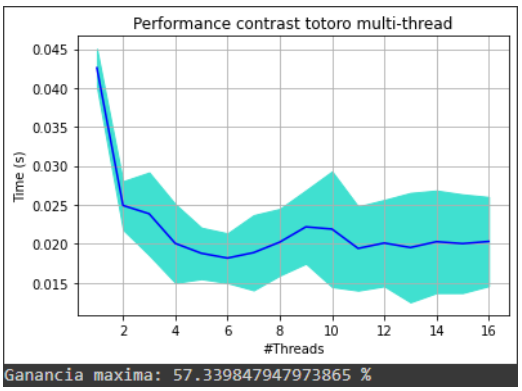
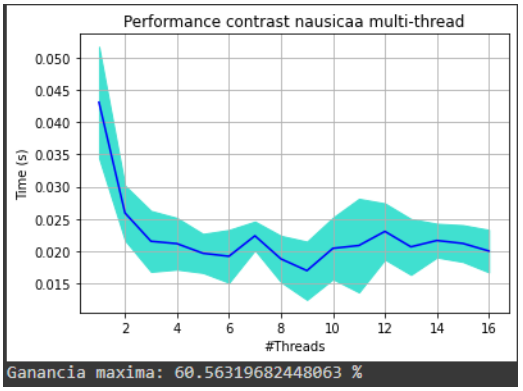
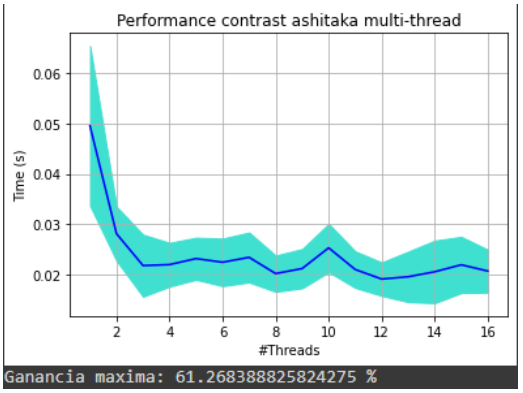
Correcciones en la reentrega:

- Brightness existía antes, solo que no se invocaba en main, pero si lo hacía en el loader, contrast se comporta mejor.
- El loader ahora sí paraleliza la carga y no las imágenes individualmente, a la vez que evita tener threads sin hacer nada.
- En cuanto a la web se deberían de invocar correctamente los filtros en su totalidad.
- En cuanto a la implementación, nunca usamos multithread de un solo thread, el código decidía si aplicar la versión multithread o la normal, que no usa threads, según la cantidad de threads.
- Implementamos más casos de prueba y un análisis respecto a activar más flags.

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?



