Grado en Ingeniería Informática
G653 - Programación Paralela Concurrente y de Tiempo Real

Práctica 2

Paralelismo funcional asíncrono

NOTA: Es muy importante leer atentamente los enunciados de los ejercicios, para realizarlos correctamente. Los fallos derivados de no haber leído el enunciado se penalizarán fuertemente en la nota de la práctica.

1. Introducción

En esta práctica se propone la paralelización de un problema de filtrado de imágenes bidimensionales en un multiprocesador de memoria compartida mediante el paradigma de programación OpenMP. El problema consiste en leer de disco un fichero que contiene un vídeo, compuesto por un conjunto de imágenes. A cada imagen se le aplica un filtro y se almacena el resultado en un fichero de salida. El objetivo final es conseguir mejorar todo lo posible el tiempo de ejecución del programa secuencial proporcionado en el enunciado de esta práctica, utilizando el paralelismo de tareas (`tasks`).

2. Objetivos

Mediante el desarrollo de la práctica se pretenden cubrir los siguientes objetivos:

- Diseñar e implementar una solución paralela basada en el paradigma de paralelismo funcional.
- Utilizar tareas de OpenMP para implementar la aplicación de forma que se permita un paralelismo asíncrono y flexible en el número de threads.

3. Material Necesario

Para realizar la práctica se proporciona un fichero comprimido `prac2.zip`, mediante la plataforma Moodle. Este archivo incluye:

- `video_task.c`: Fichero con la implementación del algoritmo de filtrado de imagen, que incluye también la lectura del fichero de entrada y la generación del fichero de salida. Este fichero se puede modificar todo lo que sea necesario para alcanzar el máximo *speedup* posible.
- `generator.c`: Programa que genera el fichero de entrada `movie.in`.

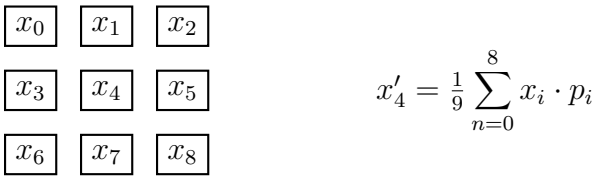

Figura 1: x'_4 es el nuevo valor del píxel x_4, obtenido a partir de la media de sus vecinos.

4. Descripción del Problema

En el campo del procesamiento digital de imágenes se denomina **filtrado de la imagen** a una operación que toma como entrada una imagen y obtiene otra imagen de salida, aplicando a todos los píxeles la misma operación matemática. Esta técnica se utiliza para muchas operaciones como puede ser detectar los bordes de los objetos que hay en una imagen, suavizar la imagen o eliminar ruido de fondo.

La implementación de un algoritmo de filtrado consiste en hacer una convolución con una *matriz de máscara*. Para cada filtro se define una máscara que es una matriz de 3×3 o 5×5, cuyos valores se multiplican por los correspondientes píxeles, para obtener el resultado del píxel de la imagen filtrada. Con esta máscara se recorren todos los píxels de la imagen de entrada y para cada uno se obtiene un nuevo valor. El píxel filtrado se obtiene haciendo el sumatorio del producto de cada elemento de la máscara por el píxel de la imagen correspondiente, y dividiendo por una constante que depende de cada filtro. Un ejemplo sencillo, se puede ver en la figura 1. En esta figura los valores x_i representan los elementos de la máscara, mientras que los p_i, representan el valor de cada píxel de la imagen. Cambiando los valores de estas máscaras se pueden obtener diferentes operaciones de filtrado sobre la imagen de entrada.

En este caso concreto el fichero de entrada contiene un vídeo compuesto por una serie de imágenes con una resolución de 1920×1440 píxeles. Sobre cada imagen individual se aplica un filtro de Gauss, almacenándose el resultado en un fichero de salida diferente. El *Filtro de Gauss* es un filtro de paso bajo que aproxima a una distribución gaussiana.

