Efecto borroso de una imagen con el uso de pthreads en C++ Computación paralela

Jhonatan Javier Guzmán[1], Jonathan Martinez Chingaté[2]

***Abstract*— El efecto borroso o blur en inglés, es el procedimiento algorítmico que se aplica sobre una imágen en cada uno de sus pixeles para crear la sensación de distorsión sobre ella. En la actualidad existen diferentes métodos para lograr dicho efecto y también ajustes en los parámetros de dichos métodos para ajustar la severidad del efecto, en este caso se le referencia como tamaño del kernel al valor que ajusta la distorsión. El propósito de la práctica es implementar un propio algoritmo que realizará esta operación en dos programas diferentes, el primero se ejecuta de forma secuencial y el segundo de forma paralela mediante el uso de hilos POSIX (pthreads) con el fin de comparar los resultados en tiempo de ejecución. Las pruebas se ejecutaron sobre un procesador Intel Core i3 M330 de 2.13GHz de 2 núcleos físicos y 2 virtuales y se encontró que el tiempo de ejecución secuencial para imágenes de 720p, 1080p y 4K fue de 0.14255, 0.32418 y 1.29642 respectivamente, mientras que en la versión paralela los tiempos con 4 hilos fueron de 0.07431, 0.15891, 0.62758. Finalmente el Speedup encontrado para cada una de las imágenes es de 1.92, 2.04 y 2.07**

I. DESCRIPCIÓN DEL PROGRAMA

La implementación del programa fué realizada en C++. Mediante el uso de la librería opencv se realiza la apertura de la imagen y se pone en memoria mediante el uso de una estructura de datos tipo matriz nativo de la misma librería para permitir el desplazamiento rápido por la imagen mediante el uso de ciclos for. Cada una de las posiciones de la matriz a su vez consta de un arreglo de 3 posiciones y cada posición es de 8 bits, los cuales representan el código RGB de la imagen.

La implementación del efecto consta de la suma de cada uno de los valores RGB de los pixeles vecinos, estos se promedian y se asignan al respectivo pixel, por lo cual es necesario tener la imagen original cargada en memoria, mientras que a su vez se va asignando dicho promedio en la misma posición pero en diferente espacio de memoria, la distancia del vecino depende del tamaño del kernel.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & Pixel & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Fig. 1. Pixeles escogidos para efecto borroso con tamaño del kernel 2

II. PARALELISMO

La implementación paralela consta del mismo algoritmo, pero en esta ocasión cada hilo lanzado se encargará de ejecutar cierta parte de los ciclos, lo que hace que cada hilo se encargue de distorsionar cierta parte de la imagen, finalmente, se espera mediante un join a que todos los hilos terminen su ejecución para que la imagen pueda ser escrita en su totalidad con el efecto borroso y no por partes.

La distribución de la ejecución del ciclo para cada hilo es dado por el siguiente algoritmo.

```
BEGIN
    pass = floor(rows/NUM_THREADS)

    if id thread == 0
        init = 0
        final = pass
    else if id thread == NUM_THREAD - 1
        init = id thread * pass + 1
        final = rows
    else
        init = id thread * pass + 1
        final = init + pass - 1
END
```

Donde *pass* es el tamaño del paso, es decir el número de filas que cada hilo procesará y se obtiene mediante la función piso entre el número de filas que tiene la imagen en pixeles sobre el número de hilos que serán lanzados. Posteriormente, si el hilo es el primero, empezará desde el primer índice hasta el paso; El último hilo irá desde el paso multiplicado por su id mas 1 hasta el número de filas; El

*El presente trabajo es realizado para la clase de computación paralela 2017-2

[1]J. Guzmán es estudiante de Ingeniería de Sistemas y Computación , Universidad Nacional de Colombia jhjguzmanri@unal.edu.co

[2]J. Martinez es estudiante de Ingeniería de Sistemas y Computación , Universidad Nacional de Colombia jomartinezch@unal.edu.co

resto de hilos empezarán con la misma fórmula del último hilo, hasta su inicial mas el paso menos 1. Esto garantiza un empalme perfecto entre los hilos y así todos los pixeles son cubiertos y distorsionados.

