Programación Concurrente

Trabajo Práctico

Se desea implementar en Java un programa que aplique *filtros de convolución* a imágenes. Para esto, vamos a pensar en la representación de una imagen I como una grilla de $n \times m$ *píxeles*, donde en cada uno se almacenan valores enteros sin signo de 8 bits (es decir, entre 0 y 255), que denotan el valor de ese píxel en distintos *canales* (rojo, verde, azul). Cuando una imágen posee un solo canal, está en *escala de grises*, y en tal caso el valor (único) del pixel marca la intensidad del tono de gris en esa posición, siendo 0 negro y 255 blanco.

El concepto de *filtro* consiste en aplicar alguna transformación a los píxeles de una imágen. En particular, en un *filtro de convolución*, se define un *kernel*, que es una matriz cuadrada de tamaño impar (Para este TP, de 3×3) de números racionales (no necesariamente enteros), y lo *aplica* a cada píxel $I_{i,j}$ de la imágen. La aplicación consiste en tomar los vértices vecinos a $I_{i,j}$, multiplicarlos uno a uno por el valor en la celda correspondiente del kernel, sumarlos y escrbir el resultado en una nueva imagen $I'_{i,j}$ como se ve en la Figura 1. Concretamente, la operación de convolución puede definirse como:

$$\begin{aligned} I'_{i,j} = & I_{i-1,j-1}k_{1,1} + I_{i-1,j}k_{1,2} + I_{i-1,j+1}k_{1,3} + \\ & I_{i,j-1}k_{2,1} + I_{i,j}k_{2,2} + I_{i,j+1}k_{2,3} + \\ & I_{i+1,j-1}k_{3,1} + I_{i+1,j}k_{3,2} + I_{i+1,j+1}k_{3,3} \end{aligned}$$

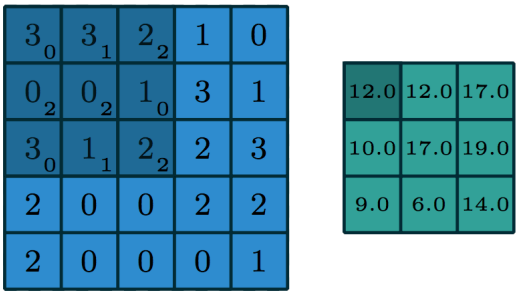

Figura 1: Operación de convolución donde el kernel k de 3×3 tiene las filas $(0, 1, 2)$, $(2, 2, 0)$ y $(0, 1, 2)$ y se aplica sobre una imagen I de 5×5 en escala de grises (grilla azul). La grilla verde es el resultado de la convolución, I'. En particular, en verde oscuro se marca el píxel superior derecho, cuyo valor 12 surge a partir de la subgrilla de 3×3 en el borde superior izquierdo de la grilla azul, por medio de la cuenta $3 + 4 + 1 + 4$.

Es importante notar que no es posible calcular la aplicación del kernel a los bordes. Existen diversas formas de sortear este inconveniente, por ejemplo la de considerar que los píxeles por fuera de los límites de la imagen valen 0 o que poseen el mismo valor que el borde. En este TP, como se ve en el ejemplo de la Figura 1, optaremos directamente por generar una imágen de menor tamaño.

Se conocen diversos kernels de 3×3 que producen efectos reconocibles en una imágen. Como ejemplo, dejamos algunos de los más populares.

Blur: Asigna a cada pixel el promedio de su vecindad, generando un efecto de desenfoque. Se puede obtener un *box blur* aplicando el kernel:

$$k_{\text{boxBlur}} = 1/9 \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Laplaciano: Detecta bordes, regiones donde la intensidad cambia muy rápidamente. Se puede obtener aplicando el kernel (se suele mapear el 0 a 128 para mostrar el resultado final más claramente):

$$k_{\text{laplacian}} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Filtro Sobel: Efecto que aísla los bordes verticales y horizontales de la imágen, similar al Laplaciano. Se obtienen aplicando los siguientes kernels

$$k_{\text{sobel horizontal}} = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad k_{\text{sobel vertical}} = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Existen más filtros que se pueden aplicar, algunos pueden verse en las siguientes páginas [1]

