



INFORME DEL TALLER 5

MAYRA ALEJANDRA SANCHEZ SALINAS (2040506)

FUNDAMENTOS DE PROGRAMACIÓN FUNCIONAL Y CONCURRENTE

JUAN FRANCISCO DIAZ FRIAS

**UNIVERSIDAD DEL VALLE
SANTIAGO DE CALI, VALLE DEL CAUCA
2022**

Filtro de desenfoque paralelo

El filtro de desenfoque entrega una imagen de salida en la cual cada pixel tiene un valor promedio de los píxeles vecinos en la imagen original. El filtro de desenfoque es un ejemplo de un problema fuertemente paralelo (es decir, se necesita muy poco esfuerzo para separarlo en tareas paralelas). Cada pixel de la imagen de salida, se puede calcular independientemente de los otros pixeles y en paralelo.

Preliminares

Primero se define el tipo RGBA para denotar los posibles valores que puede tomar un pixel de una imagen. Se trabajará con imágenes donde cada pixel se representa con un entero de 32 bits.

```
/** El valor de cada pixel es representado con un entero de 32 bits. */
type RGBA = Int
```

Se llama RGBA porque cada pixel está compuesto de cuatro componentes que representan el contenido de cada color básico en el color del pixel (R, de rojo, G, de verde y B de azul) y el nivel de transparencia del pixel (A de alpha). A cada uno de estos componentes los denominan canales. El valor de cada canal es un número entre 0 y 255, es decir, es un número que se codifica con 8 bits. Por eso 32 bits son suficientes para encapsular estos cuatro componentes. Los primeros 8 bits del entero guardan el valor correspondiente a la presencia del color rojo en el pixel, los siguientes 8 bits guardan el valor correspondiente a la presencia del color verde en el pixel, los siguientes 8 bits guardan el valor correspondiente a la presencia del color azul en el pixel, y los últimos 8 bits guardan el valor correspondiente al nivel de transparencia (alpha) del píxel.

Cada uno de esos componentes de un pixel pueden ser extraídos vía los siguientes métodos auxiliares:

```
/** Devuelve el componente de rojo */
def rojo(c: RGBA): Int = (0xff000000 & c) >>> 24

/** Devuelve el componente de verde */
def verde(c: RGBA): Int = (0x00ff0000 & c) >>> 16

/** Devuelve el componente de azul */
def azul(c: RGBA): Int = (0x0000ff00 & c) >>> 8

/** Devuelve el componente alpha */
def alpha(c: RGBA): Int = (0x000000ff & c) >>> 0
```

Se puede calcular el valor de tipo RGBA que representa ese pixel:

```
/** Usado para crear un valor RGBA a partir de sus componentes */
def rgba(r: Int, g: Int, b: Int, a: Int): RGBA = {
    (r << 24) | (g << 16) | (b << 8) | (a << 0)
}
```

Se define el tipo para manipular imágenes:

```
/** La imagen es un arreglo de dos dimensiones de valores de pixeles */
class Img(val ancho: Int, val alto: Int, private val datos: Array[RGBA]) {
    def this(an: Int, al: Int) = this(an, al, new Array(an*al))
    def apply(x: Int, y: Int): RGBA = datos(y * ancho + x)
    def update(x: Int, y: Int, c: RGBA): Unit = datos(y * ancho + x) = c
}
```

Para calcular coordenadas x e y de un pixel, y sea necesario asegurar que ellas quedan dentro de los límites de la imagen, se puede usar el método cercar a continuación:

```
/** Restringe el entero al rango especificado */
def cercar(v: Int, min: Int, max: Int): Int = {
    if (v < min) min
    else if (v > max) max
    else v
}
```

