

Práctica 2 - Efecto borroso de una Imagen con OpenMP

Tania Paola Hurtado Pinto, Nicolás Leonardo Maldonado Garzón

Universidad Nacional Bogotá, Colombia

Email: tphurtadop@unal.edu.co, nlmaldonadog@unal.edu.co

Resumen—El siguiente documento busca presentar la implementación hecha del algoritmo gaussiano para desenfocar imágenes implementado en el lenguaje de programación C paralelizado con `posix` y `OpenMP`, lo que permite realizar una comparación de los tiempos de procesamiento alcanzados en cada caso. Se implementó el algoritmo usado en la anterior practica con uso de hilos para mostrar las diferencias que tienen las tecnicas de paralelización usadas en el procesamiento de las imágenes.

gran tamaño como 4k. Se analizaron los tiempos alcanzados para las distintas imágenes procesadas y el número de hilos usados; se compararon estos datos, con los obtenidos en la primera práctica, para apreciar las diferencias entre las dos técnicas de paralelización y observar cual puede ser una mejor opción a la hora de realizar este procedimiento.//

I. INTRODUCCIÓN.

El procesamiento de imágenes es un método usado desde hace alrededor de 60 años para mejorar el aspecto de las imágenes, haciendo evidente ciertos detalles que se desea hacer notar. Existen tanto métodos ópticos como digitales para realizar este proceso, donde actualmente los mas usados son los métodos digitales gracias a los recursos computacionales con los que se cuenta[3]. En muchas ocasiones se requiere procesar una imagen borrosa o desenfocada para poder observarla mucho mas clara y notable, como es el caso de la aplicación en imágenes obtenidas con fines de diagnostico médico o análisis de imágenes aéreas del terreno para determinar recursos naturales, fallas geológicas, entre otras. Este procesos se realiza a partir de la apliación de filtros que eliminan ruido, suavizan la imagen, realzan o detectan bordes realizando operaciones sobre los pixeles para conseguir el efecto especial que se desee.//

La mayoría de estos métodos requieren de muchos recursos computacionales y en muchas ocasiones toma bastante tiempo el procesamiento de las imágenes, ante esto la paralelización es una muy buena alternativa al permitir dividir probemas grandes en mas pequeños que luego son resueltos simultaneamente. A pesar de que dicha alternativa no siempre es buena, puesto que en algunas ocasiones toma mas tiempo la paralelización que su mismo uso, en este caso es facilmente aplicable a la operación entre pixeles.//

En esta práctica se usó un filtro gaussiano con el cual se obtiene un efecto de suavizado y borrosidad de la imagen. Este efecto, mezcla ligeramente los colores de los pixeles vecinos, lo cual hace que la imagen se vea más suave, aunque menos nítida. Para la implementación de este efecto, se usó el lenguaje de programación C, una librería para el procesamiento de imágenes, `OpenCv`, la cual permite manejar imágenes en todos los formatos y tamaños, y una librería, `OpenMP`, para realizar la paralelización y agilizar dicho proceso dado que el procesamiento requiere altos recursos computacionales y tarda demasiado tiempo especialmente al trabajar con imágenes de

II. MARCO TEÓRICO.

II-A. Procesamiento Digital.

El procesamiento digital de imágenes es un conjunto de técnicas que se aplican a las imágenes con el fin de mejorar su calidad o modificar algo en ellas; éste ha tenido grandes avances y sumado a los que han tenido algunas tecnologías como la paralelización, estos cambios se pueden realizar de una manera rapida y eficaz.

II-B. Paralelización.

Podemos definir la paralelización o paralelismo como la forma de que un procesamiento, en nuestro caso el procesamiento de la imagen, pueda dividirse en subprocesos, realizar varios de estos al tiempo y así agilizar el proceso.

II-C. Kernel.

Se entiende como una matriz de coeficientes donde el entorno del punto (x,y) que se considera en la imagen para obtener $g(x,y)$ está determinado por el tamaño y forma del kernel seleccionado.

II-D. OPENMP.

Es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en multiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran. Está disponible en muchas arquitecturas, incluidas plataformas de Unix y Windows. Se compone de un conjunto de directivas de compilador, rutinas de biblioteca, y variables de entorno que influyen en el comportamiento en tiempo de ejecución.

