TRABAJO PRÁCTICO - CONCURRENTE CONCURCONVOLUTION

Autoría:

Ariel Segovia ariel-110595@hotmail.com
Legajo: 47685
Tobías Romero tobias.romero1711@gmail.com
Legajo: 42806

INTRODUCCIÓN

El proyecto consta de un programa en Java que aplica filtros convolucionales de manera concurrente a una imagen.

Para llevar a cabo esta tarea, en la clase Main te pide mediante inputs ingresar el tamaño del Buffer en el cual se cargaran tareas con las información de los píxeles que los workers irán tomando para ejecutar, la cantidad de Threads a iniciar, la ruta en la que se encuentra la imagen que quiere procesar y el filtro que quiere aplicar, y al finalizar devuelve en una imagen de nombre "salida.jpg" el resultado.

Explicación de las clases creadas en el programa:

- **El Buffer, es un recurso compartido entre los threads generados y lo creamos como un monitor. De esta forma, nos aseguramos el mutex tanto en la escritura(evitando que se "pisen" datos) como en la lectura de datos (evitando la pérdida de tareas para analizar). La estructura del buffer es similar a la de un buffer circular, es decir, funciona con algoritmo FIFO pero teniendo una capacidad limitada para agregar**

datos, teniendo que esperar, si este está lleno, a que se libere alguna posición para insertar uno nuevo. En este Buffer se irán cargando un cantidad de tareas equivalente a la cantidad de pixeles de la foto eliminando los bordes siempre respetando la capacidad del Buffer y además al final se agregaran PoisonPills equivalentes a la cantidad de Workers para que dejen de trabajar en cuanto las consuman.

- **El FilterWorker extiende de Thread, representa cada uno de los Threads que ingresamos por input al iniciar el programa y es el encargado de leer una tarea del Buffer y ejecutarla.**

- **Las PoisonPills se encargan de finalizar el trabajo del FilterWorker y aumentar el WorkerCounter en uno.**

- **El ProductorThreadPool en base a la cantidad de píxeles de la imagen original, le quita los bordes, crea una cantidad de tareas equivalente a los píxeles restantes y las carga en el Buffer. Luego de cargar todas, crea PoisonPills equivalentes a la cantidad de Workers y las carga en el buffer también.**

- **Cada Task (tarea) tiene toda la información necesaria para leer el pixel, aplicarle el filtro y replicar el resultado en la imagen de salida.**

- **El WorkerCounter está implementado como un monitor ya que también es un recurso compartido, y es el encargado de ir controlando que el Main no finalice hasta que todos los Workers terminen, por lo que va llevando un conteo de los workers que van terminando y ,una vez terminados todos, hace el stop.**

- **El ThreadPool crea cada uno de los FilterWolkers en base al valor ingresado por input y comienza la ejecución de cada uno, también crea el ProductorThreadPool y lo ejecuta.**

- **Contamos también con una clase UserPool que solo lanza el ThreadPool y guarda el momento en el que se inició y una clase PoolStopper que ejecuta el stop del WorkerCounter, crea el archivo de imagen resultado de aplicar los filtros (salida.jpg) y compara el momento en que inició el programa guardado en el UserPool con el**

momento en que terminó para mostrar el tiempo que tardó la ejecución.

- **Por último tenemos los filtros guardados como un Enum Filtro que poseen un nombre y la matriz para aplicarlo, permitiendo así fácilmente meter nuevos simplemente agregando un nuevo valor en el código fuente.**

CÓMO EJECUTAR EL PROGRAMA:

Para ejecutar por CLI:

1. **Tener instalado JDK-19, ubicarte en la carpeta src desde la consola y lanzar el comando "java Main".**
2. **El programa iniciará y te pedirá que indiques el tamaño del Buffer, ingresar el valor deseado y presionar enter.**
3. **Ahora te pedirá la cantidad de Workers a iniciar, ingresar el número deseado y presionar enter.**
4. **Seguido a eso te pedirá la ruta absoluta de la imagen que quieras procesar(Ejemplo: C:\Users\Administrator\Desktop\TP_Concurrente\michi.jpg), ingresar la ruta y presionar enter.**
5. **Por último pedirá el filtro que quieras aplicarle, debe ingresar alguno de los siguientes valores sin comillas "blur", "sharpen" , "sobel vertical", "sobel horizontal" , "laplacian" y "emboss". Presionar enter.**
6. **Esperar a que finalice la ejecución y revisar el archivo salida.jpg ubicado en la misma carpeta src.**