El núcleo del filtro de desenfoque paralelo

La función desenfoqueNuclear recibe una imagen fuente f te, las coordenadas x y y de un pixel y el radio del desenfoque, y devuelve el valor RGBA del pixel desenfocado, el cual corresponde, componente por componente del pixel, al valor promedio de los componentes de los pixeles vecinos en un radio alrededor del pixel

```

/** Calcula el valor RGBA del pixel desenfocado correspondiente a un pixel de la imagen de entrada.
def desenfoqueNuclear(fte: Img, x: Int, y: Int, radio: Int): RGBA ={
    val minX = cercar(x - radio, min = 0, fte.ancho - 1)
    val maxX = cercar(x + radio, min = 0, fte.ancho - 1)
    val minY = cercar(y - radio, min = 0, fte.alto - 1)
    val maxY = cercar(y + radio, min = 0, fte.alto - 1)
    var accX = minX
    var accY = minY
    var pixelProcessed = 0
    var accR = 0
    var accG = 0
    var accB = 0
    var accA = 0
    while(accX <= maxX) {
        while(accY <= maxY) {
            val pixel = fte.apply(accX, accY)
            accR += rojo(pixel)
            accG += verde(pixel)
            accB += azul(pixel)
            accA += alpha(pixel)
            pixelProcessed += 1
            accY += 1
        }
        accX += 1
        accY = minY
    }
    rgba(accR / pixelProcessed, accG / pixelProcessed, accB / pixelProcessed, accA / pixelProcessed)
}

```

Implementando el filtro de desenfoque

Desenfoque por bandas verticales

La implementación de desenfoque debe basarse en la implementación de desenfoqueNuclear, la función desenfoquePar que se encuentra en el archivo DesenfoqueVertical.scala, la cual toma una imagen fuente fte, una imagen destino dst (sin valores iniciales), una valor numT áreas indicando en cuantas tareas en paralelo se desea hacer el cálculo y un radio indicando el radio del desenfoque nuclear, y realiza el desenfoque de la imagen en paralelo, dividiendo la imagen en tantas bandas verticales como numTareas se especifiquen, y creando numTareas, cada una encargada de realizar el desenfoque de la respectiva banda.

- Desenfoque

```

def desenfoque(fte: Img, dst: Img, inicial: Int, fin: Int, radio: Int): Unit = {
    for(
        row <- inicial until fin;
        column <- 0 until fte.alto;
        if row >= 0 && row < fte.ancho
    ) yield {
        dst.update(row, column, desenfoqueNuclear(fte, row, column, radio))
    }
    // POR HACER: implemente esta funcion usando el la funcion `desenfoqueNuclear`
}

```

- DesenfoquePar

```

def desenfoquePar(fte: Img, dst: Img, numTareas: Int, radio: Int): Unit = {
    // POR HACER: implemente usando el constructor 'task' y la funcion 'desenfoque'
    //for (i <- 0 until fte.ancho; j <- 0 until fte.alto) dst.update(i,j, fte(i,j))

    val separateStrips = 0 to fte.ancho by (fte.ancho / numTareas max 1)

    separateStrips.zip(separateStrips.tail)
        .map { case (from, end) =>
            task[Unit] {
                desenfoque(fte, dst, from, end, radio)
            }
        }
        .foreach(_.join())
}

```

Corriendo el algoritmo de desenfoque vertical:

```

def main(args: Array[String]): Unit = {
    val radio = 6
    val ancho = 1920
    val alto = 1080
    val fte = new Img(ancho, alto)
    val dst = new Img(ancho, alto)
    val seqtime = standardConfig measure {
        DesenfoqueVertical.desenfoque(fte, dst, inicial = 0, ancho, radio)
    }
    println(s"tiempo de desenfoque secuencial: $seqtime ms")

    val numTareas = 32
    val partime = standardConfig measure {
        DesenfoqueVertical.desenfoquePar(fte, dst, numTareas, radio)
    }
    println(s"tiempo de desenfoque paralelo: $partime ms")
    println(s"aceleracion: ${seqtime.value / partime.value}")
}

