Considerando un filtro particular, comparar la performance de ejecución en su versión Single-thread y Multi-thread

– ¿Qué se puede decir sobre la performance del filtro en función de la cantidad de threads utilizados?

A	B	C
	Time	Thread
	0.649693	1
	0.660025	1
	0.651926	1
	0.647391	1
	0.648716	1
	0.650423	1
	0.64851	1
	0.655554	1
	0.646986	1
	0.668295	1
	0.641084	1
	0.650245	1
	0.643601	1
	0.65536	1
	0.645866	1
	0.648128	1
	0.646984	1
	0.713699	1
	0.649387	1

A	B	C
	Time	Thread
	0.0154257	20
	0.0164722	20
	0.0166601	20
	0.0186396	20
	0.0172273	20
	0.0193169	20
	0.0157678	20
	0.0161197	20
	0.0125425	20
	0.0185251	20
	0.0160328	20
	0.0159513	20
	0.017593	20
	0.0183933	20
	0.0186461	20
	0.0170829	20
	0.0164533	20
	0.0165193	20
	0.0179366	20
	0.0163045	20
	0.0209718	20
	0.0170383	20
	0.0201817	20
	0.0160952	20
	0.0150099	20
	0.0152471	20
	0.0170576	20

A	B	C
	Time	Thread
	0.0150371	32
	0.0198112	32
	0.016885	32
	0.0187046	32
	0.0150479	32
	0.02317	32
	0.0200288	32
	0.0248279	32
	0.0194148	32
	0.0239043	32
	0.113239	32
	0.0155591	32
	0.0166149	32
	0.0180099	32
	0.0179973	32
	0.0172004	32
	0.0167633	32
	0.0172514	32
	0.0185023	32
	0.0177937	32

Basándonos en la aplicación del filtro “Contrast”, en la imagen “totoro.ppm”, podemos observar que si usamos un único thread, claramente se nota una velocidad deficiente a comparación del mismo filtro pero con la implementación de una mayor cantidad de threads, veinte o treinta y dos por ejemplo.

Aunque, durante la implementación de estos últimos, no observamos una gran diferencia de velocidad. Esto nos hace pensar que no siempre más threads equivale a mayor rapidez, sino que en un momento dado, se va a llegar a un pico y el tiempo que conlleva la ejecución comienza a ser casi el mismo de forma constante, por lo que podríamos decir que el procesador de la PC es responsable de esto, ya que llegó a un máximo de threads reales en uso. Si a esto le sumamos que se consume tiempo al crear threads y asignarles valores, terminamos con resultados que indican que a una mayor cantidad de threads termina siendo contraproducente o la diferencia de tiempo no logra ser mayor.

– ¿Qué impacto tiene considerar imágenes "grandes" en lugar de imágenes "chicas"?

	Time	Thread
	0.0035609	1
	0.0012867	1
	0.0034934	1
	0.0020236	1
	0.0011434	1
	0.0020724	1
	0.001127	1
	0.0013807	1
	0.0011226	1
	0.001274	1
	0.0022011	1
	0.0031526	1
	0.0012345	1
	0.0015822	1
	0.0010663	1
	0.0017574	1
	0.0021762	1
	0.001174	1
	0.0010814	1
	0.0013087	1
	0.0017001	1
	0.0010475	1

	Time	Thread
	0.0034232	20
	0.0053642	20
	0.0039783	20
	0.0030045	20
	0.0055548	20
	0.0043396	20
	0.0034688	20
	0.0051139	20
	0.005023	20
	0.0048946	20
	0.0033715	20
	0.0054683	20
	0.0042549	20
	0.0044409	20
	0.0098142	20
	0.0038581	20
	0.0046008	20
	0.0043896	20
	0.0039638	20
	0.0040665	20
	0.0045955	20
	0.0042598	20
	0.010452	20
	0.0065207	20
	0.0046662	20
	0.0418895	20
	0.0048061	20
	0.0040815	20
	0.0042155	20

	Time	Thread
	0.0074438	32
	0.0079544	32
	0.0080029	32
	0.0077792	32
	0.0053198	32
	0.0079767	32
	0.0059096	32
	0.0067918	32
	0.0064897	32
	0.0072563	32
	0.0085229	32
	0.007036	32
	0.0093471	32
	0.0067811	32
	0.0057439	32
	0.0061782	32
	0.005742	32
	0.0048395	32
	0.0040455	32
	0.003804	32

En esta experimentación seguimos con el filtro “Contrast” pero usamos la imagen “house-1.ppm”, si la comparamos con la imagen “totoro.ppm”, en cuanto a tiempos, podemos observar que en las imágenes de menor tamaño con la implementación de un solo thread va a ser más rápido que con más threads, justamente retomando que más thread implica pasar por el proceso de creación más veces y hasta inclusive encontrarnos con threads que no hacen aportes debido a que la imágen ya está cubierta por otros.

