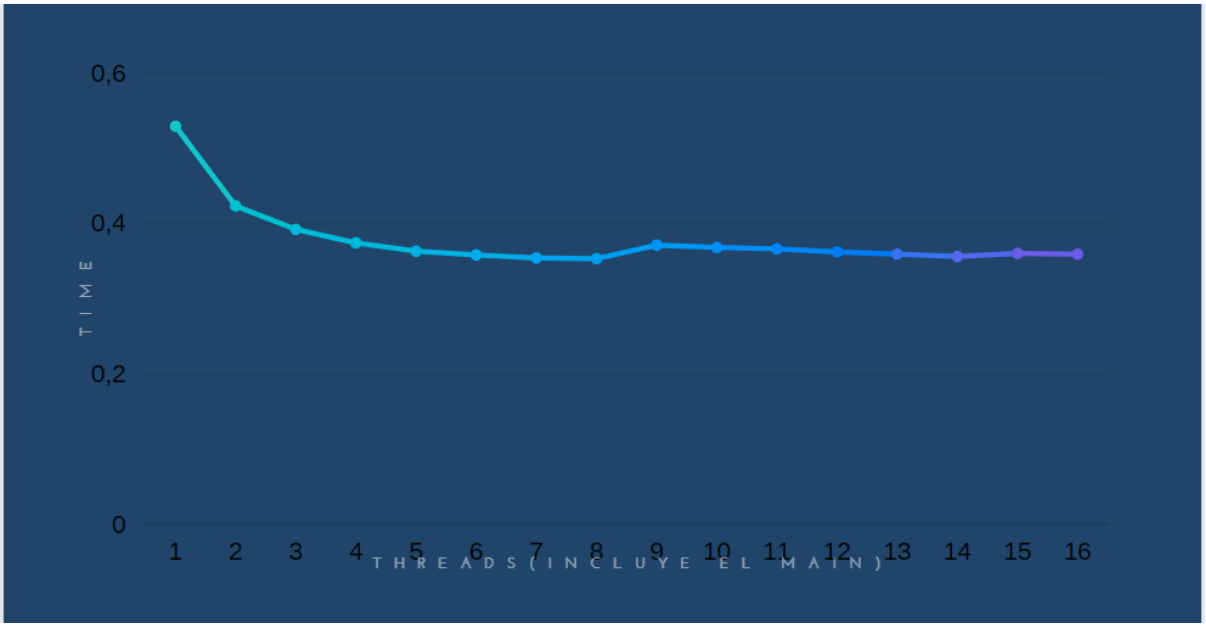
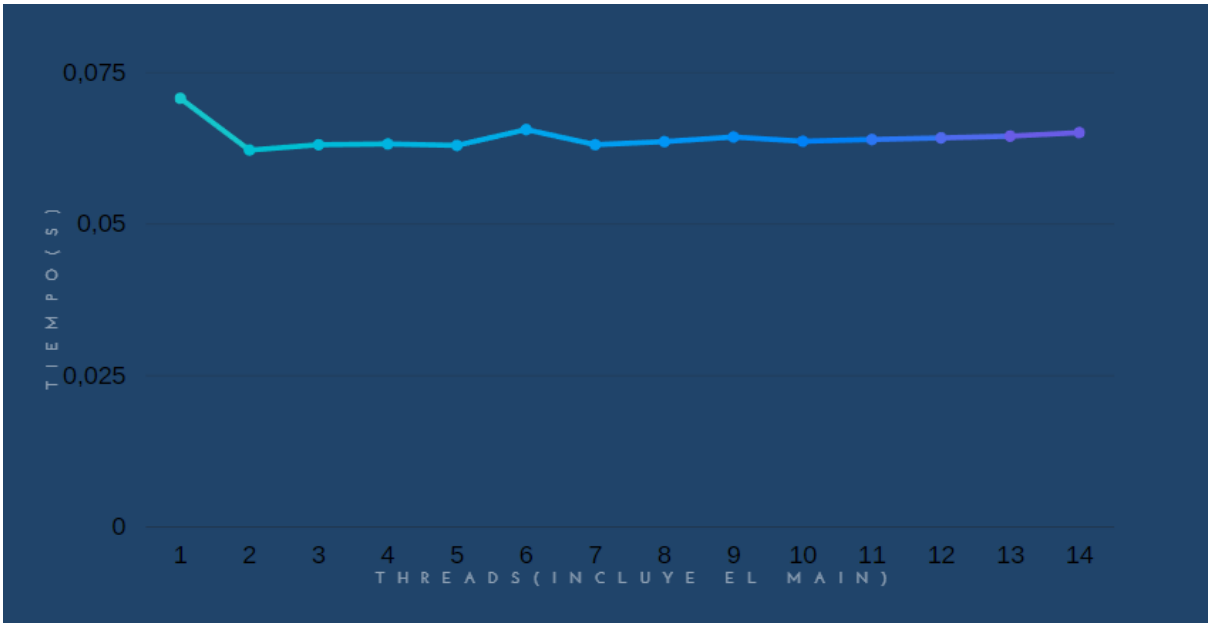
Considerando un filtro particular, comparar la performance de ejecución en su versión Single-thread y Multi-thread:

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



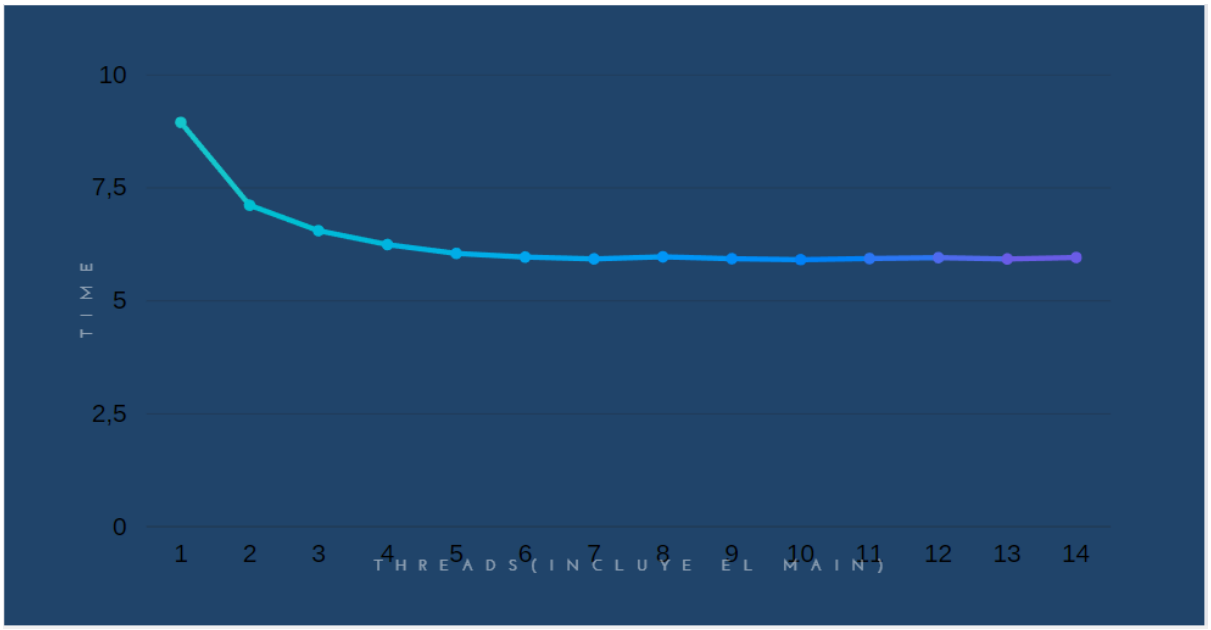
Consideramos para hacer las pruebas el filtro del zoom. Como se puede observar en el gráfico, la performance empieza desde su punto más alto en el single thread (1) y va descendiendo a medida que se le van aumentando los threads. Sin embargo, solo desciende hasta los 8 threads (7 mas el main). Luego, aumenta de golpe un poco y vuelve a reducirse a medida que se aumentan los threads. No sería correcto decir que a cuantos más threads, menor el tiempo.

Tras aplicar las optimizaciones utilizando el comando “-O3”, se puede decir que la performance cambió completamente ya que ahora el proceso dura mucho menos tiempo en comparación al programa no optimizado, la brecha entre el single-thread y el multi-thread se redujo exponencialmente y además a medida que aumentan los threads no se reduce el tiempo sino que se mantiene constante o aumenta



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

El impacto que va a tener es un aumento significativo del tiempo de procesamiento de la imagen ya que primero se debe poder cargar la imagen en una variable y luego cada thread debe recorrer su parte correspondiente de dicha imagen. Sin embargo, la relación de optimización no varía demasiado pero lo que sí es curioso es que se mantiene de manera más estable a medida que van aumentando los threads a diferencia del gráfico anterior.



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

Es bastante determinante ya que en base a las pruebas realizadas, sobre todo en imágenes chicas. El punto de mayor eficiencia conseguida con el programa, se logra cuando se

utilizan todos los threads del CPU. Sin embargo, si se utilizan más de los threads poseídos en la CPU, el tiempo puede aumentar o pegar un salto considerable. Un ejemplo es el primer gráfico, las pruebas fueron realizadas con una CPU con 8 threads. Cuando se quieren utilizar 9 threads, el tiempo pega un salto y se eleva repentinamente.

Tras aplicar las optimizaciones utilizando el comando “-O3”, el hardware se ha vuelto mas importante, sobre todo la memoria ya que al utilizar dichas optimizaciones se estan utilizando muchos mas recursos de la memoria. Por lo que, es importante si se quiere aplicar estas optimizaciones, el tener memoria suficiente para soportarlos

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

No necesariamente, la mayoría de los filtros multi-thread mantienen una performance similar con respecto al hecho de que el single.thread siempre demora mucho más en relación con el multi-thread y que el tiempo se va reduciendo a medida que se aumentan la cantidad de threads hasta cierto punto

Existen instrucciones de Assembler (SSE en X86) que permiten vectorizar algoritmos. De esta forma, con una sola instrucción SSE podría aplicarse 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?

No siempre es conveniente paralelizar ya que si se utilizan demasiados threads, puede aumentar el tiempo necesario para poder realizar el filtro requerido. Además, si se utilizan más threads de los que tiene la computadora puede resultar ineficiente. El valor principal a considerar es la relación de tiempo entre: cuanto tarda en crearse un thread y cuanto tarda en procesarse ese pedazo de la imagen. Si se demora más en crear el thread entonces no renta, esto es lo que sucede con las imágenes con muy pocos pixeles ya que si ponemos 100 threads para una imagen de 100 pixeles, probablemente sea más rápido procesarla de una que crear thread por thread para 1 pixel

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.

En nuestro caso el loader se hizo como una extensión de los métodos normales por lo que este llama al filtro normal por cada imagen que debe filtrar la opción multi thread convierte el filtrado en multi thread por lo que no hay una gran diferencia de rendimiento el tiempo que el loader tarde en filtrar un lote de imágenes se puede expresar como: tiempo que tarda el filtro normalmente x cantidad de imágenes