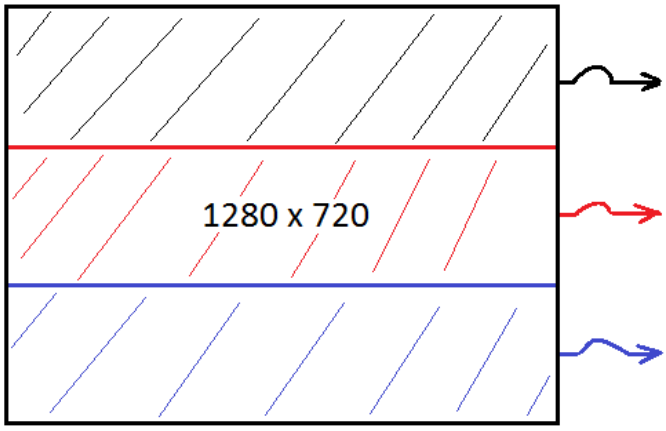

Fig. 2. Empalme con 3 hilos.

Finalmente el tiempo de ejecución es tomado para la versión secuencial y la versión paralelizada.

III. RESULTADOS Y ANÁLISIS

Haciendo uso de un procesador Intel Core i3 M330 de 2.13GHz de 2 núcleos físicos y 2 virtuales se ejecutó la versión secuencial con imágenes de 720p, 1080p y 4K. Arrojando los siguientes resultados.

Imagen	Tiempo (s)
720p	0,14255
1080p	0,32418
4K	1,29642

TABLE I

Tiempos de ejecución secuencial.

En la versión paralela se realizó la ejecución del mismo modo para las 3 imágenes y variando el número de hilos respectivamente. Arrojando los siguientes resultados.

	Tiempo (s)		
Hilos	**720p**	**1080p**	**4K**
2	0,07685	0,16106	0,63787
4	0,07431	0,15891	0,62758
8	0,07286	0,15813	0,63891
16	0,07058	0,15759	0,63436

TABLE II

Tiempos de ejecución paralelo.

En la ejecución de ambas versiones los resultados producidos en las imágenes son exactamente los mismos.

Fig. 3. Imagen 720p original.

Fig. 4. Imagen con efecto borroso y tamaño del kernel 5.

Luego, los tiempos de ejecución pueden ser resumidos en las siguientes gráficas.

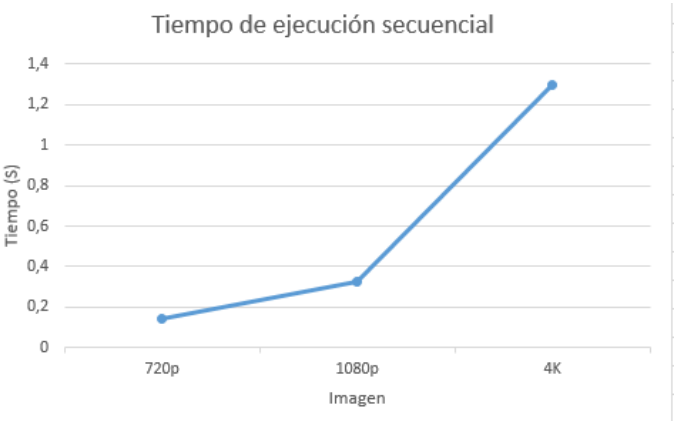

Fig. 5. Gráfica de imagen vs tiempo en modo secuencial.

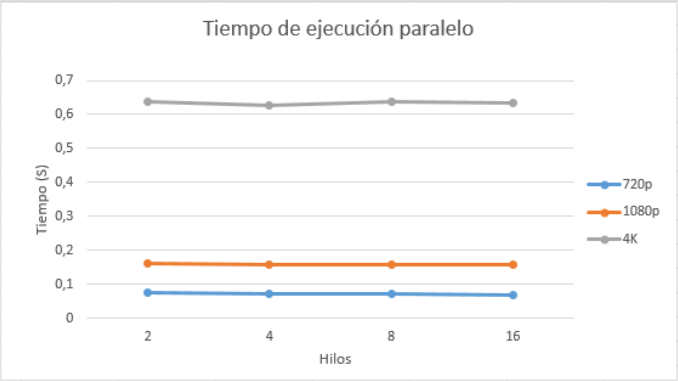

Fig. 6. Tiempo de ejecución según hilos e imágenes

Finalmente, se puede ver el SpeedUp producido para cada imagen y según el número de hilos en la siguiente tabla.

	SpeedUp		
Hilos	**720p**	**1080p**	**4K**
2	1,85	2,01	2,03
4	1,92	2,04	2,07
8	1,96	2,05	2,03
16	2,02	2,06	2,04

TABLE III

SpeedUp según el número de hilos para cada imagen.