```

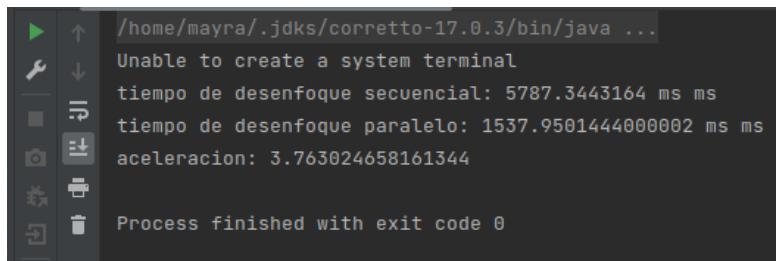
```

/home/mayra/.jdks/corretto-17.0.3/bin/java ...
Unable to create a system terminal
tiempo de desenfoque secuencial: 5364.7959664 ms ms
tiempo de desenfoque paralelo: 1468.8221227 ms ms
aceleracion: 3.652447688177784

Process finished with exit code 0

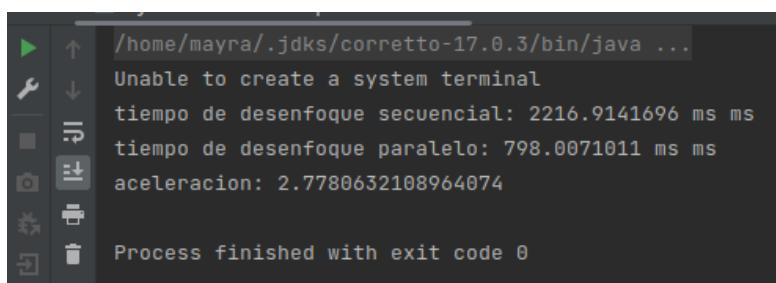
```

Si por ejemplo cambiamos el valor del radio a 6 y el número de tareas a 200, los resultados que se arrojaran serán los siguientes:



```
/home/mayra/.jdks/corretto-17.0.3/bin/java ...
Unable to create a system terminal
tiempo de desenfoque secuencial: 5787.3443164 ms ms
tiempo de desenfoque paralelo: 1537.9501444000002 ms ms
aceleracion: 3.763024658161344
Process finished with exit code 0
```

Si por ejemplo cambiamos el valor del radio a 3 y el número de tareas a 100, los resultados que se arrojaran serán los siguientes:



```
/home/mayra/.jdks/corretto-17.0.3/bin/java ...
Unable to create a system terminal
tiempo de desenfoque secuencial: 2216.9141696 ms ms
tiempo de desenfoque paralelo: 798.0071011 ms ms
aceleracion: 2.7780632108964074
Process finished with exit code 0
```

Desenfoque por bandas horizontales

Consiste en dividir la imagen en un número fijo de bandas horizontales del mismo alto, por cada banda horizontal, se lanza una tarea en paralelo de cálculo secuencial de los píxeles dentro de la banda. Los pixeles se recorren de izquierda a derecha, y de arriba a abajo, calculando el píxel resultante de invocar desenfoqueNuclear sobre cada pixel.

La implementación de desenfoque debe basarse en la implementación de desenfoqueNuclear, a continuación implemente la función desenfoquePar, la cual toma una imagen fuente f te, una imagen destino dst (sin valores iniciales), una valor $numTareas$ indicando en cuántas tareas en paralelo se desea hacer el cálculo y un radio indicando el radio del desenfoque nuclear, y realiza el desenfoque de la imagen en paralelo, dividiendo la imagen en tantas bandas horizontales como $numTareas$ se especifiquen, y creando $numTareas$, cada una encargada de realizar el desenfoque de la respectiva banda.

- Desenfoque

```

def desenfoque(fte: Img, dst: Img, inicial: Int, fin: Int, radio: Int): Unit = {
    // POR HACER: implemente esta funcion usando la funcion 'desenfoqueNuclear'

    for (
        x <- 0 until fte.ancho;
        y <- inicial until fin;
        if y >= 0 && y < fte.alto
    ) yield {
        dst.update(x, y, desenfoqueNuclear(fte, x, y, radio))
    }
}

```

- DesenfoquePar

```

def desenfoquePar(fte: Img, dst: Img, numTareas: Int, radio: Int): Unit = {
    // POR HACER: implemente usando el constructor 'task' y la funcion 'desenfoque'
    //for (i <- 0 until fte.ancho; j <- 0 until fte.alto) dst.update(i,j, fte(i,j))

    val separateStrips = 0 to fte.alto by (fte.alto / numTareas max 1)

    separateStrips.zip(separateStrips.tail)
        .map { case (from, end) =>
            task[Unit] {
                desenfoque(fte, dst, from, end, radio)
            }
        }
        .foreach(_.join())
}