Objetivo del Trabajo Práctico

Se pide implementar un programa en Java que tome un directorio que contendrá una colección de imágenes (también puede pensarse como secuencias que representan los fotogramas de un video) en formato `jpg`[2], ya sea en color o en escala de grises, y le aplique un filtro convolucional configurable de 3×3 a cada una de ellas. La aplicación de los filtros se hará concurrentemente (distintos threads se deben estar encargando de aplicar el mismo filtro a distintas imagenes). Una vez terminado el procesamiento, se debe generar un nuevo archivo con las imagenes ya filtradas e informar el tiempo que se demoró en realizar todo el proceso.

Elementos a entregar

Programa Java

Se debe presentar un programa escrito en Java con la siguiente estructura:

1. Una clase `Main` con el punto de entrada del programa, que tiene la responsabilidad de inicializar las estructuras mencionadas a continuación y producir unidades de trabajo consistentes para que múltiples threads puedan filtrar imagenes concurrentemente. El programa debe terminar únicamente cuando se hayan generado todas las imagenes de salida, y debe informar el tiempo transcurrido desde el inicio del proceso de filtrado.

2. Una clase `Buffer` (implementada como un monitor utilizando métodos synchronized) que actúa como una cola FIFO concurrente de capacidad acotada. Es decir, bloquea a un lector intentando sacar un elemento cuando está vacía y bloquea a un productor intentando agregar un elemento cuando está llena. La capacidad del Buffer también debe ser un parámetro configurable desde la clase `Main`.

3. Una clase `WorkerCounter` (implementada como un monitor utilizando métodos synchronized) que evita que `Main` termine su ejecución mientras queden threads trabajando.

4. Una clase `Task` que implementa `Runnable`, cuyo método `run` realiza la convolución a partir de un kernel y una imagen dada y escribe su salida en disco. Aunque se sugiere el esquema anterior, es posible, a su vez, dividir a cada imagen en subregiones achicando así el procesamiento asociado a cada tarea.

5. Una clase `FilterWorker` que hereda de `Thread` y que, por siempre, resuelve las tareas disponibles en el buffer (obtenido en su creación).

6. Una clase `PoisonPill` que hereda de `Task` y que hace que el `FilterWorker` que la tome termine su ejecución (puede ser por medio de una excepción, como en la práctica).

[1] `https://www.taylorpetrick.com/blog/post/convolution-part3`
`https://docs.gimp.org/2.6/en/plug-in-convmatrix.html`
`http://www.math.utah.edu/~gustafso/s2017/2270/projects-2017/asherSorensen/AsherSorensenLinear.pdf`ç

[2] En rigor podría hacerse compatible con otros formatos con pocos cambios, pero algunos, por ejemplo `png`, tienen un canal *alpha* que representa transparencia, en el que no habría que aplicar los filtros

7. Una clase `ThreadPool`, que se encarga de instanciar e iniciar la cantidad de `FilterWorkers` pedida por un usuario asignándoles el mismo Buffer.

8. Cualquier otra clase que considere necesaria.

Al iniciar el programa la clase `Main` debe delegar la iniciación de los threads necesarios en la clase `ThreadPool` y luego introducir de a una la tareas a procesar en el `Buffer`. Cada `FilterWorker` en funcionamiento debe tomar regiones de a una del `Buffer` y aplicar el filtro. Cabe destacar que es inadmisible utilizar una cantidad de threads menor a la solicitada por el usuario.

El programa, además, debe estar correctamente documentado. En particular, debe contener las instrucciones para configurar el archivo de entrada, el nombre del archivo de salida, el tamaño del buffer, la cantidad de `FilterWorker`s y la matriz correspondiente al filtro. Idealmente hacer que el programa se pueda ejecutar por línea de comandos y tome estas opciones por parámetro o por entrada estándar.

