Metodología de la Programación Paralela 2018-2019
Prueba práctica, 24 enero 2019

Estructura de la prueba:

- Entre 16:00 y 17:30, se realizará una prueba escrita respondiendo las cuestiones que se plantean a continuación, pudiendo utilizar material bibliográfico pero sin trabajar con el ordenador. Se entregarán las soluciones antes de empezar la prueba sobre el ordenador. Cada alumno se puede quedar con esquemas de las soluciones que proponen.

- De 17:30 a 19:15, los alumnos trabajarán resolviendo las cuestiones sobre el ordenador, pudiendo acceder a los recursos de internet que consideren, pero sin copiar. Se tendrán en cuenta sólo soluciones que implementen las soluciones propuestas por el alumno en la primera parte. El sistema se cerrará automáticamente a las 19:15.

- Antes de las 19:30 se entregará un documento escrito en el que se indicarán los cambios realizados sobre la versión presentada en la parte escrita y los números de envío en los que se implementan. Para cada cuestión hay que indicar un único número de envío que será el que se evalúe. Para las cuestiones que no se tenga envíos aceptados, el alumno puede indicar el envío que considere que tiene más cercano a una solución correcta.

Se puede trabajar en el laboratorio, pero hay que hacer los envíos finales a calisto.inf.um.es.
Se trabajará con el concurso 2019Enero.
El profesor dejará en un anuncio en el aula virtual los enunciados de los problemas junto con los esquemas y los ficheros con las soluciones secuenciales.
Se muestra la puntuación máxima de cada cuestión. Para cada una de ellas, la mitad de la puntuación corresponde a la parte escrita y la otra a la parte sobre ordenador.
La suma de la puntuación de todas las cuestiones es 11, pero la puntuación máxima que se puede obtener es 10. Esta puntuación se transformará a una puntuación sobre 3 o 4 según se siga la evaluación continua o la no continua.

PROBLEMA 1: Conteo de distancias a elementos iguales en una matriz.
Dada una matriz de caracteres, A, de dimensión $N \times N$, se obtiene otra matriz de enteros D de las mismas dimensiones. En cada posición (i, j) de D se almacena la menor distancia en A entre la posición (i, j) y otra posición con el mismo carácter $A[i, j]$. La distancia entre dos posiciones $(x1, y1)$ y $(x2, y2)$ es la distancia de Manhatan: $|x2 - x1| + |y2 - y1|$. Cuando no hay otra posición con el mismo valor se almacena el valor cero. Finalmente se suman todos los valores almacenados en D.
Por ejemplo, dada la matriz
aabc
cbab
bcca
accb
las distancias son
1 1 2 3
2 2 2 2
2 1 1 2
3 1 1 2
y el resultado es 28.
Se resuelven varios problemas. Para cada problema la función a paralelizar tiene parámetros:

- Input parameter:

  -int N: número de filas y columnas

  -char $*A$: matriz de caracteres, de dimensión $N \times N$

- Return:

  int suma de los valores almacenados en D y calculados como se ha indicado

La entrada tiene en la primera línea el número de problemas. Para cada problema hay una línea con tres valores: N, la semilla para la generación aleatoria, número de caracteres distintos a incluir en el array.
CUESTIÓN 1-a (2 puntos)
Hacer una implementación paralela con OpenMP que utilice paralelismo de bucles. Hay que justificar qué bucles se paralelizan y cuáles no e indicar la forma en que se paralelizan: schedule, variables privadas... (Problema A en mooshak).
CUESTIÓN 1-b (2 puntos)
Hacer una implementación paralela con OpenMP que utilice tareas. Hay que justificar dónde se lanzan las tareas y cuántas se lanzan (Problema A).
CUESTIÓN 1-c (2 puntos)
Hacer una implementación paralela con MPI (Problema B). Explicar con $N = 7$ y 3 procesos cómo funcionaria con la matriz

abcddcb
cdbbadd
cbdbbaa
bddbbac
dbddacc
adccbdb
dcbbbad

CUESTIÓN 1-d (1.5 puntos)
Hacer una implementación MPI+OpenMP (Problema C).

PROBLEMA 2: Subcadena común más larga entre dos cadenas de caracteres.

Dadas dos cadenas de caracteres, $C1$ y $C2$, con N caracteres cada una, se trata de encontrar la longitud de la subcadena más larga común a las dos cadenas, pudiendo haber huecos en los caracteres seleccionados en las cadenas. Por ejemplo, dadas "abbcb" y "abcab" la longitud es 4, y corresponde a la subcadena "abca", que se encuentra en las dos cadenas en varias formas: "ab-cb" y "abc-b", o "a-bcb" y "abc-b".

