REPORTE: PRIMERA PRACTICA DE PROGRAMACIÓN PARALELA Y DISTRIBUIDA

Julian David Rodríguez Ruíz, Alberto Nicolai Romero Martínez, Edder Hernández Forero

Ingeniería de sistemas y computación - Computación paralela y distribuida

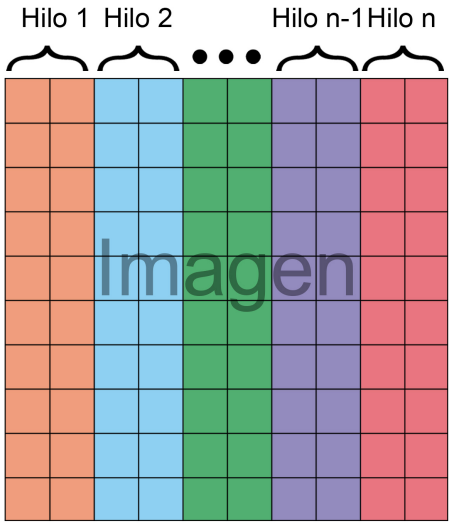
1. MÉTODO DE DESENFOQUE

Para la generación de efecto borroso en la imagen, se implemento un desenfoque que consiste en promediar los componentes R, G y B de los píxeles en las cercanías para hallar el valor de los componentes del píxel central. Este procedimiento se ejecuta para cada píxel. En caso de encontrarse en el borde de la imagen, se realiza un efecto espejo en el que al intentar leer un píxel fuera de los límites de la imagen cierta distancia se lee un píxel existente dentro de los límites de la imagen esa misma distancia. Se puede entender el algoritmo en otros términos como una convolución entre la imagen y un kernel de tamaño k cuyos componentes son todos $1/k^2$.

2. PARALELIZACIÓN

El método de paralelización consiste en dividir la matriz de píxeles según el número de hilos y asignar una parte a cada hilo. Como se puede ver en la **Fig. 1** Por lo cual, el método de división de datos es *block wise*.

Fig. 1. Representación de imagen y su división de datos tipo *block wise* por columna



La imagen es dividida por columnas con el propósito de que la división de trabajo, entre los hilos, sea lo más equitativa posible. Las imágenes de prueba (con resolución 720p, 1080p y 4k) tienen un número de columnas divisible por todos los números de hilos a probar (1, 2, 4, 8 y 16), por lo tanto cada hilo tendrá la misma carga de trabajo al tener un misma cantidad de píxeles con los que trabajar. En el caso en que el número de columnas no sea divisible por el número de hilos, el algoritmo simplemente toma la parte entera de la división $No.Columnas/No.Hilos$ para asignar el trabajo correspondiente, y el ultimo hilo toma la parte restante.

3. EXPERIMENTOS

Las pruebas del algoritmo están orientadas en medir el tiempo que tarda el algoritmo en transformar las imágenes y calcular el *speedup* según el número de hilos. El propósito es ver el cómo paralelizar puede mejorar el rendimiento de tiempo de ejecución.

El número de hilos, el tamaño del kernel y la resolución de las imágenes de prueba son tomadas como variables de independientes, mientras que el tiempo y *speedup* cumplen el rol de variable dependiente. Los recursos (principalmente el tipo de procesador) que disponga el computador donde sea ejecutado el algoritmo será la variable de control.

Las pruebas están categorizadas por las tres variables independientes. Primero se tomada una imagen de las tres imágenes de prueba de diferente resolución, las cuales son 720p, 1080p o 4k; después se fija el número de hilos que tratarán el problema, se emplean 1, 2, 4, 8, o 16 hilos; y finalmente el tamaño del kernel, que toma los valores de 3, 5, 7, 9, 11, 13 o 15.

El algoritmo fue probado en un computador con procesador intel core i7, con 4 núcleos y 8 hilos, a 4GHz como base . Cada prueba es repetida 10 veces para posteriormente calcular el tiempo promedio que toma al proceso terminar. Con los tiempos promedio se calcula el speedup, $S_p = T_s/T_h$. Tiempo que toma el programa en su ejecución en forma secuencial (o un hilo), sobre el tiempo que toma en su forma paralelizada.