5. Datos de Entrada y Resultados de Salida

La datos de entrada para este programa consisten en un fichero que contiene un vídeo, compuesto de un conjunto de imágenes. La carga de trabajo del programa la determinan por tanto el número de imágenes y su resolución. Las imágenes están compuestas por un conjunto de píxeles, representados por un número entero de 32 bits. Estos 32 bits corresponden a los tres colores primarios del formato RGB (rojo, verde y azul), ocupando un byte cada uno de ellos. El último byte corresponde a información de transparencia para el modelo de iluminación.

El fichero de entrada es binario y tiene el siguiente formato: las dos primeras palabras de 32 bits representan el **ancho** y **alto** de la imagen respectivamente. El resto de los datos corresponden a los píxeles de las imágenes que aparecen de forma contigua, sin ninguna señal que diferencie una imagen de la siguiente. El fichero de entrada debe

llamarse `movie.in`. Este fichero se puede generar con el programa `generator.c`. Este programa debe compilarse (`gcc -o generator generator.c`) y ejecutarse para generar el fichero de datos de entrada. Este programa tiene tres variables que definen el vídeo:

- `width`: Determina el ancho de las imágenes. Se inicializa en la línea 30 del programa. Por defecto el valor es 1920.
- `height`: Determina el alto de la imagen. Se inicializa en la línea 31 del programa. Por defecto el valor es 1440.
- `max`: Número de imágenes del vídeo. Se inicializa en la línea 39 del programa. Por defecto el valor es 80.

Es muy importante comprobar que los resultados obtenidos por el programa paralelo, coinciden con los obtenidos por la versión secuencial. En la fase de depuración del programa se aconseja definir una carga de trabajo pequeña, con unas pocas imágenes. Para determinar la ganancia de tiempos de ejecución se deben dejar los valores que aparecen en el programa por defecto. Los resultados del filtrado quedan almacenados en el fichero: `movie.out`

6. Desarrollo

NOTA: Ejecución: El trabajo de esta práctica debe realizarse completamente en el directorio local `/tmp`. En caso contrario, se estará trabajando con el disco remoto compartido, el cual tiene un tiempo de acceso muy elevado que tendrá un fuerte impacto sobre el rendimiento final del programa.
NOTA: Verificación de Resultados En todos los ejercicios es muy importante comprobar que los resultados obtenidos en la versión paralela son correctos. Para ello, debes almacenar los resultados de la versión secuencial y de la paralela en diferentes ficheros y compararlos con el comando `diff` de linux. Si no lo conoces, puedes consultar su manual (`man diff`).

Ejercicio 1

Paraleliza el programa secuencial `video_task.c` siguiendo el paradigma de **paralelismo funcional asíncrono**, utilizando las directivas del compilador y/o funciones de *runtime* de OpenMP. El thread maestro debe ir creando tareas que ejecutarán el resto del threads del equipo. Cada thread debe imprimir en pantalla el número de la imagen que filtra. **Es muy importante que garantices que la lectura y escritura de las imágenes del ficheros respeta el orden del vídeo**. Implementa esto de dos maneras diferentes.

1. Justifica el etiquetado de las variables.

2. Explica cómo se consigue el paralelismo en este programa en cada una de las implementaciones.
3. Calcula la ganancia (speedup) obtenido con la paralelización para todo el programa en cada implementación y explica los resultados obtenidos y sus diferencias, de haberlas.
4. Explica cómo realizarías paralelizaciones equivalentes utilizando la librería `threads` y compara el esfuerzo que requerirían.

7. Evaluación

La evaluación de estas prácticas se realizará mediante una única entrega, que tendrá como fecha límite el día **5 de Diciembre de 2023.** Esta entrega constará de una memoria en formato `pdf` siguiendo la plantilla proporcionada. En esta memoria se recogerán los resultados de todos los ejercicios propuestos en las diferentes prácticas. Es fundamental **explicar los resultados y conclusiones obtenidas**, ya que en caso contrario se penalizará fuertemente la nota. Adicionalmente, se deben entregar los ficheros con el código fuente de los programas desarrollados. La entrega del informe y los programas se realizará en un único fichero comprimido en `zip`, a través de la plataforma Moodle.