III. LIBRERÍAS

Para el manejo de imágenes se empleó la librería "OpenCV" en C/C++ la cual permitió cargar una imagen representada como una matriz de enteros (grupo de 3 enteros) correspondientes a los canales RGB y modificarla de acuerdo a los efectos que se plantearon en el taller.

Para esta segunda práctica se utilizó la librería "OpenMP" que nos sirve para la programación multiproceso de memoria compartida por medio de ejecución paralela de varios subprocesos.

IV. MULTIPLICACIÓN DE MATRICES.

Consiste en el proceso de producir una matriz con dimensiones $M \times O$, realizando el producto entre otras dos matrices, siempre y cuando cumplan la característica de que la primera tenga dimensiones $M \times N$ y la segunda $N \times O$. Este proceso se realiza, multiplicando cada elemento de una fila i de la primera matriz respectivamente con un elemento de una columna j de la segunda matriz, luego sumar los resultados de esos productos y así obtener un valor para la casilla i,j de la nueva matriz.

En general una matriz de tamaño $M \times N$ se puede multiplicar con cualquier matriz que tenga un tamaño $N \times O$ lo cual da como resultado una matriz de tamaño $M \times O$.

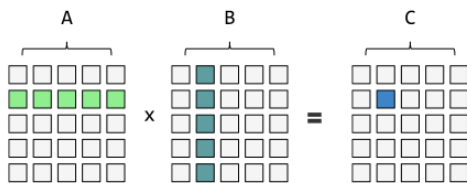


Figura 1: Multiplicación entre matrices.

V. PARALELIZACIÓN DEL ALGORITMO.

Para la implementación del algoritmo se usaron dos funciones principales: *matriz_filtro* en la cual se aplica la función gaussiana a la matriz kernel, la cual será del tamaño que el usuario desee. Para esto se usó la ecuación:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

En las siguientes imágenes se puede observar la aplicación de la función gaussiana a una matriz.

(-1,1)	(0,1)	(1,1)
(-1,0)	(0,0)	(1,0)
(-1,-1)	(0,-1)	(1,-1)

Figura 2: Matriz de posiciones para cambiar el efecto gaussiano.

0.0947416	0.118318	0.0947416
0.118318	0.147761	0.118318
0.0947416	0.118318	0.0947416

Figura 3: Matriz gaussiana con kernel = 3.

Posteriormente, para aplicar el efecto gaussiano, en la función *blur* se realiza el proceso de multiplicar la matriz kernel obtenida de la función anterior, con los píxeles de la imagen. Esto se hace tomando cada píxel y los píxeles que lo rodean en un radio de $kernel/2$. Para cada uno de estos píxeles se obtiene el valor RGB, mediante la librería utilizada, y cada valor se multiplica con el valor de la celda de la matriz gaussiana correspondiente. Después de esto se suman los valores resultantes de la multiplicación y este será el valor correspondiente al RGB del píxel tomado. Este proceso se realiza para cada uno de los píxeles de la imagen. En las siguientes imágenes se observa la multiplicación realizada para cada píxel asumiendo los valores para un color.

14	15	16
24	25	26
34	35	36

Figura 4: Valores de color verde (0-25) en una región de la imagen.

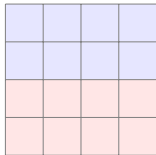
14x0.0947416	15x0.118318	16x0.0947416
24x0.118318	25x0.147761	26x0.118318
34x0.0947416	35x0.118318	36x0.0947416

Figura 5: Multiplicación del valor del pixel por la matriz gaussiana.

1.32638	1.77477	1.51587
2.83963	3.69403	3.07627
3.22121	4.14113	3.4107

Figura 6: Resultado de la aplicación del efecto.

Para realizar la multiplicación de manera paralela, se planteó dividir la matriz en el número de procesos o hilos a utilizar, y de esta manera cada unidad de procesamiento se encargaría de operar únicamente las filas que le correspondieran.



