Práctica 4

Implementación paralela multihebra de algoritmos modelo usando OpenMP

En esta práctica se aborda la implementación paralela del algoritmo de Floyd para el cálculo de todos los caminos más cortos en un grafo etiquetado usando directivas y funciones de OpenMP. Al igual que en la práctica 1, se desarrollarán dos versiones paralelas del algoritmo que difieren en el enfoque seguido para asignar datos a las hebras.

La definición del problema a resolver (problema de los caminos más cortos) y del algoritmo a paralelizar (algoritmo de Floyd) es la misma que se dio en la práctica 1 y se puede usar el programa secuencial que se ofrece en la práctica 1 como plantilla para realizar la práctica.

```
procedure floyd secuencial
begin
I_{i,j} = A
  for k := 0 to N-1
    for i := 0 to N-1
      for j := 0 to N-1
        I^{k+1}_{i,j} = min{I^k_{i,j}, I^k_{i,k} + I^k_{k,j}}
  end;
```

Se desarrollarán dos versiones paralelas del algoritmo, tal como se hizo en la práctica 1. En cada versión, la distribución de los datos entre las tareas es idéntica a la mostrada en la práctica 1 pero teniendo en cuenta que ahora las tareas son hebras que se comunican entre sí mediante variables compartidas.

4.1 Descomposición unidimensional (por bloques de filas)

Asumimos que el número de vértices N es múltiplo del número de procesos P.

En esta versión, las filas de la matriz I se distribuyen entre las hebras por bloques contiguos de filas, por lo que cada hebra almacena N/P filas de I y de la matriz de salida S.

Se podrán utilizar hasta N hebras como máximo. Cada hebra es responsable de una o más filas adyacentes de I y ejecutará el siguiente algoritmo:

```
procedure floyd paralelo 1
```

```
begin
I_{i,j} = A
  for k := 0 to N-1
    for i := local_i_start to local_i_start
      for j := 0 to N-1
        I^{k+1}_{i,j} = min{I^k_{i,j}, I^k_{i,k} + I^k_{k,j}}
  end;
```

4.2 Descomposición bidimensional (por bloques 2D)

Por simplicidad, se asume que el número de vértices N es múltiplo de la raíz del número de hebras P.

Esta versión del algoritmo de Floyd utiliza una distribución por bloques bidimensionales de la matriz I entre las hebras, pudiendo utilizar hasta N^2 hebras. Suponemos que las hebras se organizan lógicamente como una malla cuadrada con $\sqrt{P}$ hebra en cada fila y $\sqrt{P}$ hebras en cada columna.

Cada hebra trabaja con un bloque de $N/\sqrt{P}$ subfilas alineadas (cubren las mismas columnas contiguas) con $N/\sqrt{P}$ elementos cada uno y ejecuta el siguiente algoritmo:

```
procedure floyd paralelo 2
begin
I_{i,j} = A
  for k := 0 to N-1
    for i := local_i_start to local_i_end
      for j := local_j_start to local_j_end
        I^{k+1}_{i,j} = min{I^k_{i,j}, I^k_{i,k} + I^k_{k,j}}
  end;
```

4.3 Ejercicios propuestos.

1. Implementar los algoritmos de cálculo de todos los caminos más cortos que han sido descritos previamente. Se usará como plantilla la versión secuencial en C++ del algoritmo de Floyd proporcionada en la práctica 1. Se debe crear una carpeta diferente para cada versión paralela (`Floyd-1` y `Floyd-2`).

   Cada carpeta mantendrá los mismos archivos que se usan en la versión secuencial pero con diferente código. De esa forma el `Makefile` se puede reutilizar y se percibe claramente la diferencia entre las versiones secuencial y paralela.

2. Realizar medidas de tiempo de ejecución sobre los algoritmos implementados. Para medir tiempos de ejecución, podemos utilizar la función `omp_get_wtime()`. Deberán realizarse las siguientes medidas:

   (a) Medidas para el algoritmo secuencial (Una sola hebra).

   (b) Medidas para el algoritmo paralelo con diferente número de hebras (P=2, 4, 8).

Las medidas deberán realizarse para diferentes tamaños de problema, para así poder comprobar el efecto de la granularidad sobre el rendimiento de los algoritmos.
Se presentará una tabla con el siguiente formato:

Tiempo	$P = 1$ (secuencial)	$P = 2$	$P = 4$	$P = 8$	Ganancia
$n = 60$					
$n = 240$					
$n = ...$					
$n = 1200$					