Para ejecutar por intelliJ:

1. **Configurar la SDK del proyecto con openjdk-19 de Oracle**
2. **Correr el Main.**
3. **El programa iniciará y te pedirá que indiques el tamaño del Buffer, ingresar el valor deseado y presionar enter.**
4. **Ahora te pedirá la cantidad de Workers a iniciar, ingresar el número deseado y presionar enter.**
5. **Seguido a eso te pedirá la ruta absoluta de la imagen que quieras procesar(Ejemplo: C:\Users\Administrator\Desktop\TP_Concurrente\michi.jpg), ingresar la ruta y presionar enter.**

6. **Por último pedirá el filtro que quieras aplicarle, debe ingresar alguno de los siguientes valores sin comillas “blur”, “sharpen” , “sobel vertical”, “sobel horizontal” , “laplacian” y “emboss”. Presionar enter.**
7. **Esperar a que finalice la ejecución y revisar el archivo salida.jpg ubicado en la misma carpeta del proyecto “TP_Concurrente”.**

EVALUACIÓN:

AMD Ryzen 7 5700G 3.80 GHz, 8 núcleos, 16 hilos(lógicos); 16GB RAM DDR4, 6GB Disponibles a la hora de ejecutar las pruebas.

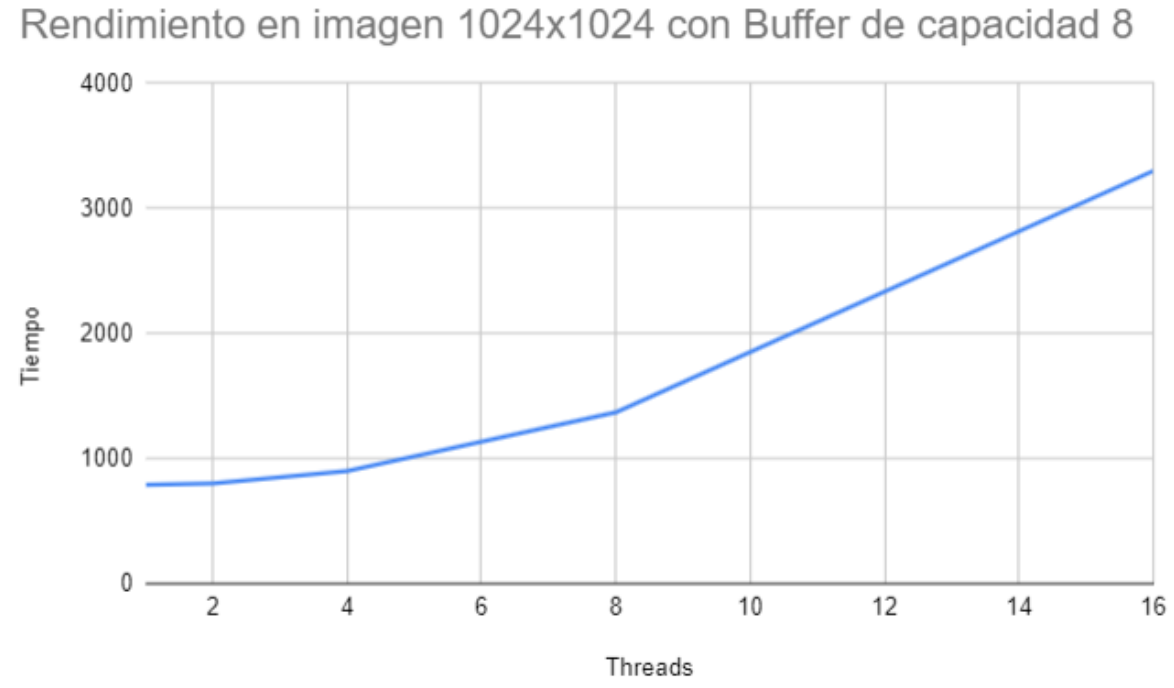
Intel Core i7-11700 @ 2.50GHz, 8 nucleos, 16 hilos(lógicos); 16GB RAM DDR4, 13GB Disponibles a la hora de ejecutar las pruebas.