4. RESULTADOS POSIX

De los experimentos realizados, se obtuvieron los siguientes resultados.

Universidad Nacional de Colombia

Tiempo de ejecución en imagen 720p en segundos

	K=3	K=5	K=7	K=9	K=11	K=13	K=15
Th=1	0.22	0.31	0.46	0.63	0.86	1.11	1.42
Th=2	0.19	0.22	0.28	0.37	0.48	0.61	0.77
Th=4	0.17	0.19	0.23	0.26	0.32	0.39	0.48
Th=8	0.15	0.17	0.20	0.24	0.30	0.36	0.44
Th=16	0.15	0.17	0.22	0.25	0.31	0.37	0.50

Table 1. Tiempo de la imagen en resolución 720p

Speedup en imagen 720p

	K=3	K=5	K=7	K=9	K=11	K=13	K=15
Th=1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Th=2	1.16	1.41	1.64	1.70	1.79	1.82	1.84
Th=4	1.29	1.63	2.00	2.42	2.69	2.85	2.96
Th=8	1.47	1.82	2.30	2.62	2.87	3.08	3.23
Th=16	1.47	1.82	2.09	2.52	2.77	3.00	2.84

Table 2. Speedup de la imagen en resolución 720p

Fig. 2. Tiempos con imagen de 720p

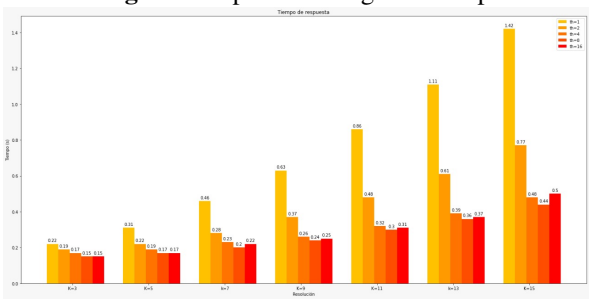
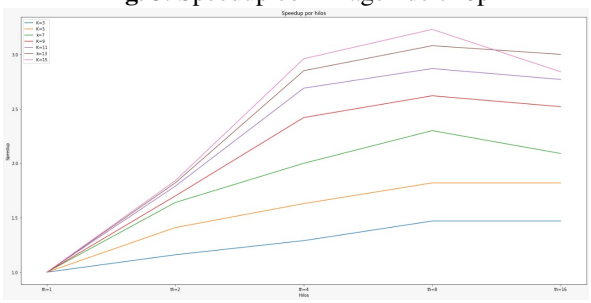


Fig. 3. Speedup con imagen de 720p



Tiempo de ejecución en imagen 1080p en segundos

	K=3	K=5	K=7	K=9	K=11	K=13	K=15
Th=1	0.42	0.65	0.97	1.36	1.90	2.50	3.16
Th=2	0.32	0.43	0.60	0.80	1.04	1.35	1.70
Th=4	0.29	0.35	0.43	0.53	0.67	0.85	1.09
Th=8	0.27	0.32	0.42	0.54	0.65	0.80	1.08
Th=16	0.27	0.32	0.43	0.51	0.68	0.83	1.11

Table 3. Tiempo de la imagen en resolución 1080p

Speedup en imagen 1080p

	K=3	K=5	K=7	K=9	K=11	K=13	K=15
Th=1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Th=2	1.31	1.51	1.62	1.70	1.83	1.85	1.86
Th=4	1.45	1.86	2.26	2.57	2.84	2.94	2.90
Th=8	1.56	2.03	2.31	2.52	2.92	3.12	2.93
Th=16	1.56	2.03	2.26	2.67	2.79	3.01	2.85