En la figura anterior, podemos ver un ejemplo de la subdivisión de cargas para dos unidades de procesamiento. En la primera, se dedicará a multiplicar las filas de color azul por la otra unidad de procesamiento rojas, esta división se realizó por medio de la implementación de la librería OpenMP, la cual por debajo hace todo el procesamiento de generar los hilos que se ejecutan de forma paralela y finalmente coloca llena la matriz resultado.

VI. EXPERIMENTOS Y RESULTADOS

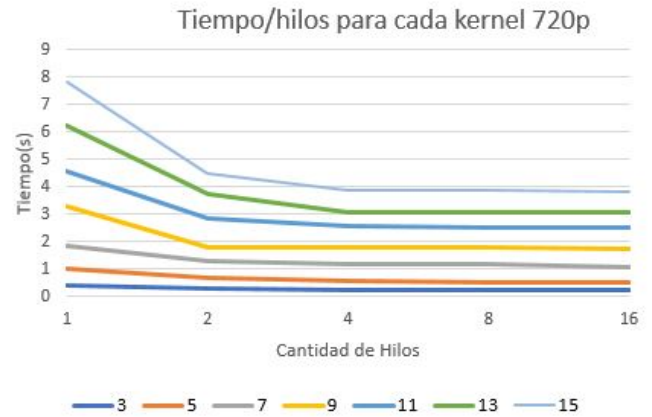
VI-A. Usando POSIX (Practica 1)

Se tomaron imágenes de 720p, 1080p y 4k para realizar el procedimiento del efecto borroso, éste se realizó con kernels de 3, 5, 7, 9, 11, 13 y 15 para cada imagen y a su vez se paralelizó con 1, 2, 4, 8 y 16 hilos POSIX. El procedimiento se realizó un total de 10 veces por combinación, donde se tomó el tiempo de cada ejecución del programa, se obtenía el promedio por combinación, se llenaron las siguientes tablas con esos datos que posteriormente se graficaron.

Cuadro I: Tiempo en segundos para aplicar el efecto en imagen de 720px

Hilos	Kernel 3	Kernel 5	Kernel 7	Kernel 9	Kernel 11	Kernel 13	Kernel 15
1	0,389	1,013	1,823	3,273	4,569	6,244	7,807
2	0,269	0,663	1,299	1,818	2,877	3,726	4,492
4	0,253	0,556	1,189	1,82	2,569	3,092	3,857
8	0,252	0,526	1,191	1,812	2,541	3,071	3,86
16	0,251	0,522	1,061	1,731	2,527	3,092	3,841

Figura 7: Gráfica de tiempos para cada kernel en función de la cantidad de hilos.



Cuadro II: Tiempo en segundos para aplicar el efecto en imagen de 1080px

Hilos	Kernel 3	Kernel 5	Kernel 7	Kernel 9	Kernel 11	Kernel 13	Kernel 15
1	0,802	1,908	3,724	5,96	8,814	12,255	16,206
2	0,604	1,165	2,091	3,295	4,83	6,672	8,741
4	0,481	1,06	2,02	3,217	4,713	6,527	8,582
8	0,46	1,054	1,994	3,197	4,628	6,495	8,549
16	0,45	1,035	1,956	3,128	4,583	6,34	8,432

Figura 8: Gráfica de tiempos para cada kernel en función de la cantidad de hilos.



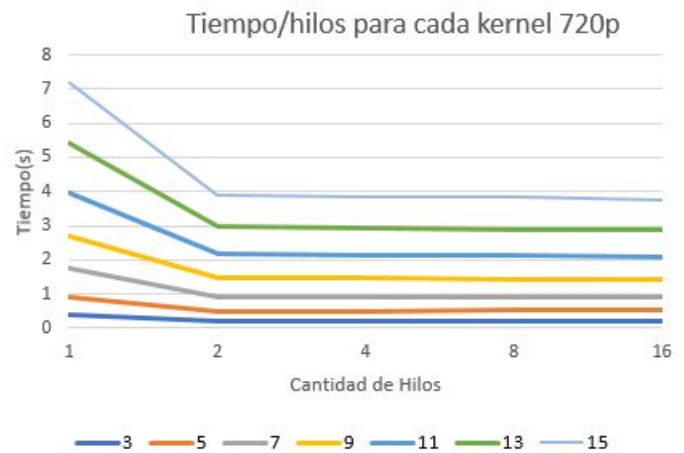
Cuadro III: Tiempo en segundos para aplicar el efecto en imagen de 4k