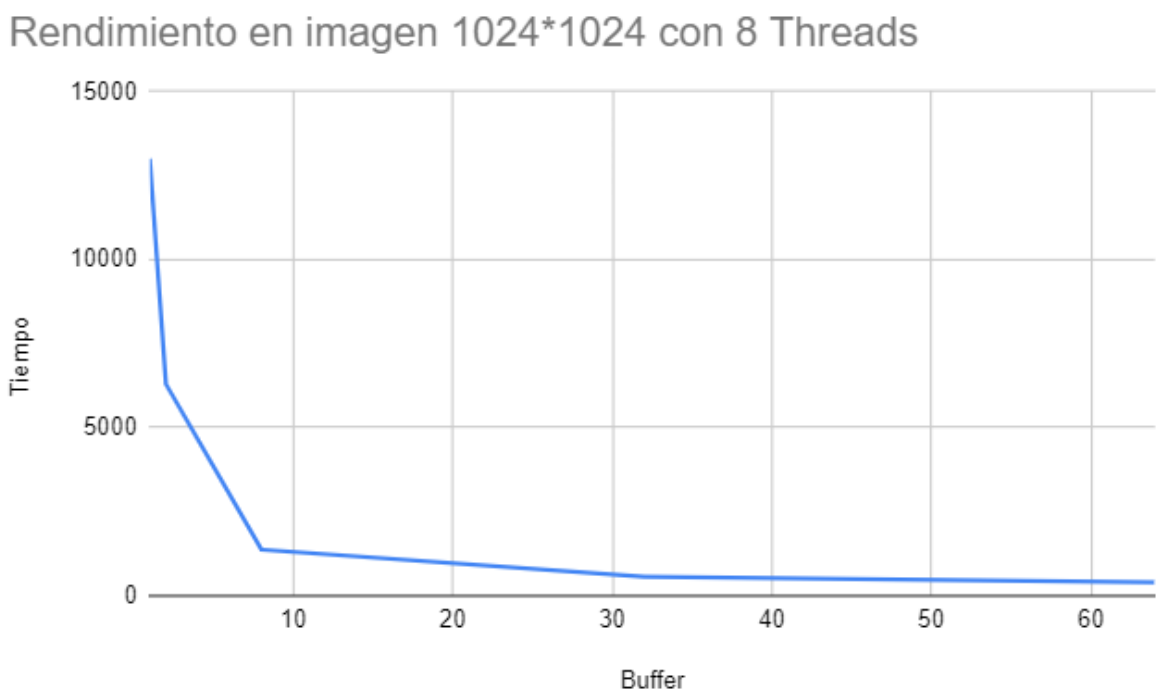
Como ambos equipos daban resultados demasiado cercanos, decidimos hacer los gráficos en base al promedio de sólo uno de ellos ya que no había diferencia que marcar.