Podemos atribuir la culpa de este retraso a que cuando se usan muchos threads, estos tienen que esperar para que se les asignen los valores particulares y ejecutarse, en una imagen chica esto genera lentitud en el programa ya que debe ir al disco en cada cambio y esto lo ralentizará, en cambio con un solo thread esta espera no es ocasionada en gran medida.

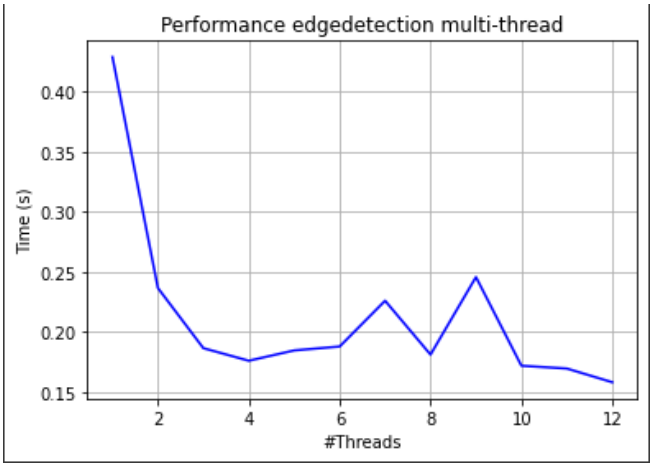
– ¿Cuán determinante es la configuración de hardware donde se corren los experimentos y cómo puede relacionarse con lo observado?

La configuración del hardware es muy determinante porque, no es lo mismo hacer estos experimentos en un núcleo de un único thread, que en una PC de varios núcleos con treinta y dos threads. Esto va a afectar a nuestro programa porque cada PC va a tener un límite de threads reales que va a usar y cuando lleguen a ese límite, el tiempo de ejecución, va a comenzar a ser casi el mismo de forma constante o inclusive tardar más, ya que los threads faltantes van a comenzar a funcionar en una especie de simulación a la vez que modificar partes de la imágen ya modificadas.

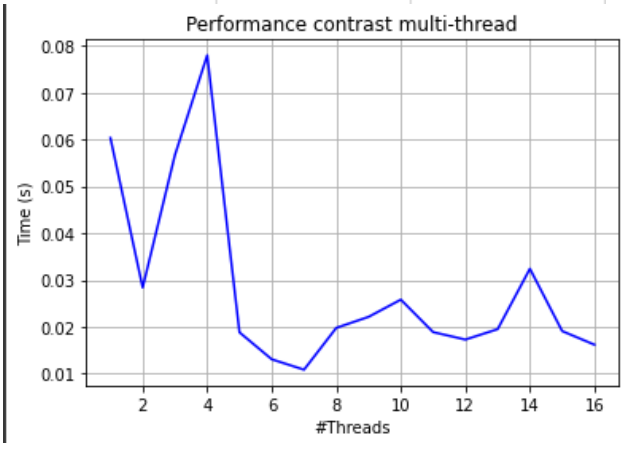
¿Hay diferencias de performance para los distintos tipos de filtros Multi-thread?

Edge detection | | Contrast

0.428663	1
0.236716	2
0.186718	3
0.176266	4
0.184848	5
0.188093	6
0.226146	7
0.181355	8
0.245835	9
0.172102	10
0.169793	11
0.158392	12



	Time	Thread
	0.0603327	1
	0.0283955	2
	0.0566828	3
	0.0779619	4
	0.0188312	5
	0.0130789	6
	0.01086	7
	0.0197893	8
	0.0221343	9
	0.0258127	10
	0.0188826	11
	0.0172881	12



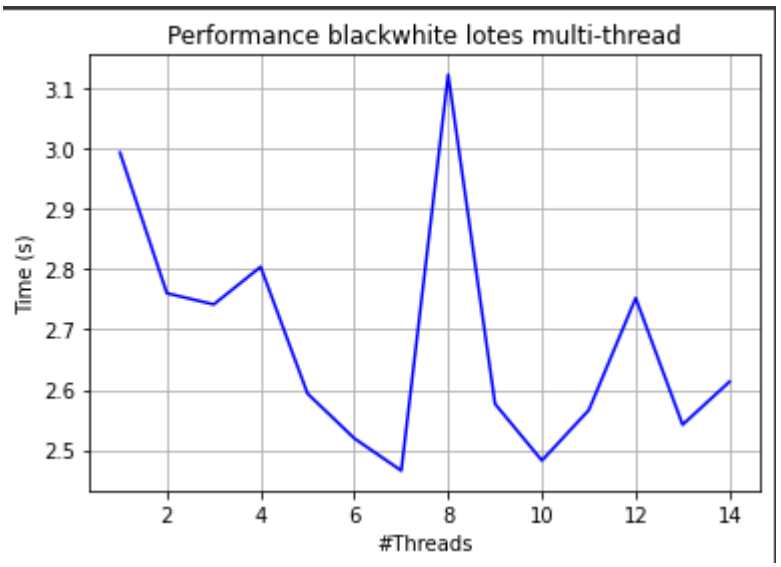
Si tomamos de referencia los filtros Edge Detection y Contrast podemos decir que no hay diferencia. En estas 2 experimentaciones vemos que en la implementación single thread el tiempo de ejecución es bastante largo a comparación al tiempo de ejecución con la implementación multithread. En conclusión en estos 2 filtros la perfomance del multithread es igual de efectiva.