Hilos	Kernel 3	Kernel 5	Kernel 7	Kernel 9	Kernel 11	Kernel 13	Kernel 15
1	3,145	7,499	14,676	23,644	34,939	47,977	64,175
2	1,781	4,216	8,139	12,974	19,021	26,159	34,593
4	1,757	4,146	7,884	12,704	18,696	25,871	34,456
8	1,752	4,122	7,846	12,723	18,562	25,686	34,315
16	1,732	4,083	7,776	12,516	18,475	25,525	33,975

Figura 9: Gráfica de tiempos para cada kernel en función de la cantidad de hilos.



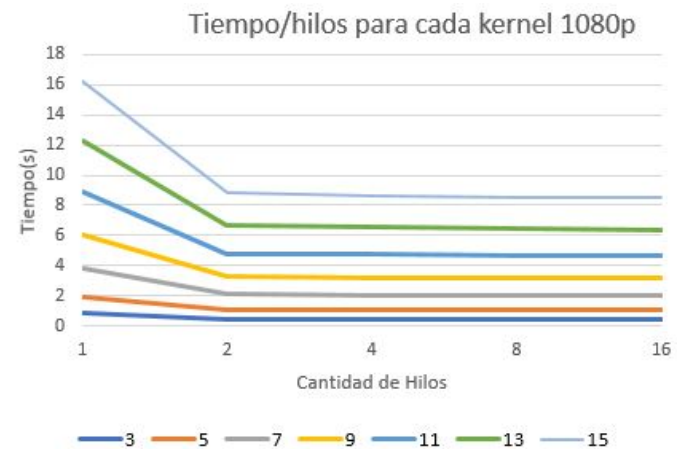
Figura 10: Gráfica de tiempos para cada kernel en función de la cantidad de hilos.



Cuadro V: Tiempo en segundos para aplicar el efecto en imagen de 1080px

Hilos	Kernel 3	Kernel 5	Kernel 7	Kernel 9	Kernel 11	Kernel 13	Kernel 15
1	0,817	1,959	3,766	6,055	8,942	12,271	16,249
2	0,458	1,08	2,09	3,299	4,805	6,632	8,836
4	0,477	1,064	2,034	3,225	4,731	6,528	8,667
8	0,461	1,07	2,012	3,214	4,705	6,493	8,571
16	0,452	1,036	1,969	3,18	4,621	6,391	8,517

Figura 11: Gráfica de tiempos para cada kernel en función de la cantidad de hilos.



VI-B. Usando OPENMP (Practica 2)

Para la implementación de OpenMP en la práctica, se hizo uso de las mismas funciones anteriormente descritas, reemplazando las líneas con los ciclos de creación y unión de los hilos por una sección pragma omp con el ciclo for de ejecución del programa. Luego igualmente se tomaron imágenes de 720p, 1080p y 4k para realizar el procedimiento del efecto borroso, con los mismos tamaños de kernels y cantidades de hilos. El procedimiento se realizó un total de 10 veces por combinación, donde se tomó el tiempo de cada ejecución del programa, se obtenía el promedio por combinación, se llenaron las siguientes tablas con esos datos que posteriormente se gráfcaron.

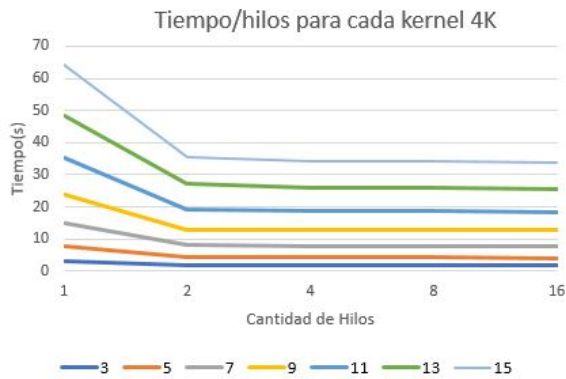
Cuadro IV: Tiempo en segundos para aplicar el efecto en imagen de 720px