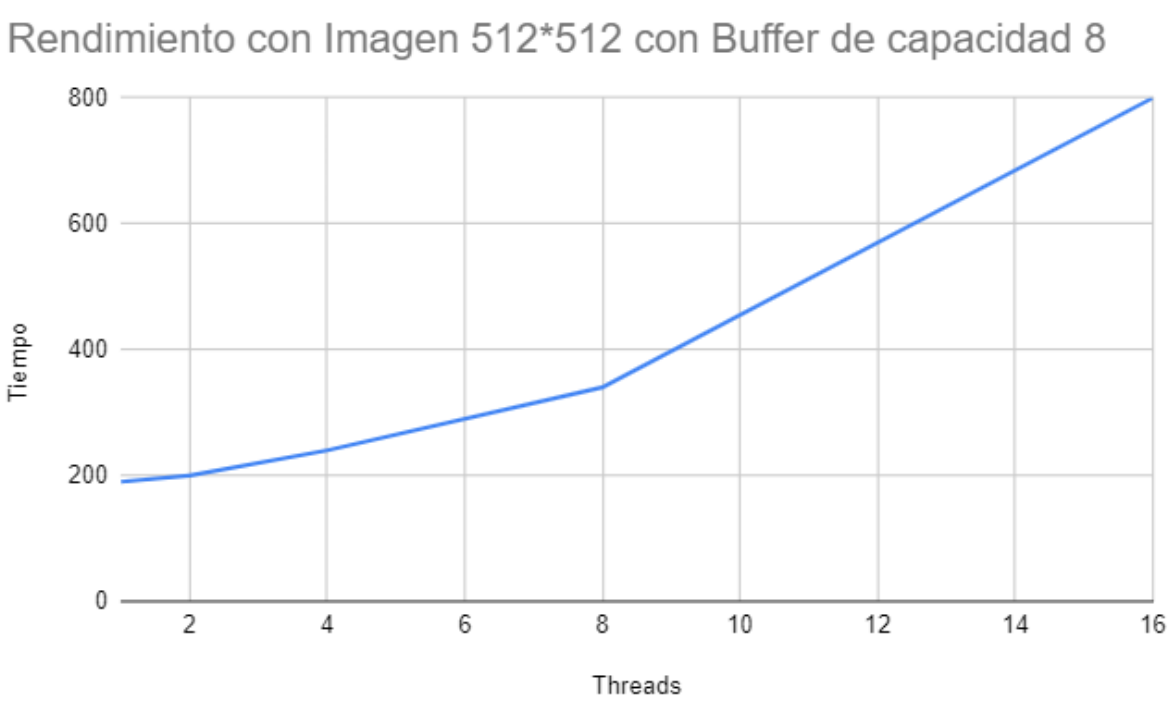
Imagen 1024*1024		
Buffer	Threads	Tiempo
1	1	6200
1	2	6200
1	4	7000
1	8	13000
1	16	12600
2	1	3100
2	2	3100
2	4	3800
2	8	6300
2	16	12800
8	1	790
8	2	800
8	4	900
8	8	1370
8	16	3300
32	1	780
32	2	820
32	4	390
32	8	570
32	16	970
64	1	330
64	2	290
64	8	400
64	16	770

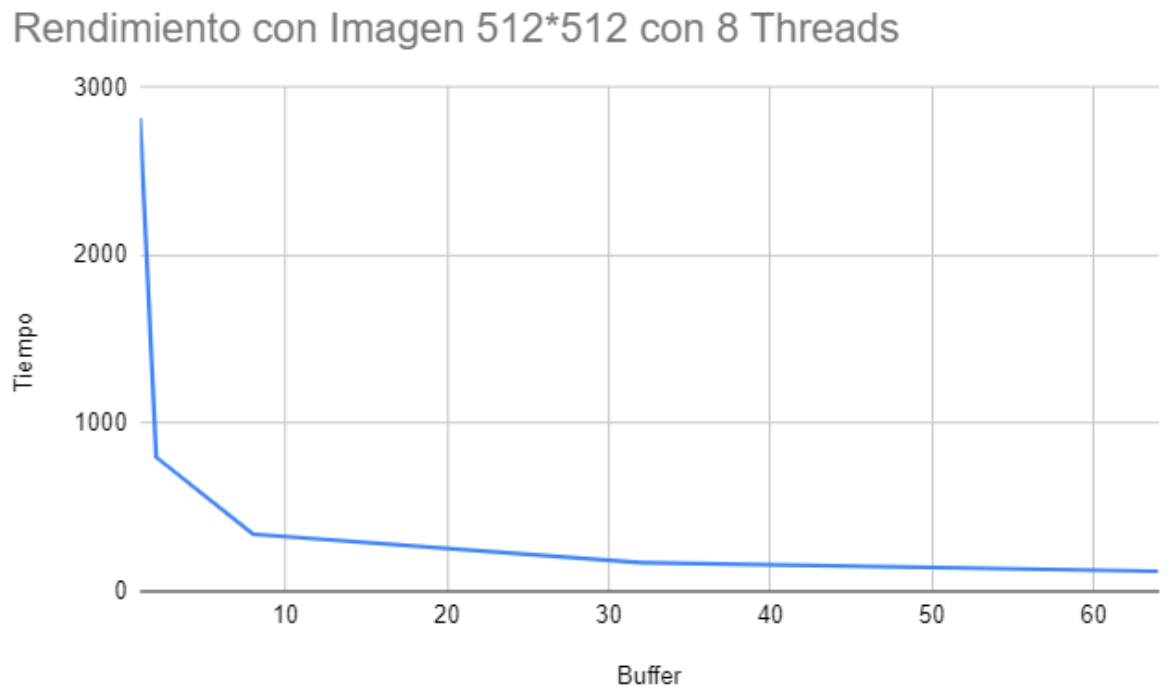
Imagen 512*512px		
Buffer	Threads	Tiempo
1	1	1560
1	2	1570
1	4	1820
1	8	2820
1	16	6500
2	1	780
2	2	800
2	4	900
2	8	1220
2	16	3500
8	1	190
8	2	200
8	4	240
8	8	340
8	16	800
32	1	100
32	2	100
32	4	110
32	8	170
32	16	240
64	1	100
64	2	90
64	8	120
64	16	240

