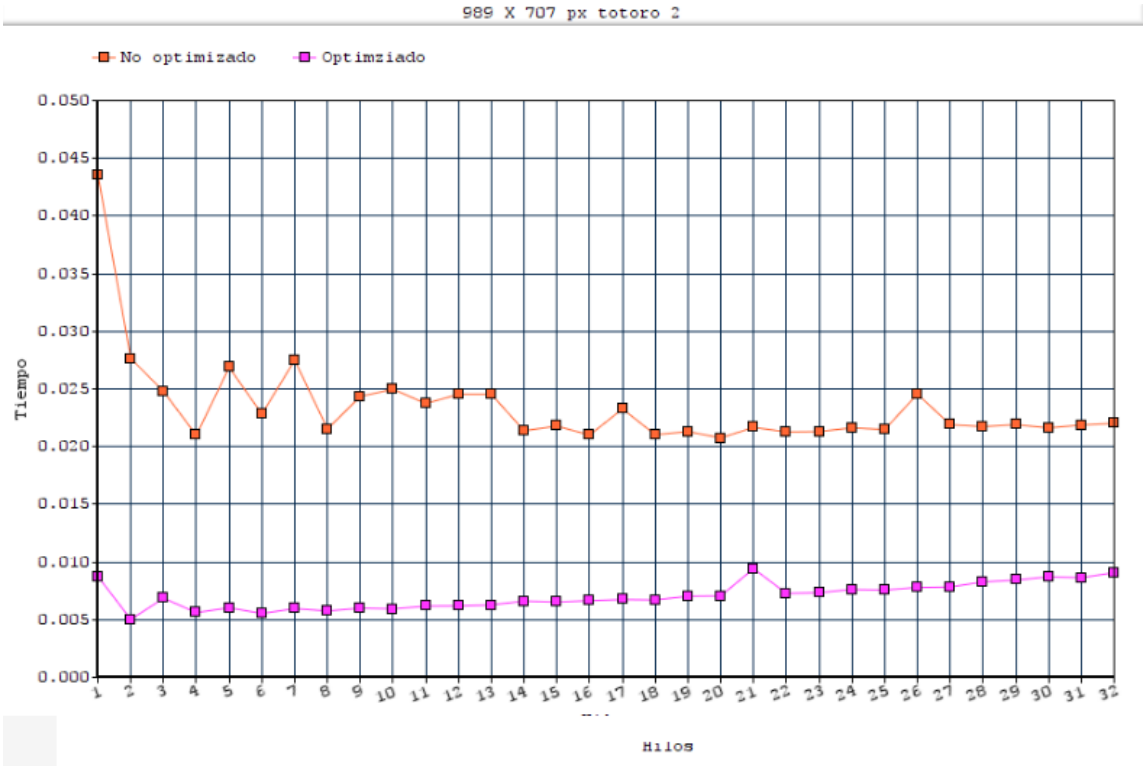
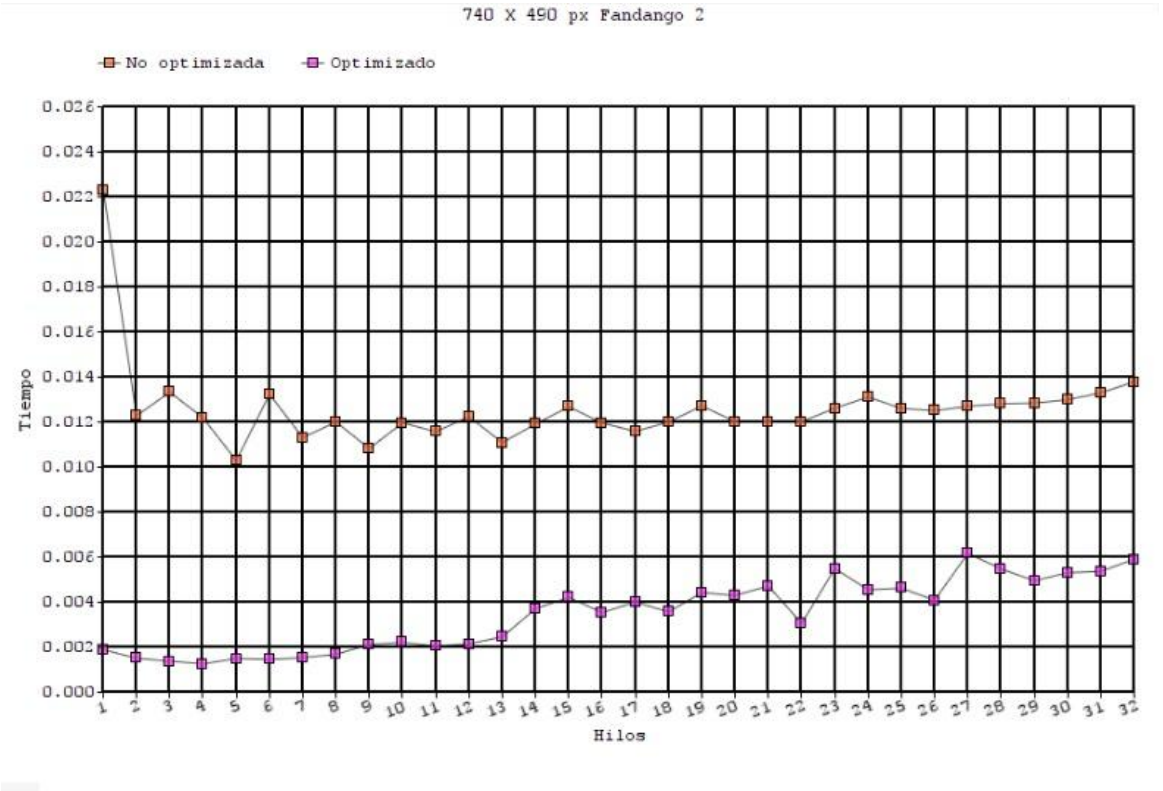
Informe