Se resuelven varios problemas. Para cada problema la función a paralelizar tiene parámetros:

- Input parameter:

  -int N: número de elementos de cada cadena

  -int $*C1$, $*C2$: las dos cadenas

- Return:

  int longitud de la subcadena común más larga

La entrada tiene en la primera línea el número de problemas. Para cada problema hay una línea con tres valores: N, la semilla para la generación aleatoria, número de caracteres distintos a incluir en las cadenas.
CUESTIÓN 2-a (2 puntos)
Hacer una implementación paralela con MPI (Problema E en mooshak). Idea: se puede utilizar el esquema de paralelismo *pipeline*.
Explicar cómo funcionaría con 3 procesos y las cadenas "acbbcaddc" y "cbdcbaca".
CUESTIÓN 2-b (1.5 puntos)
Hacer una implementación paralela con OpenMP (Problema D en mooshak). Idea: se puede adaptar el código anterior de MPI para trabajar en OpenMP.

Se incluyen los códigos secuenciales de los dos problemas:
PROBLEMA 1:

```
/*
  CPP_CONTEST=2019Enero
  CPP_PROBLEM=A
  CPP_LANG=CPP+OPENMP
  CPP_PROCESSES_PER_NODE=calisto 1
*/

#include <stdlib.h>
```

```
#include <omp.h>

void distancia(int n,int row,int column, char *A,int *D)
{
    int distancia=0; //guarda la distancia, si no lo encuentra es 0
    int paso=1; //indica el paso por el que vamos
    int encontrado=0; //indica si se ha encontrado
    char elemento=A[row*n+column]; //elemento con el que vamos a comparar
    int rowact,columnizq,columnder; //en cada paso: fila actual, columnas a
//izquierda y derecha de la que vamos
    while(!encontrado && paso!=(2*n-1)) //hay un numero maximo de pasos.
//Si se encuentra se sale
    {
       for(rowact=row-paso,columnizq=column,columnder=column;
!encontrado && rowact<=row;rowact++,columnizq--,columnder++)
//el paso indica cuantas filas por arriba, la primera vez estamos en la misma
//columna, tanto a izquierda como a derecha
//si se encuentra se acaba, o si se llega a la fila actual
//se va bajando de fila y la columna izquierda se hace una a la izquierda y l
//a derecha una a la derecha
       {
          if((rowact>=0) && ((columnizq>=0 &&
elemento==A[rowact*n+columnizq]) || (columnder<n
&& elemento==A[rowact*n+columnder])))
//si no nos salimos de la matriz por arriba y
//no se sale la columna izquierda y coincide con el elemento, o no se sale
//la columna derecha y coincide con el elemento
          {
            encontrado=1;
            distancia=paso;
          }
       }
      for(rowact=row+1,columnizq=column-paso+1,columnder=column+paso-1;
!encontrado && rowact<=row+paso;rowact++,columnizq++,columnder--)
//se empieza en la fila de abajo, la primera vez estamos paso-1 columnas
//mas a la izquierda y a la derecha
//si se encuentra se acaba, o si se llega a la fila actual mas el paso por el que vamos
//se va bajando de fila y la columna izquierda se hace una a la derecha y
//la derecha una a la izquierda
       {
          if((rowact<n) && ((columnizq>=0 && elemento==A[rowact*n+columnizq])
|| (columnder<n && elemento==A[rowact*n+columnder]))) {
            encontrado=1;
            distancia=paso;
          }
       }
       paso++;
    }
    D[row*n+column]=distancia;
}

int sumar(int n,int *m)
//suma todos los elementos del array
{
  int s=0;
  for(int i=0;i<n;i++)
    s+=m[i];
```

```
  return s;
}

int  sec(int n,char *A)
{
  int *D=(int*) calloc(sizeof(int),n*n);
  int result;
  for (int i = 0; i <n; i++) {
     for(int j=0;j<n;j++) {
        distancia(n,i,j,A,D); //calcula la distancia para cada elemento
     }
  }
  result=sumar(n*n,D);
  delete[] D;
  return result;
}
```

PROBLEMA 2:

```
/*
  CPP_CONTEST=2019Enero
  CPP_PROBLEM=D
  CPP_LANG=CPP+OPENMP
  CPP_PROCESSES_PER_NODE=calisto 1
*/
#include <stdlib.h>
#include <omp.h>

int  sec(int n,char *C1,char *C2)
{
  int dim=n+1; //fila y columna 0 para casos base
  int resul;
  int *T=(int*) calloc(sizeof(int),dim*dim); //tabla para programación dinámica

  for(int i=0;i<dim;i++) //caso base fila
     T[i]=0;
  for(int i=1;i<dim;i++) //caso base columna
     T[i*dim]=0;
  for (int i = 1; i <dim; i++) {
     for(int j=1;j<dim;j++) {
//máximo de la fila y columna anterior
        int maximo=(T[(i-1)*dim+j]>T[i*dim+j-1]?T[(i-1)*dim+j]:T[i*dim+j-1]);
//si los elementos de las cadenas coinciden y añadiendo 1 al valor en la diagonal
//anterior es mayor que el máximo, se actualiza
        if(C1[i-1]==C2[j-1] && T[(i-1)*dim+j-1]+1>maximo)
          maximo=T[(i-1)*dim+j-1]+1;
        T[i*dim+j]=maximo;
     }
  }
//la longitud de la cadena más larga es el último valor en la tabla de programación dinámica
  resul=T[dim*dim-1];
  delete[] T;
  return resul;
}
```