Hilos	Kernel 3	Kernel 5	Kernel 7	Kernel 9	Kernel 11	Kernel 13	Kernel 15
1	0,369	0,896	1,752	2,688	3,977	5,429	7,2
2	0,214	0,49	0,91	1,463	2,172	2,967	3,892
4	0,225	0,506	0,901	1,453	2,123	2,93	3,85
8	0,217	0,516	0,9	1,443	2,124	2,899	3,834
16	0,218	0,508	0,885	1,448	2,081	2,879	3,776

Cuadro VI: Tiempo en segundos para aplicar el efecto en imagen de 4k

Hilos	Kernel 3	Kernel 5	Kernel 7	Kernel 9	Kernel 11	Kernel 13	Kernel 15
1	3,241	7,667	14,871	23,756	35,13	48,58	64,311
2	1,756	4,226	8,153	12,987	19,011	27,397	35,272
4	1,769	4,157	7,911	12,727	18,718	25,816	34,136
8	1,772	4,12	7,867	12,67	18,693	25,743	34,023
16	1,749	4,093	7,798	12,648	18,485	25,587	33,912

Figura 12: Gráfica de tiempos para cada kernel en función de la cantidad de hilos.



VII. SPEED UP.

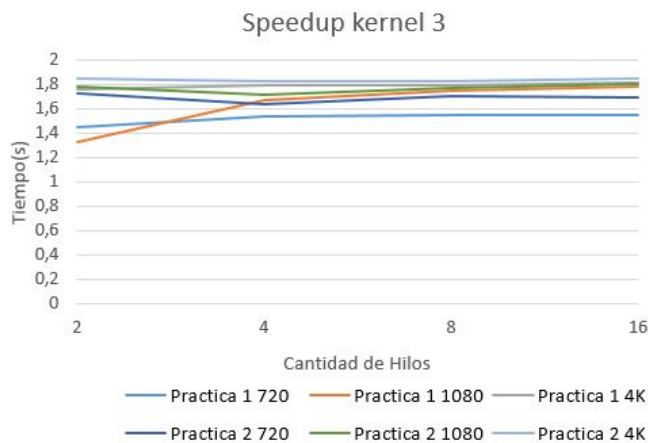
El Speedup es una medida de rendimiento mediante la cual se busca aumentar la eficiencia de ejecución de un proceso ejecutado de dos formas diferentes, en este caso, éstas serán de forma secuencial y paralelizada con diferentes cantidades de hilos en las dos diferentes versiones de paralelización, este valor es calculado con la siguiente función tomando los datos obtenidos en las dos prácticas y consignados en las anteriores tablas.

$$S_p(n) = \frac{T^*(n)}{T_p(n)}$$

Cuadro VII: Resultado de realizar el Speedup con un kernel de tamaño 3

Hilos	720 Pr1	1080 Pr1	4k Pr1	720 Pr2	1080 Pr2	4k Pr2
2	1,45	1,33	1,77	1,72	1,78	1,85
4	1,54	1,67	1,79	1,64	1,71	1,83
8	1,54	1,74	1,80	1,70	1,77	1,83
16	1,55	1,78	1,82	1,69	1,81	1,85

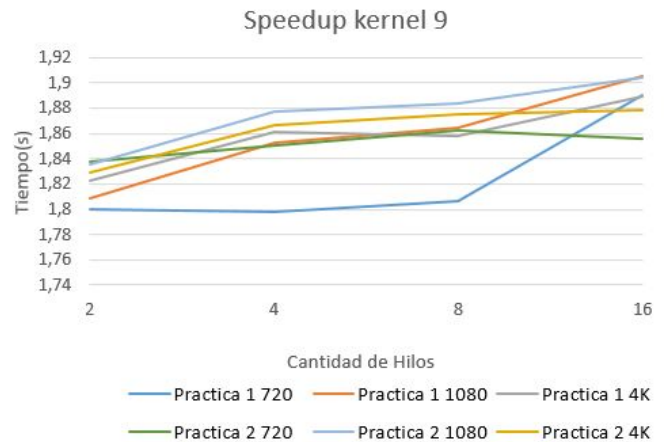
Figura 13: Gráfica del Speedup para cada imagen procesada con un kernel 3.



