa retorna el número de threads que el programa té treballant.

2. Indicate the two alternatives to supersede the number of threads that is specified by the OMP NUM THREADS environment variable.
Per tal de modificar el número de threads treballant en una regió del programa, podem fer servir una de les dues operacions següents:

```
omp_set_num_threads(n_th)
#PRAGMA OMP PARALLEL num_threads(n_th)
```

3. Which is the lifespan for each way of defining the number of threads to be used?
La primera opció (`omp_set_num_threads(n_th)`), tindrà com a regió de validesa des de que s'executa fins que arribi la següent secció paral·lela.

Per una altra banda, la segona opció (`#PRAGMA OMP PARALLEL num_threads(n_th)`), té com a regió de validesa la línia o regió de codi (delimitada per claus { })que ve just després.

4.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, firstprivate and reduction)? Is that the value you would expect? (Execute several times if necessary)

Shared:
Al ser compartida, x serà igual a la suma de cada valor del thread id, per tant tindrem:

$$x = \sum_{i=0}^{15} i = 120$$

Private:
El resultat és 5, ja que al ser privada el valor de x per a cada thread és una còpia, per tant, al sortir de la regió paral·lela no es modifica l'original. A més a més la còpia s'inicialitza en 0, no en el seu valor original (5).

Firstprivate:
El resultat és 5 i el motiu és el mateix que l'anterior però aquesta vegada sí que s'inicialitza al valor que hi havia abans de la secció paral·lela, és a dir 5.

Reduction:
Finalment, reduction fa una còpia per cada thread i al final suma tots els resultats de totes les versions paral·leles. S'inicialitza al valor que hi havia abans de la secció paral·lela (com a Firstprivate).
El resultat és 125.

5.datarace.c

1. Should this program always return a correct result? Reason either your positive or negative answer.
No sempre hauria de retornar un resultat correcte degut a que és shared i pot passar el següent:

Thread A	Thread B
temp = 15	
	temp = 11
max = 15	
	max = 11

Com podem comprovar, el Thread A escriu a temp un valor màxim de 15. No obstant haver trobat ja el màxim, està en una variable temporal, i tot just abans d'acabar l'execució, el Thread B pot escriure el seu màxim (11) a la mateixa variable definitiva que havia escrit Thread A.

2. Propose two alternative solutions to make it correct, without changing the structure of the code (just add directives or clauses). Explain why they make the execution correct.
Una opció seria utilitzar "`#PRAGMA OMP critical`" abans de l'if dins del for. La condició critical obliga als threads a esperar que acabi l'execució de l'anterior per començar. D'aquesta manera evitariem el problema que hem esmentat.

Una altra opció seria utilitzar "`#PRAGMA OMP reduction(max:maxValue)`". En aquest cas, utilitzem reduction amb el paràmetre "max", que permet a tots els threads executar-se i, finalment, recuperar tots els valors i treure el màxim i guardar-lo a "maxValue".

3. Write an alternative distribution of iterations to implicit tasks (threads) so that each of them executes only one block of consecutive iterations (i.e. N divided by the number of threads.
Per tal de fer el que especifica l'enunciat haurem de canviar les condicions del for de la següent manera:

```
for(i=id; i < id + N / howmany; ++i)
```

Cada thread comença amb l'iterador al seu identificador, i anirà iterant fins a $N/n_{threads}$ iteracions més enllà.

6.datarace.c

1. Should this program always return a correct result? Reason either your positive or negative answer.
Tenim el mateix problema que en l'apartat 5.1, el resultat no és el correcte.

2. Propose two alternative solutions to make it correct, without changing the structure of the program (just using directives or clauses) and never making use of critical. Explain why they make the execution correct.
Una alternativa seria utilitzant el reduction com el cas anterior però amb el paràmetre de suma en lloc del "max":

```
#PRAGMA OMP reduction(+:maxCount)
```

Una altra seria utilitzar l'atomic. En aquest cas es pot utilitzar a diferència de l'anterior donat que son operacions purament aritmètiques, que és l'únic que permet atomic. Per tant només cal afegir la següent línia:

```
#PRAGMA OMP atomic
```

7.datarace.c

1. Is this program executing correctly? If not, explain why it is not providing the correct result for one or the two variables (countmax and maxvalue)
No funciona correctament ja que està sumant tots els màxims que troba cada un dels threads ja que es separen en subconjunts i no troba el màxim del conjunt sencer.

2. Write a correct way to synchronise the execution of implicit tasks (threads) for this program.
S'han de separar els dos bucles. El primer busca el número més gran i el segon conta quants hi ha.

8.barrier.c

1. Can you predict the sequence of printf in this program? Do threads exit from the #pragma omp barrier construct in any specific order?
No, no es poden predir de cap manera ni els logs d'entrada ni els de sortida tot i tenir temps de sleep totalment diferents cadascun.

Dia 2: Tasques explícites

1.single.c

1. What is the nowait clause doing when associated to single?
En comptes d'executar un per un els threads esperant a que acabi el thread anterior, ara executa els 4 (o tants com threads s'hagin creat) alhora . Al ser single, cada thread executarà una instància diferent.

2. Then, can you explain why all threads contribute to the execution of the multiple instances of single? Why those instances appear to be executed in bursts?
El temps d'espera que hi ha al final de cada iteració ajuda a que tots els threads hagin de participar ja que si traiem el sleep, ho fa gairebé tot el mateix thread al ser accions molt poc exigents.

