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 foto “totoro.ppm” podemos ver que si usamos solo 1 thread claramente se nota una velocidad deficiente a comparación del mismo filtro pero con la implementación de 20 o 32 threads.

Aunque entre la implementación de estos últimos no hay una gran diferencia de velocidad. Esto nos hace pensar que no siempre más threads equivale a mayor rapidez, sino que en un momento 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.

– ¿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” y comparandola en la experimentación con la imagen “totoro.ppm” podemos ver que en las imagenes mas chicas con la implementación de un solo thread va a ser más rápido que con más threads.

Esto es ocasionado porque cuando se usan muchos threads estos tienen que esperar para que se les asigne un píxel, 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.

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

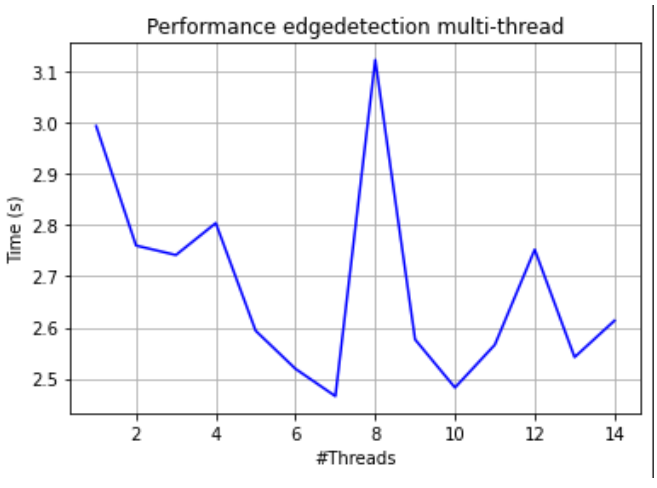
La configuración del hardware es muy determinante porque no es lo mismo hacer estos experimentos en un núcleo de 1 thread a una pc de varios núcleos con 32 threads. Esto va a afectar a nuestro programa porque cada PC va a tener un límite de threads reales que va a usar y cuando lleguen a ese limite el tiempo de ejecución va a comenzar a ser casi el mismo de forma constante, ya que los threads faltantes van a comenzar a funcionar en una especie de simulación.

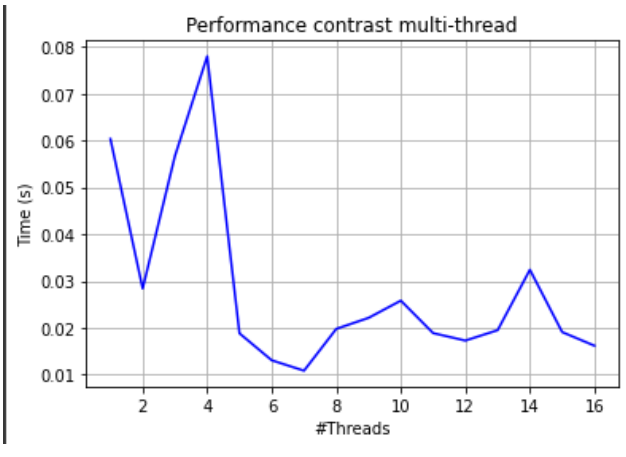
• ¿Hay diferencias de performance para los distintos tipos de filtros Multi-thread?

Edge detection | | Contrast

	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

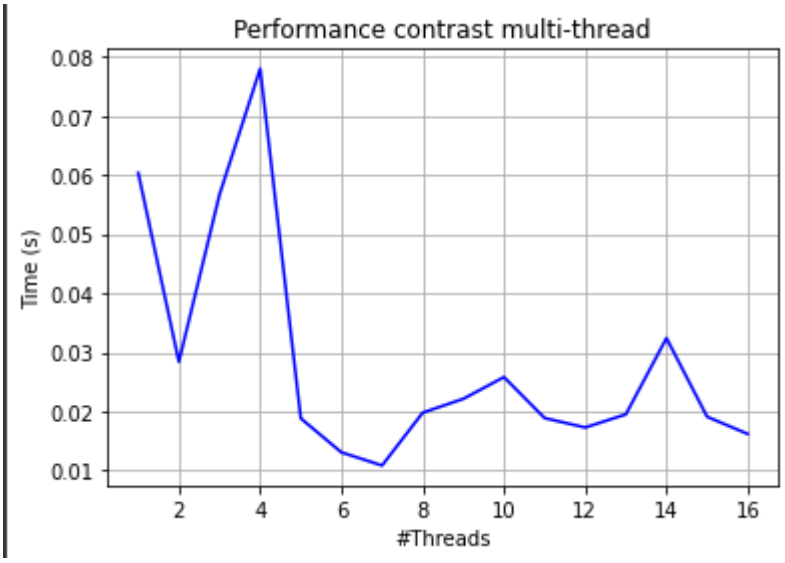
	Time	Thread
	0.0603327	1
	0.0283955	2
	0.0566828	3
	0.0779619	4
	0.0188312	5
	0.0130789	6
	0.01086	7
	0.0197893	8
	0.0221343	9
	0.0258127	10
	0.0188826	11
	0.0172881	12
	0.0195092	13
	0.0324035	14
	0.0191016	15
	0.0161924	16