Table 4. Speedup de la imagen en resolución 1080p

Fig. 4. Tiempos con imagen de 1080p

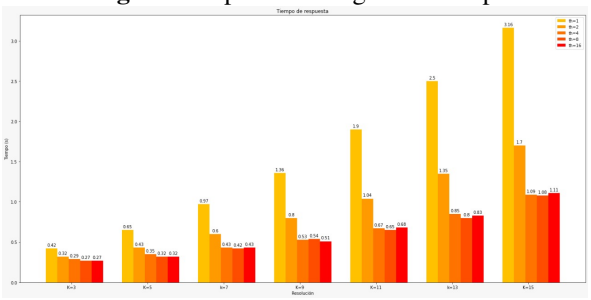
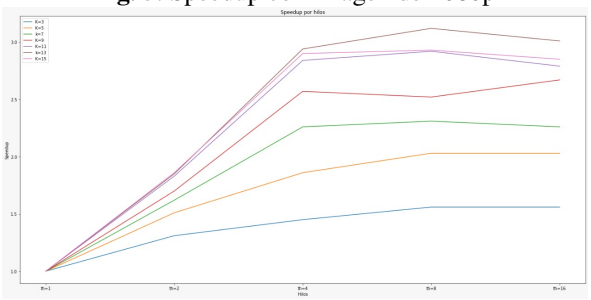


Fig. 5. Speedup con imagen de 1080p



Tiempo de ejecución en imagen 4K en segundos

	K=3	K=5	K=7	K=9	K=11	K=13	K=15
Th=1	1.72	2.62	3.85	5.47	7.46	9.83	12.58
Th=2	1.31	1.75	2.38	3.20	4.21	5.48	6.93
Th=4	1.14	1.36	1.67	2.10	2.86	3.57	4.45
Th=8	1.10	1.34	1.61	2.14	2.79	3.32	4.43
Th=16	1.12	1.33	1.69	2.20	2.81	3.50	4.36

Table 5. Tiempo de la imagen en resolución 4k

Speedup en imagen 4k

	K=3	K=5	K=7	K=9	K=11	K=13	K=15
Th=1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Th=2	1.31	1.50	1.62	1.71	1.77	1.79	1.82
Th=4	1.51	1.93	2.31	2.60	2.61	2.75	2.83
Th=8	1.56	1.96	2.39	2.56	2.67	2.96	2.84
Th=16	1.54	1.97	2.28	2.49	2.64	2.81	2.89

Table 6. Speedup de la imagen en resolución 4k

Fig. 6. Tiempos con imagen de 4k

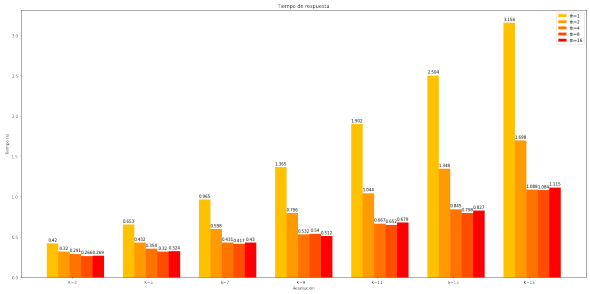
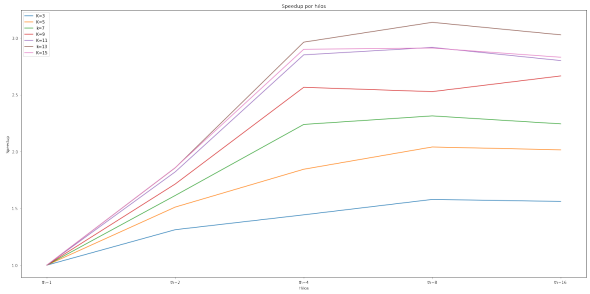


Fig. 7. Speedup con imagen de 4k



5. ANÁLISIS

Lo primero que se debe tener en cuenta es que el computador en donde fue ejecutado el código solo tiene una capacidad de 8 hilos, lo que ocasiona que, en el momento de declarar el uso de 16 hilos, no se mostrara un comportamiento real al uso de 16 hilos. Es por esta razón que no hay realmente mayor diferencia mostrada entre el uso de 8 o 16 hilos, incluso siendo peor el uso de 16 hilos. Lo anterior se puede evidenciar en las Fig 2, 4, 6 .