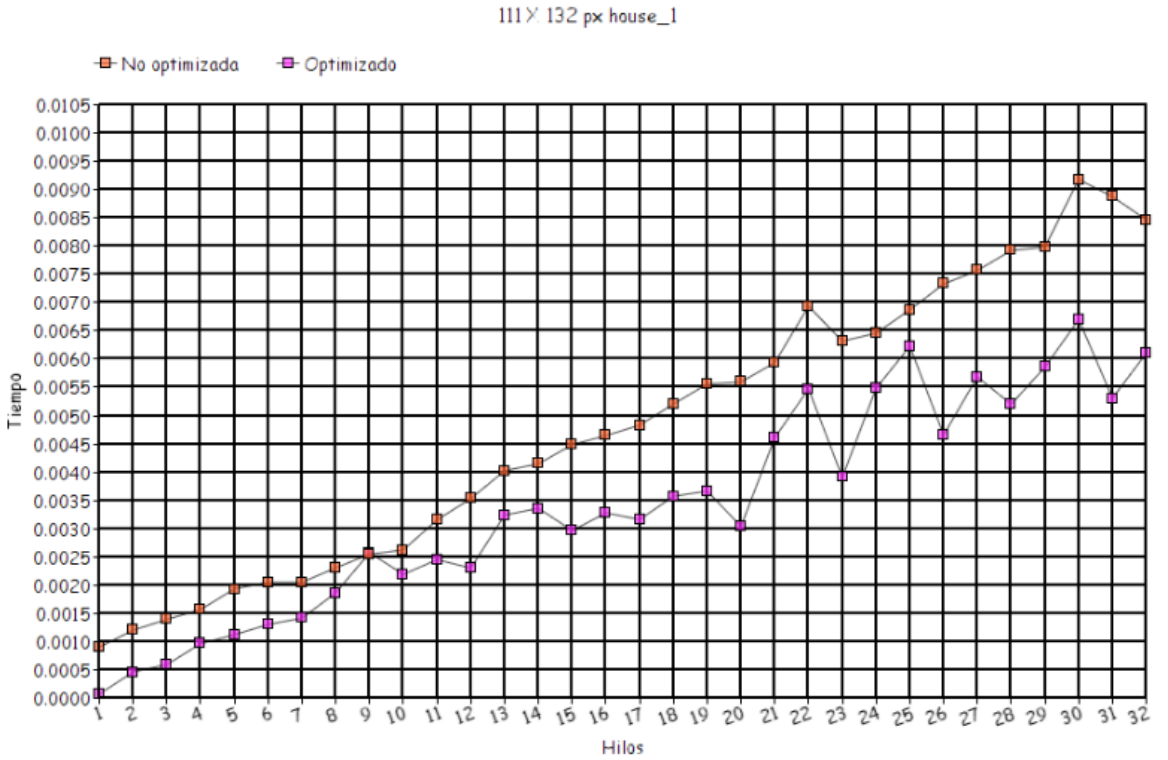
Alumnos: Ezequiel Coronel y Juan Manuel Costarelli.