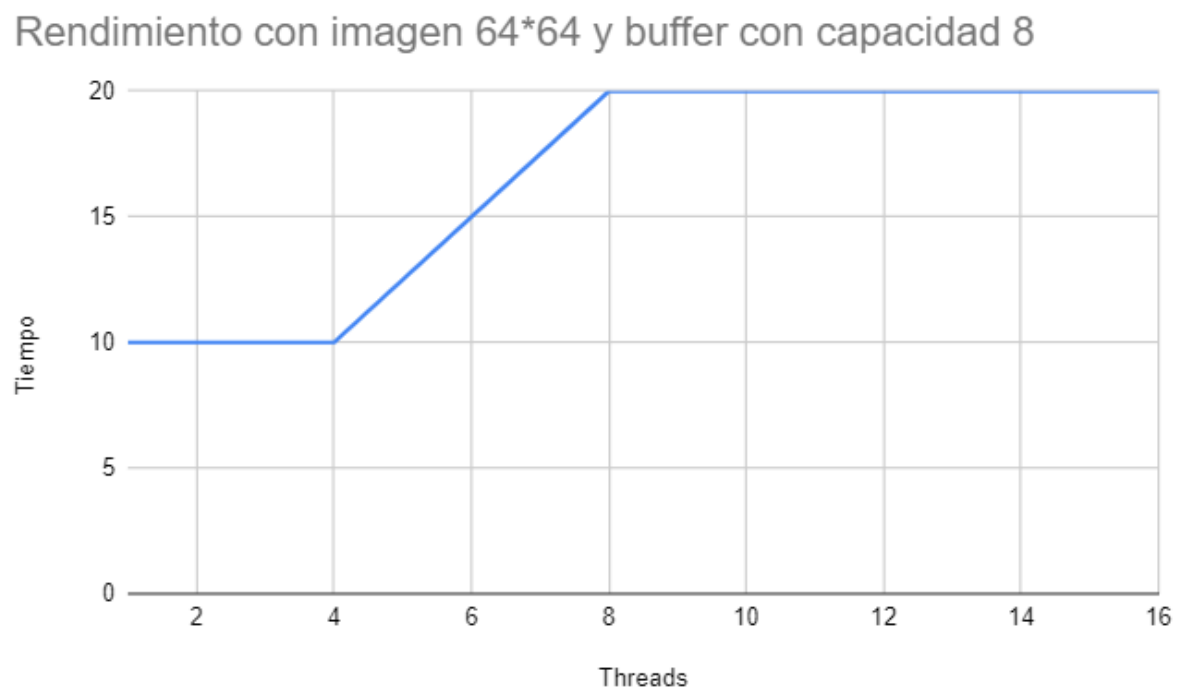
Imagen 64*64px		
Buffer	Threads	Tiempo
1	1	30
1	2	30
1	4	30
1	8	40
1	16	70
2	1	20
2	2	20
2	4	20
2	8	30
2	16	30
8	1	10
8	2	10
8	4	10
8	8	20
8	16	20
32	1	10
32	2	10
32	4	10
32	8	10
32	16	15
64	1	10
64	2	10
64	8	10
64	16	20