Cuadro VIII: Resultado de realizar el Speedup con un kernel de tamaño 9

Hilos	720 Pr1	1080 Pr1	4k Pr1	720 Pr2	1080 Pr2	4k Pr2
2	1,80	1,81	1,82	1,84	1,84	1,83
4	1,80	1,85	1,86	1,85	1,88	1,87
8	1,81	1,86	1,86	1,86	1,88	1,87
16	1,89	1,91	1,89	1,86	1,90	1,88

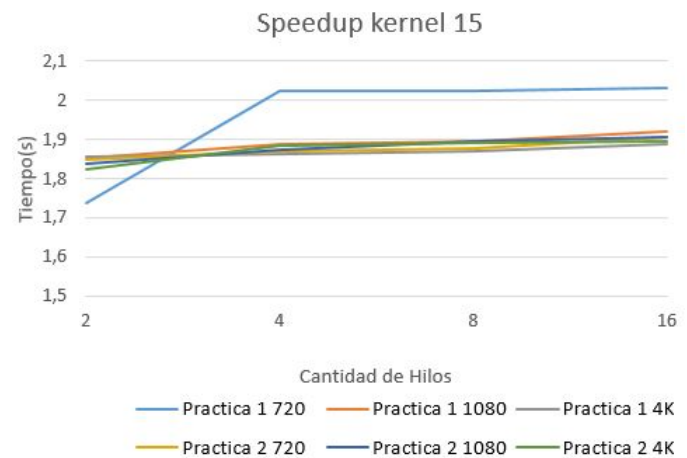
Figura 14: Gráfica del Speedup para cada imagen procesada con un kernel 9.



Cuadro IX: Resultado de realizar el Speedup con un kernel de tamaño 15

Hilos	720 Pr1	1080 Pr1	4k Pr1	720 Pr2	1080 Pr2	4k Pr2
2	1,74	1,85	1,86	1,85	1,84	1,82
4	2,02	1,89	1,86	1,87	1,87	1,88
8	2,02	1,90	1,87	1,88	1,90	1,89
16	2,03	1,92	1,89	1,91	1,91	1,90

Figura 15: Gráfica del Speedup para cada imagen procesada con un kernel 15.



CONCLUSIONES.

1. La capacidad de medir si el tiempo de ejecución del procedimiento mejora o empeora con diferentes cantidades de hilos, depende totalmente de los recursos que se tengan a disposición.
2. El efecto borrosos de la imagen se incrementa con el tamaño del kernel usado.
3. El tiempo de ejecución aumenta a medida que se aumenta el tamaño del kernel, sin embargo para un mismo tamaño de kernel, el tiempo disminuye a medida que se aumenta el número de hilos.
4. La paralelización aplicada a ese problema reduce el speedup considerablemente.
5. En cuanto a las gráficas de speedup se puede observar que dados los limitados recursos computacionales con los que se cuentan, no se observa una gran diferencia conforme se incrementan la cantidad de hilos, debido a que al paralelizar solo encola los nuevos procesos.
6. También en cuanto al speed up, podemos observar que las dos formas del paralelismo tienen un comportamiento bastante similar al operar sobre imágenes y kernels iguales, esto puede ser debido a la limitación en recursos computacionales y a que, así los dos programas paralelizan de formas diferentes, el proceso llevado a cabo (partir la imagen, aplicarle el efecto y luego concatenarla) es el mismo y como se corrieron en la misma máquina esto no debería generar muchos cambios.

REFERENCIAS

- [1] <http://www.pixelstech.net/article/1353768112-Gaussian-Blur-Algorithm>.
- [2] Gaussian Blur. Artículo de Wikipedia. https://en.wikipedia.org/wiki/Gaussian_blur.
- [3] Daniel Malacara, Procesamiento de imágenes. Tomado de internet. http://bibliotecadigital.ilce.edu.mx/sites/ciencia/volumen2/ciencia3/084/htm/sec_9.htm.