Experimento 1: performance en función de la cantidad de threads

Lo primero que uno piensa es que dividir la tarea entre más threads debería ser siempre mejor ya que puede hacer todo el proceso al mismo tiempo. Esto lo veremos en los siguientes gráficos en los que se mostrará el tiempo que lleva el proceso,optimizado y sin optimizar, de ejecución del filtro de contraste en distintos tamaños de imágenes y con distintas cantidades de hilos.







Como vemos en los dos primeros gráficos el aumentar la cantidad de threads mejora la performance, con una ganancia del 51% y 53% en los mejores rendimientos, a comparación de tener un solo hilo, pero la mejor performance, en el caso del primer gráfico, pasa cuando hay solo 4 hilos para la ejecución, luego hay valores parecidos o peores. Pero en el tercer gráfico podemos ver que aunque aumentemos los threads no mejora el tiempo en comparación a tener un solo hilo, sino al contrario tarda más en ejecutarse.

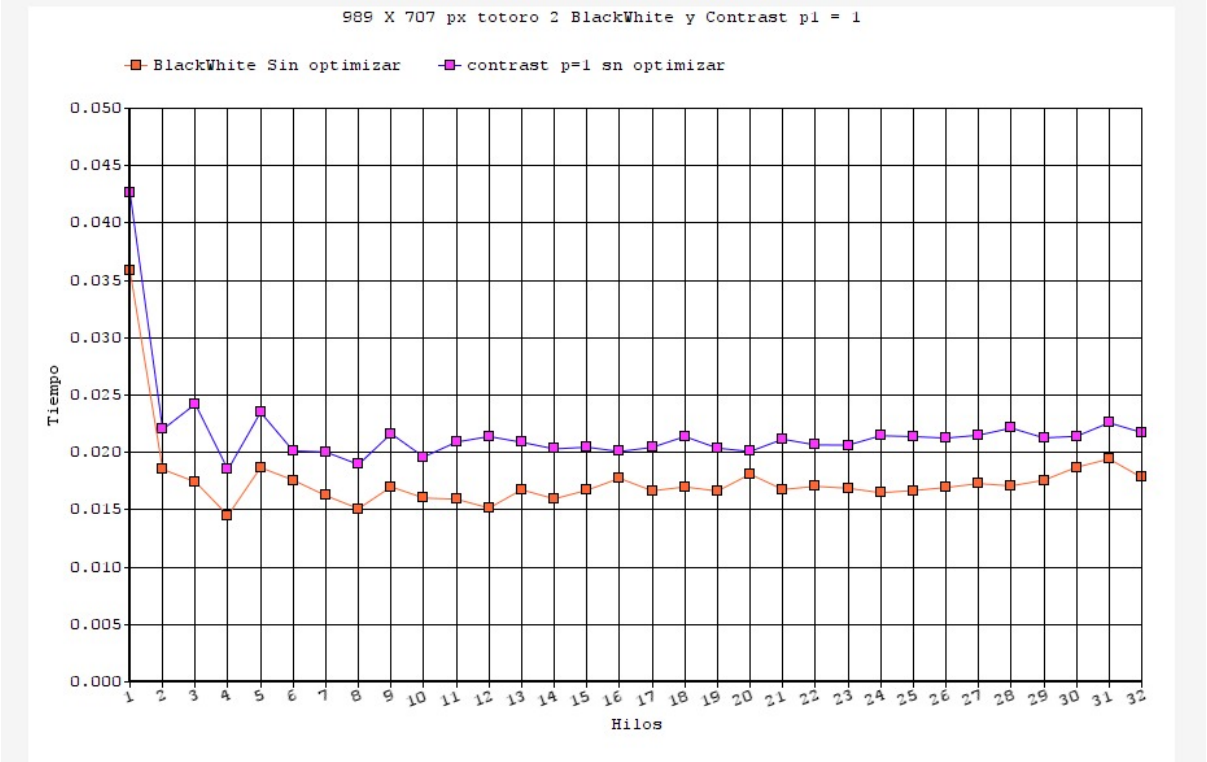
En cuanto a la optimización utilizamos -O3 el cual activa todas las flags de optimización para que nuestra ejecución tarde lo menos posible y en los gráficos se nota la diferencia, tarda mucho menos la ejecución cuando lo optimizas, pero también la mejor performance, del gráfico 2, es con 4 hilos.