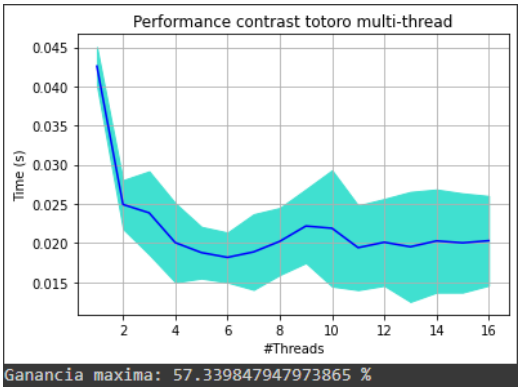


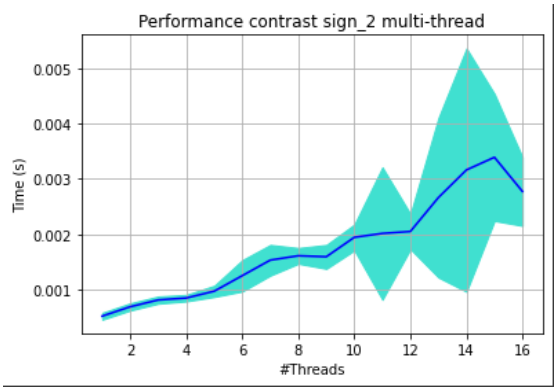


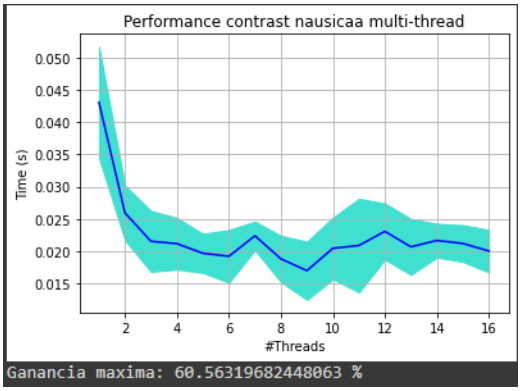
Basándonos en la aplicación del filtro “Contrast”, en la imagen “totoro.ppm”, podemos observar que si usamos un único thread, claramente se nota una velocidad deficiente a comparación del mismo filtro pero con la implementación de una mayor cantidad de threads, haciendo diez intentos con cada thread con tal de obtener un promedio.

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"?







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.

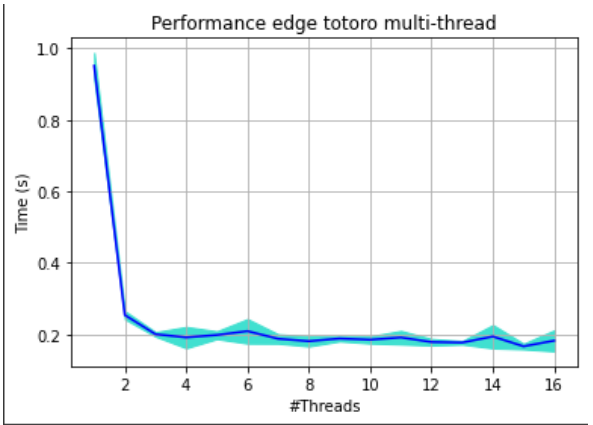
Podemos atribuir la culpa de este retraso a que cuando se usan muchos threads, estos tienen que esperar para que se les asignen los valores particulares y ejecutarse, en una imagen chica esto genera lentitud en el programa ya que debe ir al disco en cada cambio y esto lo ralentizará, en cambio con un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

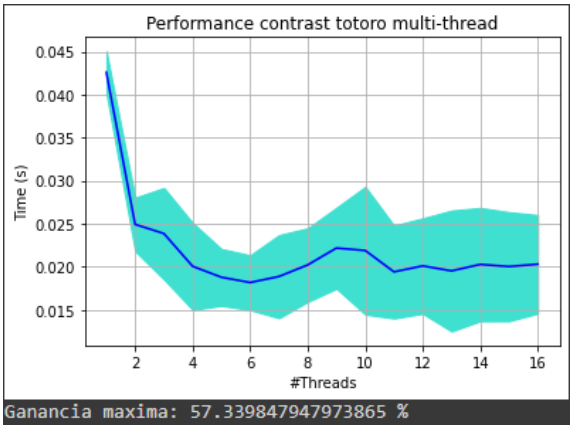
– ¿Cuán determinante es la configuración de hardware donde se corren los experimentos y cómo puede relacionarse con lo observado?

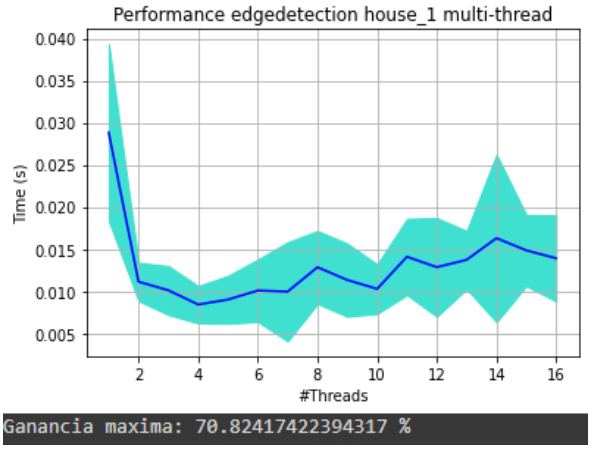
La configuración del hardware es muy determinante porque, no es lo mismo hacer estos experimentos en un núcleo de un único thread, que en una PC de varios núcleos con treinta y dos threads. Esto va a afectar a nuestro programa porque cada PC va a tener un límite de threads reales que va a usar y cuando 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.