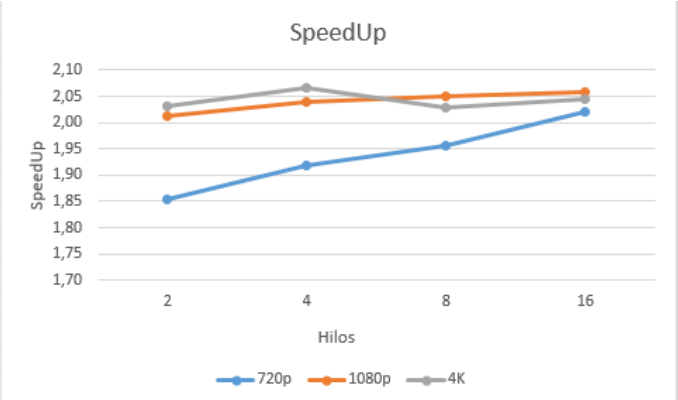

Fig. 7. SpeedUp frente a cada una de las imágenes y el número de hilos

Se puede apreciar como el SpeedUp aumenta de manera considerable con la imagen de 720p y con la imagen de 1080p aumenta solo ligeramente con el aumento del número de hilos. Sin embargo, para la imagen de 4K el SpeedUp mas beneficioso es cuando se lanzan solo 4 hilos.

IV. OPENMP

Para la paralelización del programa haciendo uso de la librería OpenMP solo hace falta hacer uso de la siguiente directiva encima del ciclo.

```
#pragma omp parallel for
```

Inmediatamente la directiva ejecutará dicha porción de código de forma paralela haciendo uso del número de hilos por defecto del procesador. Sin embargo estos pueden ser especificados mediante el uso de la siguiente función.

```
omp_set_num_threads(NUM_THREADS);
```

Haciendo uso de OpenMP se obtuvieron los siguientes resultados.

	Tiempo (s)		
Hilos	**720p**	**1080p**	**4K**
2	0,08190	0,19506	0,78331
4	0,07641	0,18273	0,73716
8	0,07540	0,17486	0,69348
16	0,07389	0,17282	0,68833

TABLE IV

Tiempos de ejecución con OpenMP.

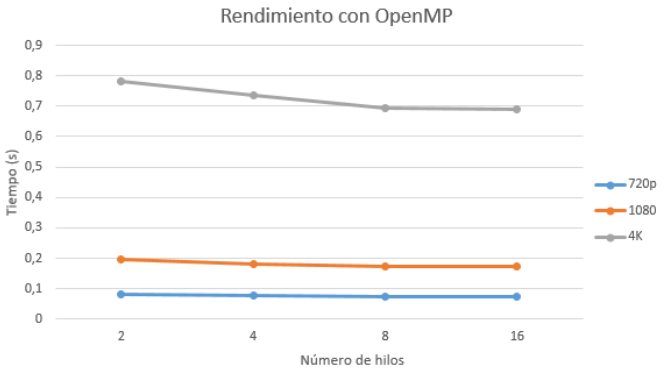

Fig. 8. Tiempo de ejecución según hilos e imágenes con OpenMP.

Se puede apreciar que los tiempos de ejecución son ligeramente mayores con OpenMP que con Posix. Sin embargo OpenMP nos evita particionar el problema ya que el mismo se encarga de partir correctamente el problema.

V. CONCLUSIONES

Se puede observar que el paralelismo usando hilos POSIX reduce drásticamente el tiempo de ejecución del algoritmo al ejecutarlo en varios hilos frente a la versión secuencial. Sin embargo, con el aumento de hilos, la diferencia de tiempo es cada vez mas reducida, debido a la capacidad que tiene el procesador en el cual se ejecutaron las pruebas. Además, el tiempo tomado se tiene en cuenta después de hacer join y esperar a que todos los hilos terminen de ejecutar su porción de código para luego proceder a escribir la imagen, pues entre mayor sea el número de hilos, dicho join también aumenta significativamente el tiempo de ejecución.

Una mayor capacidad de procesamiento debería mejorar el SpeedUp del algoritmo al ejecutarlo especialmente para hilos de 8 y 16 y aún se obtendría el mismo efecto de distorsión deseado ya que este depende del tamaño del kernel.

VI. REFERENCIAS

- Parallel Programming: for Multicore and Cluster Systems - Thomas Rauber, Gudula Rnger