**Exercicis Pràctics OpenMP (Part II)**

L’objectiu principal d’aquests exercicis és que experimenteu amb les capacitats d’OpenMP fent servir casos simples, facilitant d’aquesta manera la transició entre els continguts discutits en la class de teoria i la seva aplicació al cas pràctic (més complex) treballat en el laboratori.

El plantejament dels exercicis i la mecànica de treball per resoldre’ls consisteixen en:

1. Per cada apartat, es proporcionen un conjunt de fragments de codi que caldrà executar i analitzar (els programes corresponents els trobareu a /home/alumnos/pp/alumnos/Avaluats-problemes/OpenMP/sessio2/, copieu els arxius al vostre compte per fer els problemes).
2. Per cada apartat, es fa un conjunt de preguntes sobre cada fragment de codi presentat. Heu de respondre cada pregunta, **justificant sempre la vostra resposta**. En alguns casos la justificació serà molt curta, en altres més complexa i, en altres, potser consistirà en un nou fragment de codi.
3. La solució als exercicis s’inclourà en aquest document que s’haurà de lliurar al CV en la data establerta.

**Exercicis**

1. **Seccions Paral·leles**

Per al codi de l’**apartat-a.c**, creeu una regió paral·lela amb quatre threads que englobi els tres bucles que es proposa modificar. Apliqueu una clàusula que permeti que si el nombre d’elements **N** es menor a 100, l’execució del programa serà completament seqüencial. Dins d'aquesta regió s'han d'assignar els dos primers bucles com a tasques per a threads independents (*sections*). El tercer bucle s’haurà de paral·lelitzar amb un únic pragma amb el treball a repartir entre tots els threads.

* + 1. Mostreu el vostre codi paral·lel. Expliqueu com funciona i que fan cadascuna de les clàusules que heu utilitzat.
    2. Les clàusules utilitzades duen alguna espera implícita ? Que implica això ?
    3. Es possible modificar el codi afegint alguna clàusula d’OpenMP per tal de que els threads no esperin a la finalització dels dos primers bucles per continuar el seu treball ?
    4. Seria correcte el codi si apliquem la modificació esmentada a l’apartat (iii) ? Per què ?

1. **Paral·lelitzant Collatz.**

Considerem la següent funció sobre un nombre natural arbitrari n:

Ara, definim la seqüència següent per qualsevol nombre natural arbitrari n:

La conjectura de Collatz diu que aquesta seqüència acabarà en el valor 1 en un nombre finit de passos (anomenat temps d’aturada total) independentment del nombre natural escollit inicialment.

El següent codi (collatz.c)trobar el nombre Y amb el temps d’aturada més alt en el rang [1:X]:

#include <stdio.h>

#include <stdlib.h>

unsigned long f(unsigned long x){

return (x%2) ? 3\*x + 1 : x/2;

}

int stop( unsigned long x ){

int cont = 0;

while ( x > 1 ){ x = f(x); cont++; }

return cont;

}

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

unsigned long n, i, Num;

unsigned len, Max = 0, id;

if ( argc < 2 ){ printf("Error: cal indicar el número natural\n" ); exit(1); }

if ((n = atoi(argv[1])) <= 0 ) { printf("Error: el número ha de ser un natural (> 0)\n" ); exit(1); }

for ( i = 1; i <= n; i++ ){

len = stop(i);

if ( len > Max ) { Max = len; Num = i; };

}

printf("El el nombre menor que %lu amb temps d'auturada més alt és %lu amb %u passos\n", n, Num, Max );

}

* + 1. Sembla clar que aquest codi es pot paral·lelitzar amb OpenMP. Com ho faríeu per tal de que aquest codi s’executés en paral·lel fent servir 6 threads, sempre que el **n** sigui més gran o igual a 106? (La resposta no és trivial. Adjunteu el codi generat en la vostra resposta).
    2. Compileu la versions paral·lela i seqüencial. Feu servir l’eina perf per obtenir el temps d’execució d’ambdues versions pel cas n=206 (*perf stat ./<executable> 20000000 [l’script job.sub ja està preparat per aquesta execució]*). Quina acceleració obteniu? Quants processadors diu l’eina perf que heu utilitzat?
    3. Carregueu ara l’entorn per poder fer servir les utilitats **likwid** (*module load likwid/5.2.2*) i executeu *likwid-perfctr -C 0-5 -g FLOPS\_SP ./<executable-omp> 20000000 [l’script job.sub ja està preparat per aquesta execució]*. De tota la sortida que es produeix, observeu la primera taula METRIC (la que dona els resultats per core). En particular l’entrada **Runtime unhalted**. Estan els diferents threads trigant el mateix temps en fer la seva feina? Quin és el percentatge de variació? Que ens pot estar indicant aquesta diferència i per què?
    4. Quina clàusula podem fer servir per millorar encara més el rendiment de la nostra aplicació? (apliqueu-la i indiqueu la millora de rendiment que obteniu)