Haciendo un análisis más a profundo del funcionamiento del -O3 tenemos la siguiente tabla:

option	optimization level	execution time	code size	memory usage	compile time
-O0	optimization for compilation time (default)	+	+	-	-
-O1 or -O	optimization for code size and execution time	-	-	+	+
-O2	optimization more for code size and execution time	--		+	++
-O3	optimization more for code size and execution time	---		+	+++
-Os	optimization for code size		--		++
-Ofast	O3 with fast none accurate math calculations	---		+	+++

Bien si nos ponemos a analizarla tenemos que el "-O3" nos damos cuenta que reduce muchísimo el tiempo de ejecución, lo cual se ve en la experimentación que hicimos. Ahora teniendo el punto a favor de usar esto, también existen puntos negativos, usa un poco más de la memoria RAM lo cual si disponemos de poca memoria RAM esto va a generar que a la hora de compilarlo con este "optimization mode" todos los recursos de dicha PC estarán centrados en esa tarea y no se podrá hacer nada más. También otra de las cuestiones es que aumenta muchísimo el tiempo de compilación.
Si analizamos la "FLAG" de "-O" el compilador intenta reducir el tamaño del código y el tiempo de ejecución, sin realizar optimizaciones que consumen mucho tiempo de compilación.Ahora pasando al "-O2" , sigue los mismos pasos que lo mencionado anteriormente, pero esta opción aumenta tanto el tiempo de compilación como el rendimiento del código generado. Pasando al que se usó en los experimentos ("-O3") optimiza aún más. Cabe destacar que estas son opciones que controlan la optimización y esto quiere decir que el objetivo del compilador es reducir el tiempo de ejecución y hacer que la depuración produzca los resultados esperados. Activar estas "FLAGS" de optimización hace que el compilador intente mejorar el rendimiento y el tamaño del código a expensas del tiempo de compilación y posiblemente de la capacidad de depurar el programa, lo cual esto está relacionado con lo hablado anteriormente menor tiempo de ejecución(en los experimentos se nota la diferencia reduciendo los tiempos de ejecución 5 veces más que lo que se tenía antes), más uso de memoria y muchísimo más tiempo de compilación.

Ahora durante esta fase de experimentación nos surge otra pregunta¿Hay diferencia entre los distintos filtros multi-threaders? para esto elegimos los filtros blackWhite y contrast.