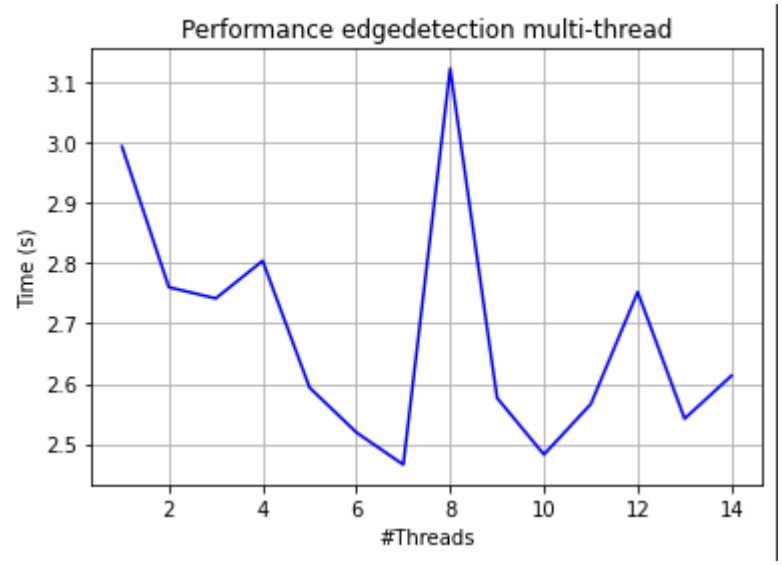




Si las hay y esto se puede ver en el filtro Contrast y Edge Detection. En la experimentación de Contrast podemos notar que en la aplicación multithread hay una mejoría notable, pero en el filtro Edge Detection el tiempo de ejecución es casi el mismo por lo tanto no siempre es conveniente usar muchos threads.

• Existen instrucciones de Assembler (SSE en X86) que permiten vectorizar algoritmos. De esta forma, con una sola instrucción SE podría aplicar una operación a varios elementos de una matriz en paralelo. Desde gcc es posible activar flags para que el código compile con optimizaciones y se fuerce el uso de este tipo de instrucciones. Indagar sobre esto y complementar los experimentos anteriores con este nuevo aspecto.



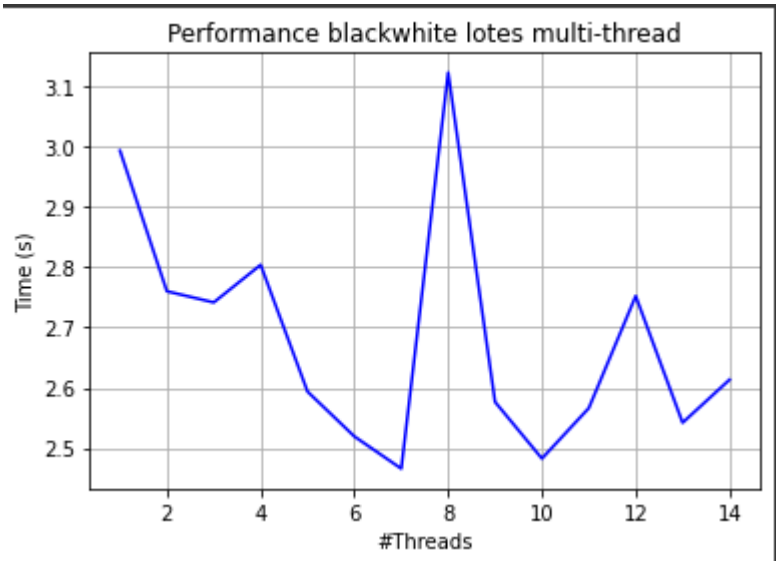


• En base a lo visto, ¿siempre es conveniente paralelizar? ¿De qué factores de la entrada depende esto?

En base a lo visto depende siempre de la imagen a usar y el filtro elegido, por ejemplo el tamaño de la imagen y el filtro 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 la carga de un filtro en lotes, por ejemplo Black White, podemos notar que entre 1 thread a más hay una mínima diferencia.