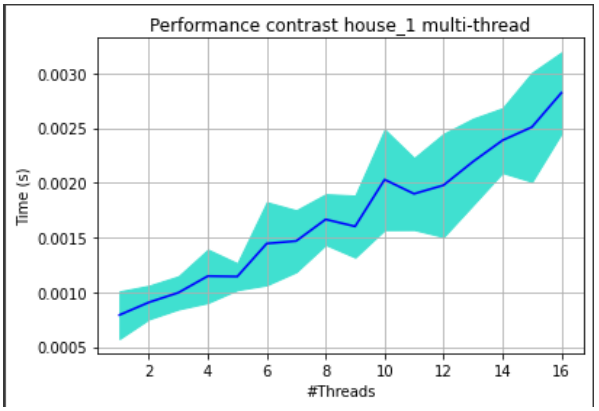
¿Hay diferencias de performance para los distintos tipos de filtros Multi-thread?

Edge detection (Convolution) | | Contrast(Pixel-to-Pixel)





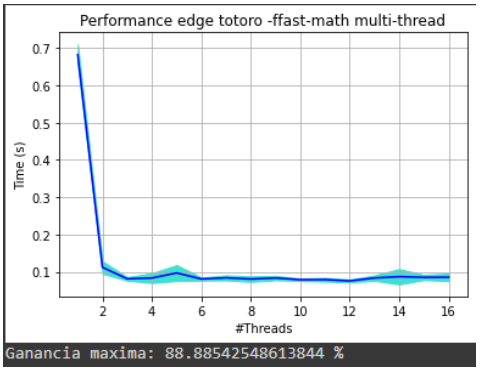


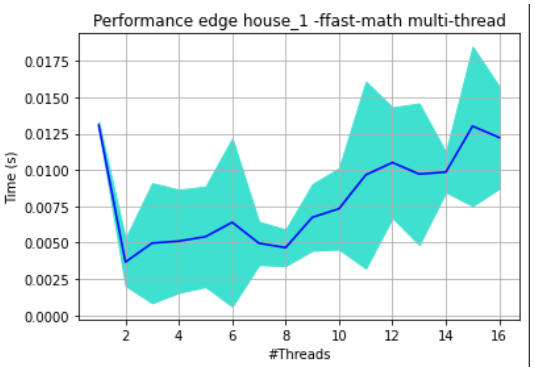


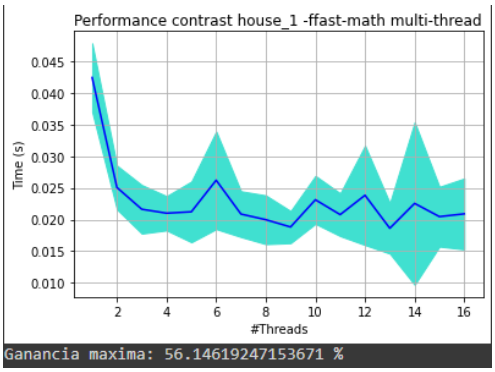
Entendemos que el tipo de filtro es aquel que determina si se puede observar una mejora notable en cuanto al rendimiento o no. En filtros del tipo pixel to pixel, en este caso contrast, notamos una cierta mejora pero que no es comparable con la alta mejora de rendimiento que se ve en filtros más complejos del tipo convolution, como lo es edge detection. La mejora observable se debe principalmente a que los filtros pixel to pixel revisan y modifican un único píxel para dar un resultado, mientras que aquellos filtros del tipo convolution requieren revisar nueve pixeles para poder entregar un único píxel de resultado, por lo que si se paraleliza este proceso con threads, es mucho más abrupta la mejora de performance que en otro tipo de filtros.

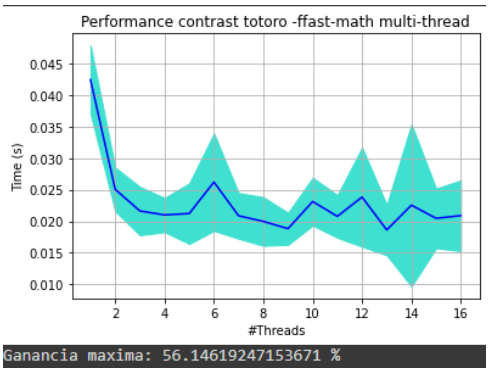
Existen instrucciones de Assembler (SSE en X86) que permiten vectorizar algoritmos. De esta forma, con una sola instrucción SSE podría aplicar una operación a varios elementos de una matriz en paralelo. Desde gcc es posible activar flags para que el código compile

con optimizaciones y se fuerce el uso de este tipo de instrucciones. Indagar sobre esto y complementar los experimentos anteriores con este nuevo aspecto.