```

Corriendo el algoritmo:

```

def main(args: Array[String]): Unit = {
    val radio = 3
    val ancho = 1920
    val alto = 1080
    val fte = new Img(ancho, alto)
    val dst = new Img(ancho, alto)
    val seqtime = standardConfig measure {
        DesenfoqueHorizontal.desenfoque(fte, dst, inicial = 0, alto, radio)
    }
    println(s"tiempo de desenfoque secuencial: $seqtime ms")
}

val numTareas = 128
val partime = standardConfig measure {
    DesenfoqueHorizontal.desenfoquePar(fte, dst, numTareas, radio)
}
println(s"tiempo de desenfoque paralelo: $partime ms")
println(s"aceleracion: ${seqtime.value / partime.value}")
}
}

```

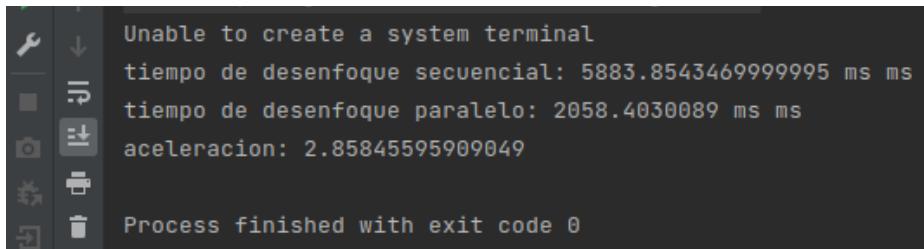
```

/home/mayra/.jdks/corretto-17.0.3/bin/java ...
Unable to create a system terminal
tiempo de desenfoque secuencial: 2018.7426726999995 ms ms
tiempo de desenfoque paralelo: 502.0129836999995 ms ms
aceleracion: 4.021295739845622

Process finished with exit code 0

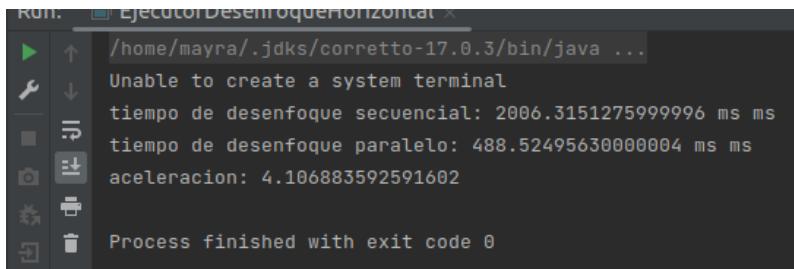
```

Si por ejemplo cambiamos el valor del radio a 6 y el número de tareas a 200, los resultados que se arrojaran serán los siguientes:



```
Unable to create a system terminal
tiempo de desenfoque secuencial: 5883.854346999995 ms ms
tiempo de desenfoque paralelo: 2058.4030089 ms ms
aceleracion: 2.85845595909049
Process finished with exit code 0
```

Si por ejemplo cambiamos el valor del radio a 3 y el número de tareas a 100, los resultados que se arrojaran serán los siguientes:



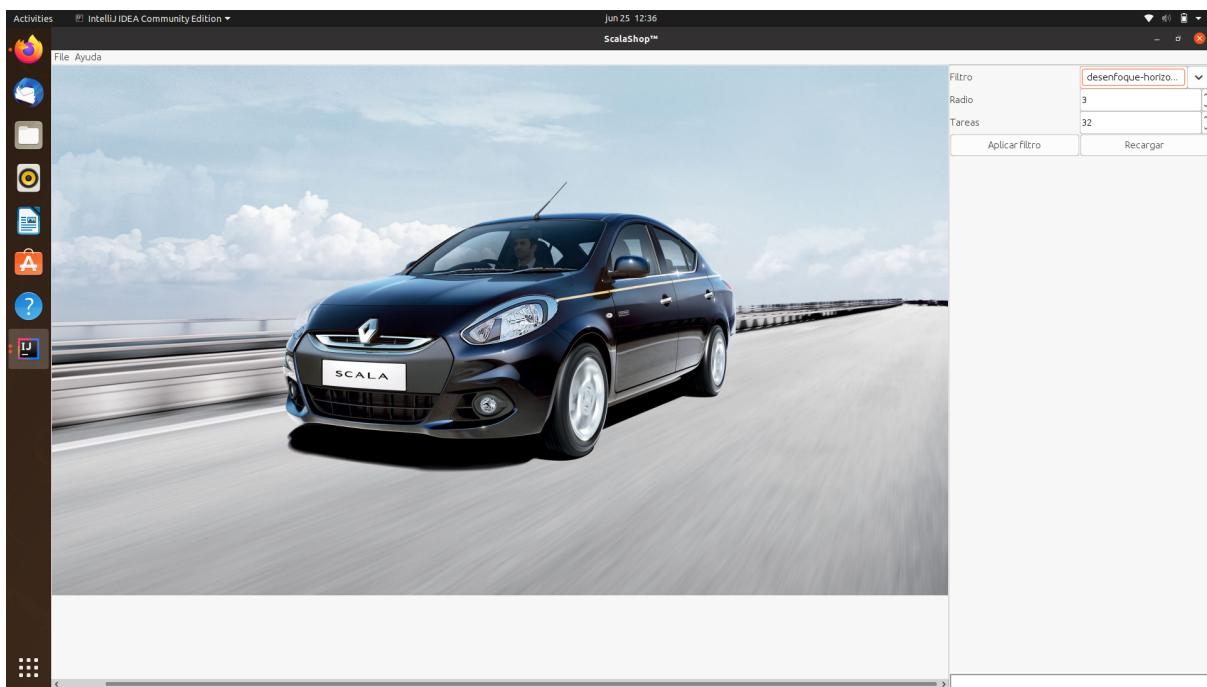
```
EjecutorDesenfoqueHorizontal
/home/mayra/.jdks/corretto-17.0.3/bin/java ...
Unable to create a system terminal
tiempo de desenfoque secuencial: 2006.315127599996 ms ms
tiempo de desenfoque paralelo: 488.52495630000004 ms ms
aceleracion: 4.106883592591602
Process finished with exit code 0
```

Comparando las alternativas de desenfoque paralelo

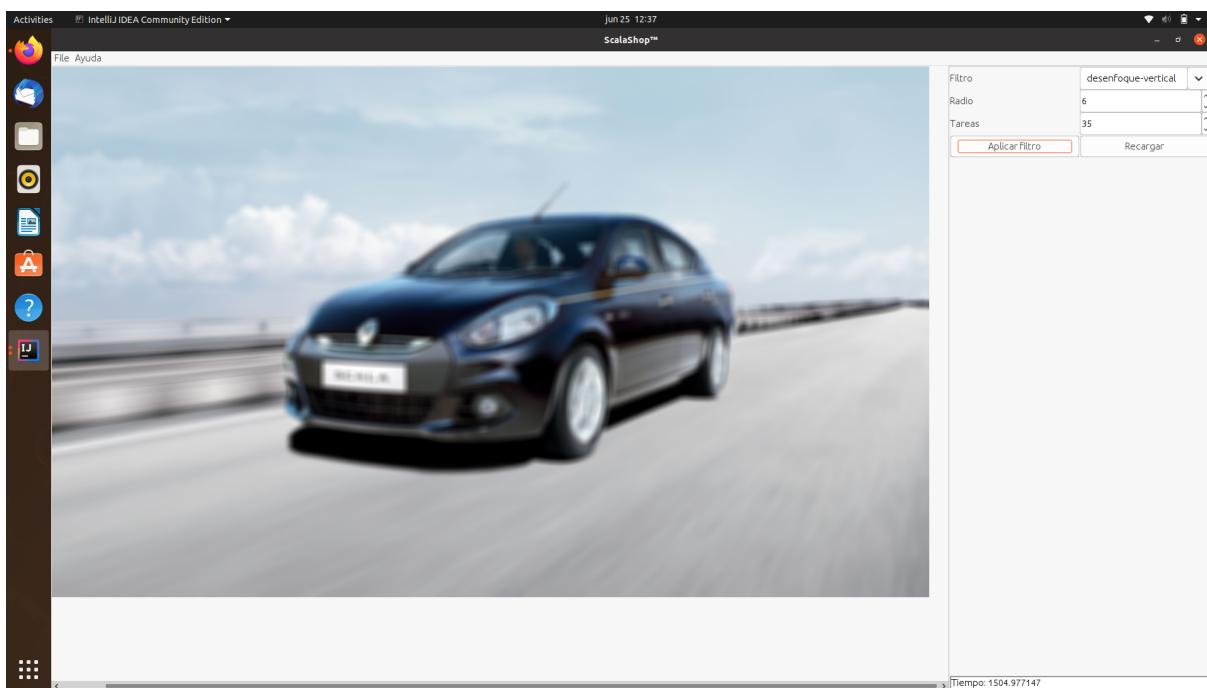
	Radio	Tareas	Desenfoque secuencial	Desenfoque paralelo	Aceleración
Horizontal	6	200	5883.85	2058.40	2.8584
Vertical	6	200	5787.34	1537.95	3.7630
Horizontal	3	100	2006.31	488.52	4.1068
Vertical	3	100	2216.91	798.00	2.7780