2.fibtasks.c

1. Why are all tasks created and executed by the same thread? In other words, why is the program not executing in parallel?
No hi ha execució en paral·lel donat que en cap moment s'explicita que s'iniciï tal comportament. No existeix cap sentència `#pragma omp parallel`

2. Modify the code so that tasks are executed in parallel and each iteration of the while loop is executed only once.
Hem afegit el `#pragma omp parallel` per fer-ho paral·lel i el `#pragma omp single` per a repartir les tasques.

3. What is the firstprivate(p) clause doing? Comment it and execute again. What is happening with the execution? Why?
Les referències es trenquen degut a que no es descarten els canvis de cada thread i aleshores estem sobreescrivint el valor de p que inclús podría ser null, el que resulta en un *segmentation fault* probable.

3.taskloop.c

1. Which iterations of the loops are executed by each thread for each task grainsize or num tasks specified?
En el cas de *task grainsize* s'executen 3 tasques de 4 iteracions mentre que el *num task* executa 4 tasques de 3 iteracions.

2. Change the value for grainsize and num tasks to 5. How many iterations is now each thread executing? How is the number of iterations decided in each case?
Grainsize(5) reparteix 2 tasques de 6 iteracions i el *num task(5)* executa 6 tasques de 2 iteracions.

3. Can grainsize and num tasks be used at the same time in the same loop?
No. En cas d'intentar afegir les dues clàusules el codi no compila.

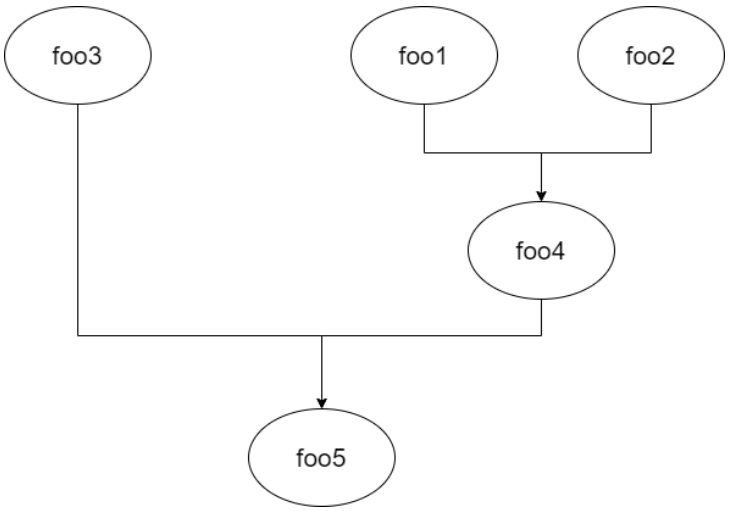
4. What is happening with the execution of tasks if the nogroup clause is uncommented in the first loop? Why?
S'executen primer els printf de fora del for i després s'executen els bucles. Això és degut a que elimina la delimitació de regió del taskloop.

4.reduction.c

1. Complete the parallelisation of the program so that the correct value for variable sum is returned in each printf statement. Note: in each part of the 3 parts of the program, all task generated should potentially execute in parallel.
Cal afegir sum=0 abans de cada for per no acumular la suma i a més a més un `#pragma omp atomic` abans de cada increment de sum.

5.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), trying to achieve the same potential parallelism that was obtained when using depend.

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


        #pragma omp parallel
        {

                #pragma omp single
                {
                        printf("Creating task foo1\n");
                        printf("Creating task foo2\n");
                        printf("Creating task foo3\n");
                        printf("Creating task foo4\n");
                        printf("Creating task foo5\n");

                        #pragma omp task
                        foo3();
                        #pragma omp task
                        foo1();
                        #pragma omp task
                        foo2();
                        #pragma omp taskwait
                        foo4();
                        #pragma omp task
                        foo5();

                }

        }

    return 0;
}
```

Per tal que funcioni el programa de la forma adequada, afegim a cada funció el task necessari, utilitzant el taskwait si s'escau.

3. Rewrite the program using only taskgroup as task synchronisation mechanism (no depend clauses allowed), again trying to achieve the same potential parallelism that was obtained when using depend.

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


        #pragma omp parallel
        {

                #pragma omp single
                {
                        printf("Creating task foo1\n");
                        printf("Creating task foo2\n");
                        printf("Creating task foo3\n");
                        printf("Creating task foo4\n");
                        printf("Creating task foo5\n");

                        #pragma omp taskgroup
                        {
                                #pragma omp task
                                foo3();
                                #pragma omp taskgroup
                                {
                                        #pragma omp task
                                        foo1();
                                        #pragma omp task
                                        foo2();
                                }
                                #pragma omp task
                                foo4();
                        }
                        #pragma omp task
                        foo5();

                }

        }

    return 0;
}
```

Per tal que funcioni el programa de la forma adequada, afegim taskgroups similant el diagrama de dependències de la versió inicial. La funció 1 amb la 2, la 3 alhora que la 4, i finalment la 5.

Conclusions

D'aquesta segona entrega de laboratori ens emportem, a part d'alguna frustració, la capacitat d'entendre el funcionament de la paral·lelització, les tasques implícites i explícites mitjançant *OpenMP*, i una gran quantitat de clàusules `#pragma` per a poder paral·lelitzar com vulguem el nostre programa.