Es evidente, con las mismas figuras, que en la imagen de resolución 720p no se muestra mayor diferencia el tener 8 o 16 hilos hasta tener un kernel de tamaño 7, en la imagen de 1080 p hasta tener un kernel de tamaño 5 y en la imagen de 4k siempre se muestra una diferencia de tiempo. Siempre hay mayor consumo de tiempo por parte de la barra de 16 hilos con respecto a la de 8 hilos. Es además posible que el uso de 16 hilos tenga un mayor consumo de tiempo a usar 4 hilos, como se evidencia claramente en la Fig 4 con un kernel de 15, y el mismo comportamiento, pero de forma no tan pronunciada, en las Fig 4 y 6.

Por la misma razón anterior, en las Fig 3, 5 y 7, y a su vez en las tablas 4, 4 y 4 que el uso de 16 hilos muy rara vez muestra un mayor speedup que respecto al uso de 8 hilos.

Por otro lado, las Fig 2, 4 y 6, muestra que el código tiene un incremento en tiempo exponencial a con respecto al aumento del tamaño del kernel. El uso de hilos hace que esa curva sea menos pronunciada. Aunque sigue siendo fácilmente distinguible con el uso de tan solo dos hilos. Ese comportamiento da a entender que con kernel de menor tamaño (3 y 5) la diferencia en tiempo secuencial con respecto a el uso de hilos no es muy pronunciada. Con mayores kernel (7, 9, 11, 13 y 15) la diferencia es fácilmente apreciable. Se da el mismo comportamiento con la apelación de 2 hilos en comparación a el uso de 4, 8 y 16 hilos; por contra parte, entre el uso de 4, 8 y 16 hilos no hay en realidad un gran diferencia en tiempo.

En realidad no hay mayor diferencia entre 4 y 8 hilos, ya que, como se había mencionado anteriormente, para el computador es físicamente imposible usar 16 hilos, por tanto, sobre el uso de 16 hilos no se puede concluir nada a aparte de que es mejor no usar esa cantidad en ese computador. Pero, ya que no hay demasiada diferencia entre el uso de 4 hilos, se podría pensar que, con 16 hilos reales, habría solo una mejora pequeña.

En las Fig 3, 5 y 7, se muestra que hay un gran aumento en cuanto a speedup con cualquier valor de kernel, pero más evidente en kernel mayores a 9, hasta el uso de 4 hilos. Con 8 hilos también hay aumento de speedup, pero es claro que empieza a estabilizarse. Por lo tanto se apoya a la teoría de que el uso de 16 hilos no aportaría mayor mejora en rendimiento. Principalmente con kernel de bajo valor.

6. RESULTADOS OPENMP

En una implementación posterior, se utiliza la librería OpenMP para la paralelización de tareas.