Scalashop

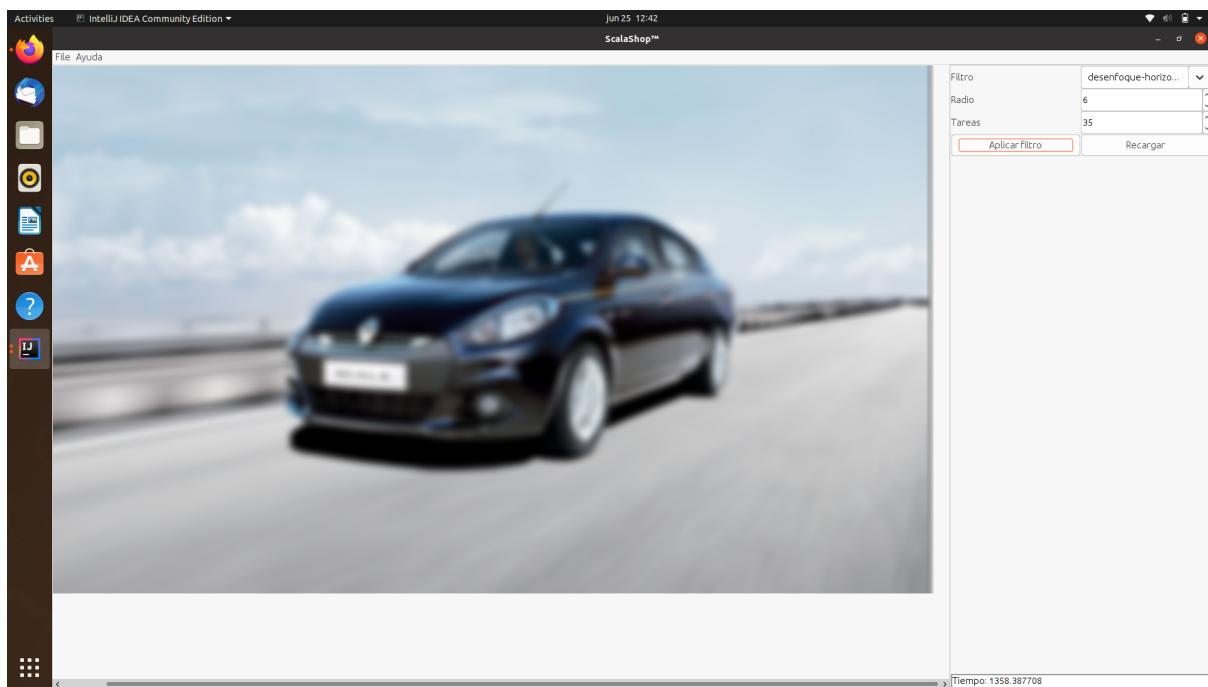
Al escribir en el shell: runMain scalashop.ScalaShop se ve a correr el siguiente programa, donde tendremos la posibilidad de escoger el tipo de filtro que queremos aplicar (desenfoque horizontal o desenfoque vertical) y se debe asignar el radio y las tareas



Por ejemplo así se ve la imagen con el filtro desenfoque vertical con un radio de 6 y un 35 en tareas



Y así se ve la imagen con el filtro desenfoque horizontal con el mismo radio y tareas:



Comparando los dos filtros y con las mismas características para un radio mayor a 6 es más rápido el filtro de desenfoque horizontal mientras que un radio menor a 6 es más rápido el filtro de desenfoque vertical.