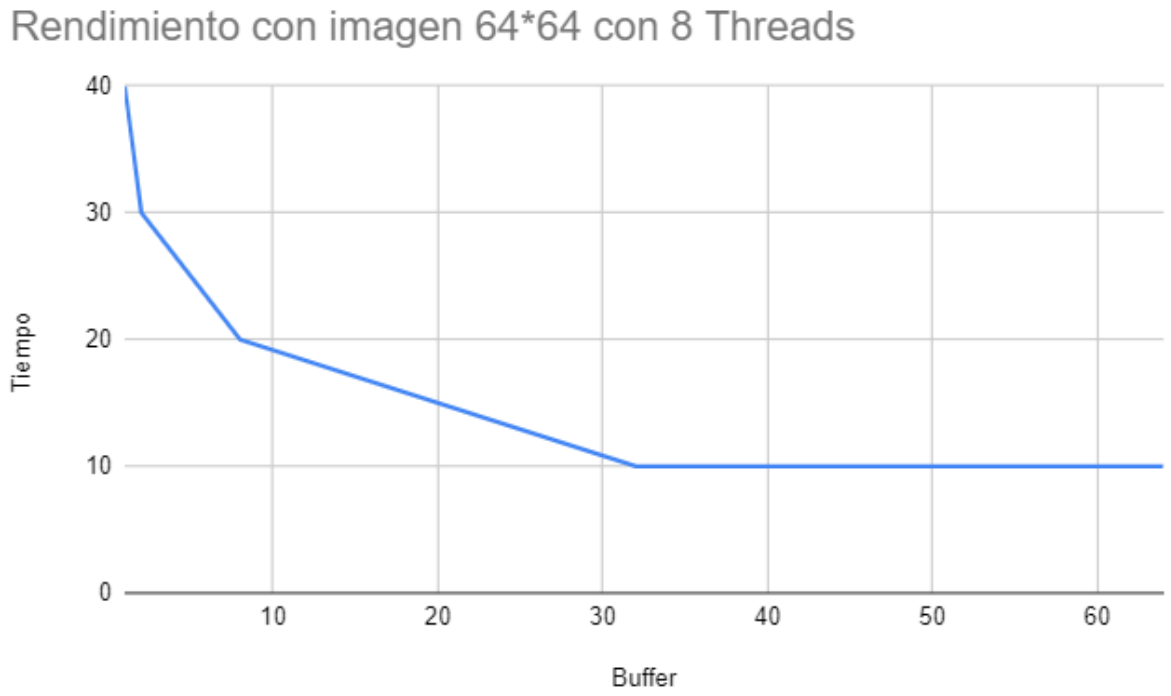

Rendimiento en imagen 1024x1024 con Buffer de capacidad 8
4000
3000
2000
1000
0
Tiempo
2
4
6
8
10
12
14
16
Threads


Rendimiento en imagen 1024*1024 con 8 Threads
15000
10000
5000
0
Tiempo
10
20
30
40
50
60
Buffer


Rendimiento con Imagen 512*512 con Buffer de capacidad 8
800
600
400
200
0
Tiempo
2
4
6
8
10
12
14
16
Threads


Rendimiento con Imagen 512*512 con 8 Threads
3000
2000
1000
0
Tiempo
10
20
30
40
50
60
Buffer





Los tiempos fueron tomados con el método System.currentTimeMillis(), el contador inicial se guarda al crearse el UserPool que es el que le manda el método launch al ThreadPool, mientras que el contador final y la comparación se hacen en el PoolStopper luego de hacer el stop del WorkerCounter que es el que finaliza la convolución.

ANÁLISIS:

Los tiempos máximos se dieron en aquellos casos en que el buffer estaba limitado a un elemento. Por otro lado, se hizo notoria la baja de performance cuando se elevaba la cantidad de threads.
Los tiempos más bajos ocurrieron cuando la capacidad del buffer era la máxima disponible y la cantidad de threads era de 2.