No es incorrecto que el programa haga impresiones por pantalla intermedias para informar su progreso, pero se pide evitar la impresión de mensajes de *debug*.

Informe

Además del código, se requiere un informe corto en formato `pdf`, donde se debe mostrar, principalmente, experimentos relacionados al análisis del tiempo de ejecución de su programa.

El objetivo principal es entender cuanto beneficio se les está extrayendo al uso de la concurrencia, y para eso se busca entender el comportamento en términos de tiempo de ejecución en función de la cantidad de imágenes a filtrar, la cantidad de Threads utilizados y el tamaño del buffer.

Para lo anterior se sugiere considerar un tamaño de imagen fijo y los siguientes valores:

- **Imagenes:** 1, 10, 100, 1000
- **Threads:** 1, 2, 4, 8 y 16
- **Tamaño del Buffer:** 2, 8, 32

Para cada combinación se deberán considerar realizar un conjunto de mediciones suficientes para poder tener una estimación algo más robusta del tiempo de ejcución (notar que las mediciones estarán sujetas a ruido proveniente del resto de los procesos corriendo del sistema, de la JVM, del estado de la memoria Cache, etc). De cada conjunto de mediciones se deberá reportar, al menos, alguna medida de centralidad (como el promedio o la mediana) e, idealmente, dispersión (como el desvío estándar).

El informe, que es **condición necesaria para la aprobación del TP**, debe respetar el siguiente formato:

1. **Autoría:** Nombres, e-mail y número de legajo de quienes hicieron el trabajo (máximo 3 personas salvo excepciones acordadas con los docentes).

2. **Introducción:** Sección explicando el dominio del TP, el diseño del código y cualquier consideración adicional que considere relevante para la correcta interpretación de los resultados.

3. **Evaluación:** Sección explicando el proceso de evaluación, incluyendo los detalles del hardware donde se ejecutan las pruebas (modelo de microprocesador con cantidad de núcleos, memoria disponible y versión del sistema operativo). En esta

sección deben incluirse tres tablas reportando los tiempos en los que se aplicaron los filtros para cada combinación de cantidad de threads y tamaño de buffer. Cada tabla se debe corresponder a experimentos realizados con distintas cantidades de imagenes (elegir los 3 más interesantes). Por otro lado se deben incluir 2 gráficos: El primero deberá ser para el tamaño de buffer 8, que en el eje x tenga las distintas cantidades de threads, y en el y el tiempo de ejecución, con una curva por cada valor de cantidad de imágenes considerados. El segundo, deberá ser para la cantidad de 8 threads, y tener en el eje x los distintos tamañaos de buffer (una vez más, en el eje y estarán los tiempos de ejecuión y deberá haber una curva por valor de cantidad de imágenes).

4. **Análisis:** Sección en la que se debe discutir los datos obtenidos en la Evaluación. Por ejemplo, discutir los resultados con las siguientes preguntas disparadoras: ¿Es verdad que aumentar la cantidad de threads siempre mejora el rendimiento? ¿Cuánto influye el tamaño del buffer? ¿Y la cantidad de imágenes? ¿Los valores de cantidad de threads y tamaño de buffer óptimos cambian con la cantidad de imágenes? ¿Puede establecerse alguna relación entre los valores óptimos y las características del hardware donde corrieron las pruebas?. En última instancia, se debe destacar la combinación de parámetros con la que se obtuvo el tiempo de ejecución mínimo y la combinación que obtuvo el tiempo máximo.

Forma de Entrega

Se deben enviar el archivo `.zip` con el código del programa, los archivos `jpg` con las imágenes utilizads, y el `.pdf` con el informe por email a las casillas de correo electrónico de *ambos* profesores con el subject:
`Entrega TP PCONC 1S2024 - (apellido1) - (apellido2) - (apellido3)`
(donde `apellido1`, `apellido2` y `apellido3` son los apellidos de las personas que constituyen el grupo). El mensaje debe ir *con copia* a las otras persona que integre el grupo, pues la devolución del TP se hará en *respuesta a todos* ese correo.