Fig. 8. Tiempos con imagen de 720p

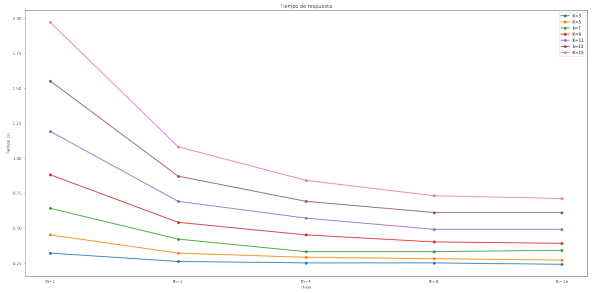


Fig. 9. Speedup con imagen de 720p

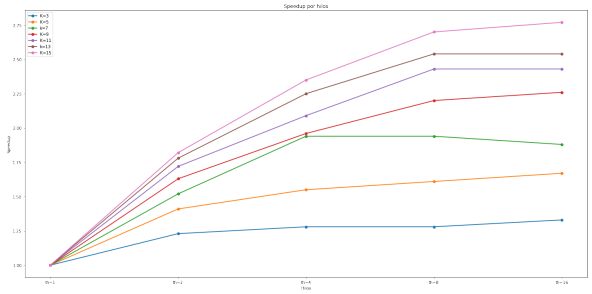


Fig. 10. Tiempos con imagen de 1080p

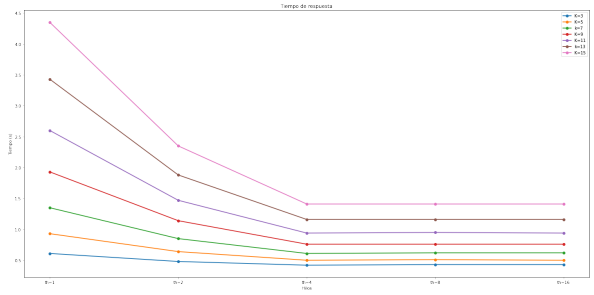


Fig. 11. Speedup con imagen de 1080p

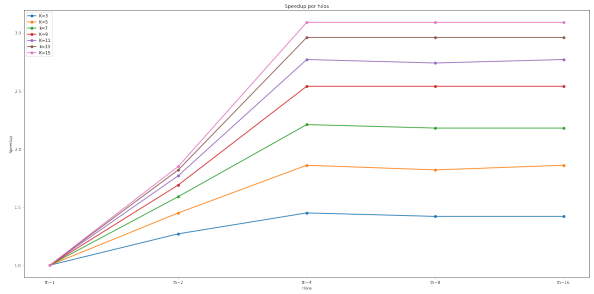


Fig. 12. Tiempos con imagen de 4k

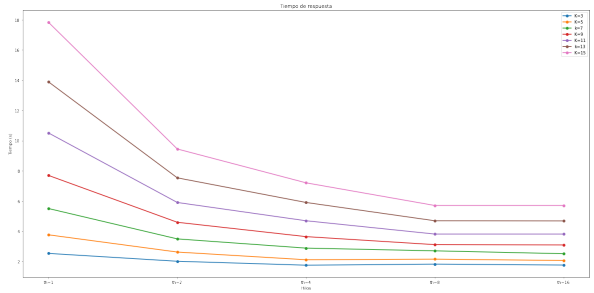
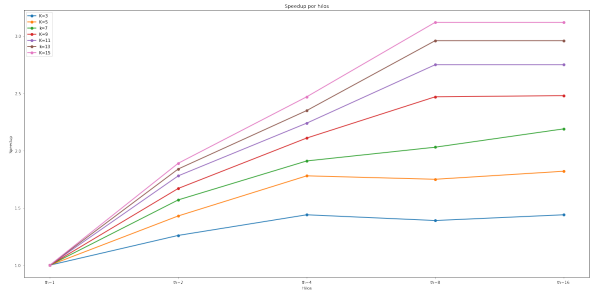


Fig. 13. Speedup con imagen de 4k



7. CONCLUSIONES

Es evidente que el paralelizar el programa de desenfoque proporciona una mejora evidente en contraste a la ejecución en secuencial. Hay una gran mejora hasta el uso de 4 hilos. El uso de 8 hilos tiene en todos los casos tiempos similares a el uso de 4 hilos, pero aun presenta mejoras apreciables.

Sobre el uso de 16 hilos se puede concluir que, para este caso, y por la falta de un dispositivo con la capacidad de correr tener 16 subprocesos, no es recomendable su uso. Principalmente porque en varios casos muestra un peor rendimiento frente al uso de 8 hilos (la máxima capacidad del computador). Es también por la misma razón que no se puede concluir nada acerca del uso de 16 hilos y su speedup en el programa, pero se puede teorizar que no tendría una mayor mejora, porque, con el uso de 8 hilos, las gráficas mostraban que empezaba a llegar a su speedup máximo.

Al utilizar OpenMP la implementación es sustancialmente más sencilla, pero los tiempos de los resultados muestran que el rendimiento empeora un poco.