En base a lo visto, ¿siempre es conveniente paralelizar? ¿De qué factores de la entrada depende esto?

En base a lo visto, depende siempre de la imagen a usar y el filtro elegido, por ejemplo el tamaño de la imagen y de qué tipo de filtro se va a usar. Entonces podemos decir que no es conveniente paralelizar en un filtro como edge detection, en el cual si aplicamos multithread la performance no cambia notablemente, lo mismo pasa si aplicamos cualquier filtro en una imagen pequeña. Por otra parte en la aplicación de Contrast vemos que es conveniente paralelizar ya que la performance mejora.

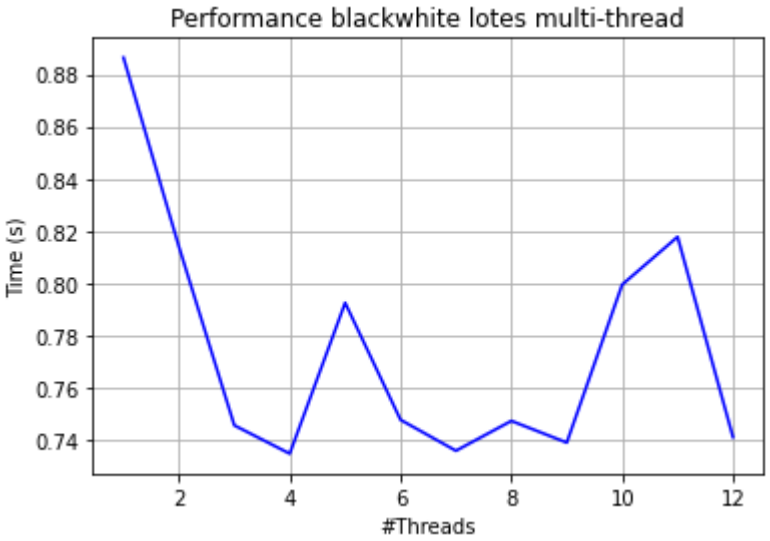
Considerar un análisis similar, pero para el caso del loader single-thread y Multi-thread, teniendo en cuenta también el tamaño de los lotes a experimentar.

	Time	Thread
	2.99372	1
	2.76013	2
	2.7416	3
	2.80403	4
	2.59354	5
	2.519	6
	2.46554	7
	3.12327	8
	2.57637	9
	2.48234	10
	2.56616	11
	2.75224	12
	2.54183	13
	2.61323	14



Si experimentamos con la carpeta “imgs” con trece imágenes de distintos tamaños y le aplicamos el filtro Black White podemos notar que en la implementación de single thread el tiempo de ejecución es más largo que implementando multithread donde se presenta una gran mejora.

0.886334	1
0.81408	2
0.74579	3
0.734973	4
0.792622	5
0.747899	6
0.736078	7
0.747488	8
0.739191	9
0.799575	10
0.817841	11
0.741283	12



Ahora si lo experimentamos con una carpeta con tres imágenes podemos notar que la implementación singlethread sigue siendo lenta a comparación a la implementación multithread. En conclusión, en la carga de un filtro por lotas podemos afirmar que siempre y cuando las imágenes del lote sean de un tamaño moderado o grande, es conveniente paralelizar. Por otra parte, si las imágenes son muy pequeñas o se trata de pocas imágenes en general, quizás no es tan conveniente paralelizar ya que no se vería una mejora importante en el tiempo.

Por lo tanto concluimos que en general, si el hardware de nuestra PC lo permite, podemos afirmar que generalmente es conveniente paralelizar. Quizás se den los casos con imágenes pequeñas o una cantidad reducida de estas, que la diferencia en la demora sea escueta o minúscula. A la vez que entendemos que no todos los filtros terminan por demorar lo mismo debido a que algunos cuentan con más complejidad que otros, por lo tanto las mayores diferencias entre single y multi thread se observan en filtros que requieren de una mayor complejidad. Además de que no siempre más threads termina por ser lo que rinde mejor, ya que hay cierto punto donde se necesitan una mayor cantidad de threads para poder observar una diferencia en cuanto a la performance del tiempo. Esto se debe a que la relación entre threads y tiempo la entendemos como una curva que comienza de forma exponencial pero luego termina amesetandose, ya que se requieren demasiados threads para ver una mejora notable.