Es posible trabajar en un repositorio *git* compartido por las personas que compongan el grupo. En este caso, el repositorio debe ser **privado**. Al entregar, se deberá agregar a los profesores al repositorio. De todos modos se solicita que envíen el mail con las condiciones especificadas, aunque en vez de tener los archivos adjuntos incluirán el link al respositorio. Se aclara que se espera que el código entregado por el grupo sea **original**. Si se detecta que el mismo fue copiado de alguna fuente online será automáticamente desaprobado sin posibilidad de reentrega.

Fecha de Entrega

El TP debe entregarse antes del **Sábado 22 de Junio** a las **23:59hs**. En caso de ser necesario, se cuenta con una fecha de reentrega el **Sábado 6 de Julio**.

Apéndice: Algunas herramientas de Java

Manipulación de imágenes

Para la manipulación de imágenes píxel a píxel que se require en este TP, se recomienda utilizar las clases `ImageIO`, `BufferedImage` y `WritableRaster`. En particular, el siguiente código:

```
File imagen = new File("entrada.png");
```

```
BufferedImage bi = ImageIO.read(imagen);
WritableRaster raster = bi.getRaster();
```

Permite obtener un objeto de la clase `WritableRaster` al cual se le puede pedir ancho, altura y cantidad de canales (con `getWidth()`, `getHeight()` y `getNumBands()`), además del valor del píxel en la posición (x, y) y canal k particular con `origen.getSample(x, y, k)`. De la misma manera, es posible setear el valor de un pixel del `WritableRaster` utilizando el método `setSample(x, y, k, valor)`, siendo `valor` su nuevo valor. **Nota importante:** Cuando se hace `setPixel` utilizando un valor no entero (por ejemplo (`double`), este es redondeado al valor entero más cercano, y es correcto. Sin embargo, si se le pasa un valor fuera del rango $[0 - 255]$, que es el que puede tomar un píxel, toma el resto módulo 255 (por ejemplo, 300 se guarda como 45). Esto no es correcto: Los valores menores a 0 deberían guardarse como 0 y los mayores a 255, como 255.

Para generar un `WritableRaster` nuevo e inicializarlo, se puede hacer:

```
int ancho = raster.getWidth();
int alto = raster.getHeight();
int canales = raster.getNumBands();
WritableRaster raster2 = raster.createCompatibleWritableRaster(
    ancho, alto));
wr_destino.setPixels(0, 0, ancho, alto,
    new double[ancho*alto*canales]);
```

Finalmente, para generar una imágen de salida, la clase `BufferedImage` provee un constructor que toma un `WritableRaster`. Se puede hacer:

```
BufferedImage bi_salida = new BufferedImage(
    bi.getColorModel(), raster2, bi.isAlphaPremultiplied(), null);
File outputfile = new File("salida.jpg");
ImageIO.write(outputImage, "jpg", outputfile);
```

(Notar que la construcción de `bi_salida` requiere algunos valores que se deben tomar del `bi` original)

Conversión de imágenes a video

Es posible extraer los fotogramas de un video a formato .jpg utilizando la herramienta ffmpeg desde consola:

```
ffmpeg -i video.mp4 -vf fps=30 ./imgs/out%d.jpg
```

Luego, dado el directorio de outputs, es posible unir todos los fotogramas filtrados haciendo:

```
ffmpeg -f image2 -r 30 -i ./outputs/out%d.jpg video.mov
```

Medición de tiempos

Para medir el tiempo de ejecución del programa, se puede llamar al método provisto por Java `System.currentTimeMillis()`. Se debe guardar el tiempo actual cuando se va a comenzar a operar (si se toman parámetros por entrada estándar, debe ser después), y cuando se terminó la ejecución (es decir, luego de que los Workers hayan parado). Luego, basta con hacer la diferencia entre ambos tiempos y dividirla por mil para obtener el tiempo demorado expresado en segundos.