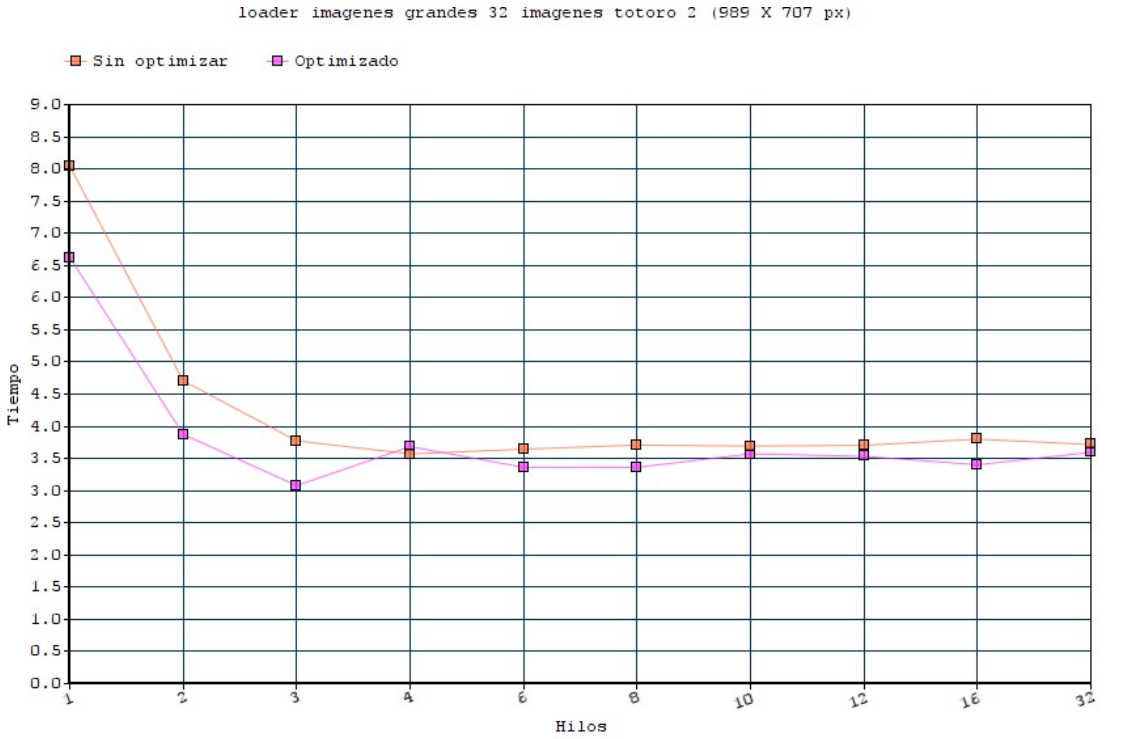


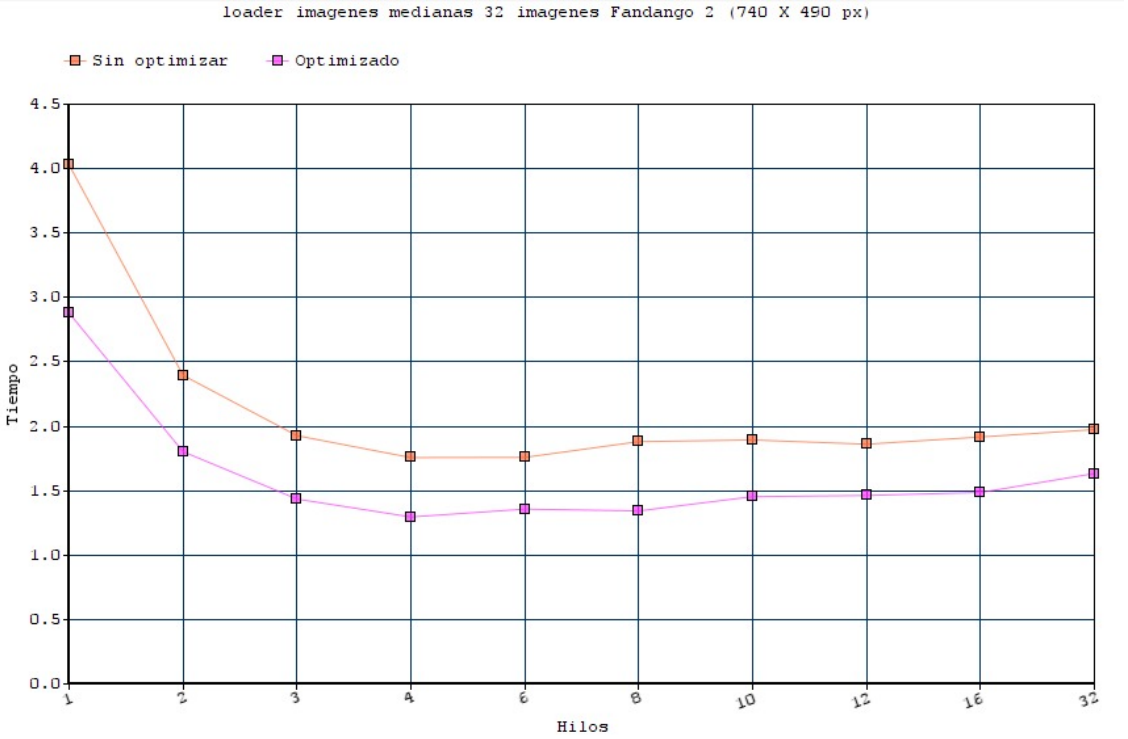
Vemos que hay una diferencia en la performance de los filtros, esto se debe a la complejidad del filtro, el código del filtro blackWhite pide los valores de un pixel los suma, divide y los setea, mientras que el contrast debe pedir los valores , aplicarles una fórmula para lograr el contraste, revisar que los números sean posibles y luego setearlos. También lo que se nota claramente es que la mejor performance, en los dos filtros , es cuando se ejecuta con 4 hilos. ¿Por qué pasa esto? La computadora sobre la que llevamos a cabo los experimentos tiene un procesador intel i5 4690K el cual tiene 4 hilos, por lo que es determinante el hardware para conseguir el mejor desempeño, cuantos más hilos tenga el procesador mejor rendimiento al paralelizar vas a conseguir porque el sistema no debe tomarse tiempo para crear el hilo.