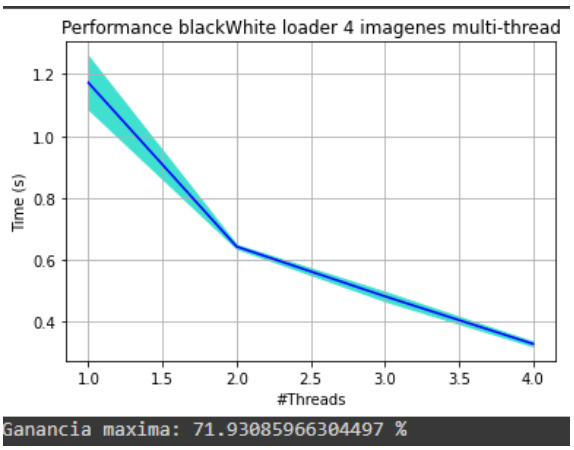
Teniendo en cuenta los experimentos anteriores, sobre multithread, tamaños de imágenes y tipos de filtros, logramos observar una mejora importante en cuanto a los tipos al utilizar la la flag “-ffast-math”. Nos decantamos por utilizar esta ya que al realizar pruebas con otros tipos que engloban una mayor cantidad de flags, como lo son “-o3”, “-os” o “ofast” nos encontramos con que las mejoras eran imperceptibles. Comprendemos que el motivo de esto es que al aplicarse tantas flags, aquellas que generaban mejoras en la performance, esta mejora se veía opacada por aquellos chequeos o mejoras que hacían las otras flags que no eran necesarias. Por lo tanto al utilizar esta única flag, que se encarga de hacer optimizaciones relacionadas a las cuentas matemáticas, observamos que aquellos filtros como edge detection, sean imágenes grandes o pequeñas, muestran una clara mejora en cuanto a la performance. Mientras que aquellos filtros que utilizan menos cuentas matemáticas, no muestran mejoras en cuanto a performance sino que se mantienen o demorar un poco más al igual que su contraparte sin la flag activada, una vez más el tamaño no afecta.

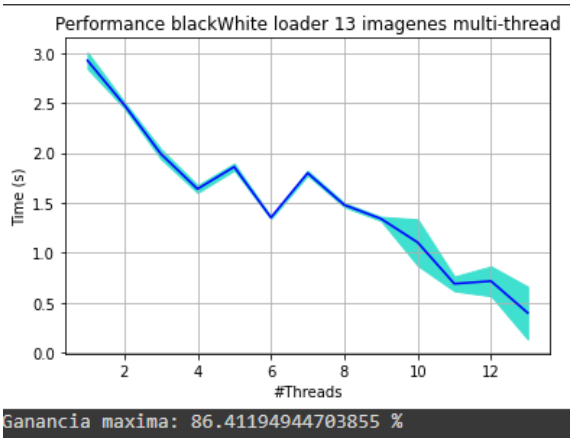
En base a lo visto, ¿siempre es conveniente paralelizar? ¿De qué factores de la entrada depende esto?

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 es conveniente paralelizar en un filtro como edge detection, en el cual si aplicamos multithread la performance si cambia notablemente, distinto es si aplicamos cualquier otro tipo de filtro que no requiere de complejidad, donde se terminará por necesitar más threads para observar una mejora notable en la performance. De tratarse de imágenes pequeñas, al

aplicar demasiados threads observamos que se puede llegar al límite de espacio dentro de una imagen, teniendo threads que no se usan, además de que el tiempo de creación y asignación de threads

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.





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. Generalmente lo que observamos es que no importan los tamaños de imágenes o el tipo de filtro, esto debido a que los threads siempre le van a sacar trabajo a los anteriores y por consecuencia va a ser más rápido que si lo hiciera un único thread, inclusive si el input de threads es mayor a la cantidad de imágenes, el programa regula esto al hacer que la cantidad máxima de threads sea la cantidad de imágenes total. De no hacerse esta regulación de threads, observaríamos que se crean threads sin uso o que realizan las mismas acciones que otros threads ya hicieron. En este caso termina por ser despreciable el tiempo que se demora en crear threads y asignarles imágenes respectivamente.

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. Además si sumamos el uso de flags adecuados en las performance de los filtros podemos llegar a una mejora considerable del programa, de no aplicarse las flags correctas podríamos encontrarnos ante una situación donde no se observan mejoras o se empeora la performance del programa.