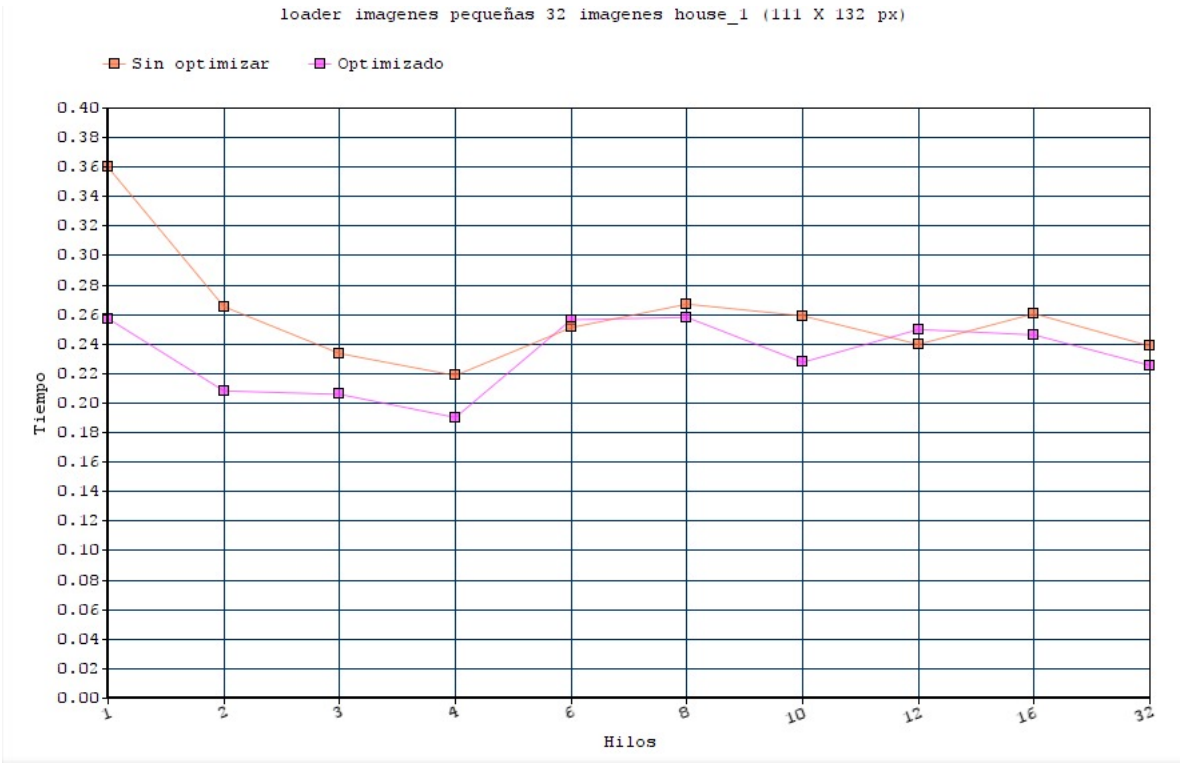
Luego de todos los experimentos llegamos a la conclusión de que no siempre es más conveniente paralelizar, para saber si corresponde primero hay que saber sobre nuestro hardware, cuantos hilos tiene nuestro procesador para leer la información sin tener que esperar a que se creen nuevos, tamaño de imagen, cuanto más pequeña la foto menos información hay que procesar por lo que no son necesarios tantos hilos para conseguir la mejor performance.

Experimento 2: performance del loader multi-thread en función del tamaño de lotes

Para este experimento empezaremos viendo las performance en lotes de 32 imágenes de un determinado tamaño con el filtro de contrast:



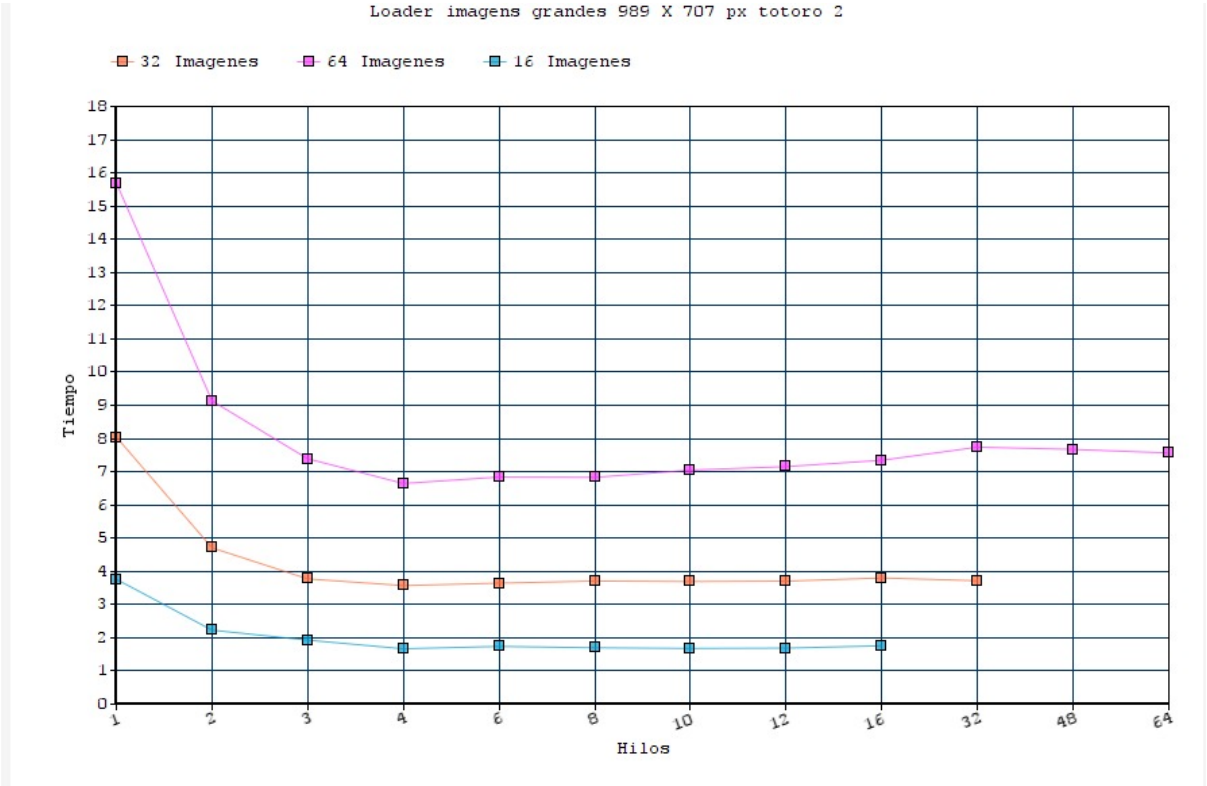




En los gráficos podemos notar la gran diferencia que aparece entre usar solo un hilo como lector y varios, la ganancia en la mejor performance es de 53% para el primer gráfico. En cuanto a la optimización baja el tiempo de ejecución pero el cambio es mucho menor, en comparación a lo que genera en los filtros, con los filtros el tiempo es 500% menor y en el loader el tiempo baja un 30%.

Vemos también que el tamaño de las imágenes es muy importante, en los gráficos el loader con imágenes grandes tarda 8 segundos, el de medianas 4 segundos y el de pequeñas 0.35 segundos,esto se debe a la información que hay que procesar cuanta menos haya más rápido terminará. Cabe destacar que nuestro loader como máximo utilizara un thread por imagen y por lo que vemos el tiempo no es malo pero no es la mejor performance, volvemos a notar la importancia del hardware porque nuestra mejor performance es con 4 hilos que son los hilos que trae nuestro procesador de fábrica.

Luego de probar lotes con imágenes de distintos tamaños ahora veremos un grafico con lotes de distintos tamaños pero con los mismos tamaños de imagen



Además de notar que la mejor performance es usando 4 hilos lectores, el tiempo que tarda es casi proporcional a la cantidad de imágenes que tiene el lote por ejemplo cuando ejecutamos un lote de 32 imágenes tarda 8 segundos si ejecutamos un lote con 64 imágenes tarda el doble, 16 segundos, y con un lote de 16 imagenes tarda 4 segundos.