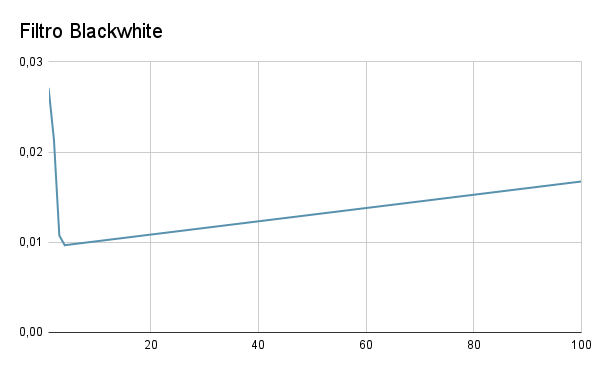
**Informe Imagine**

**Grupo: Joaquin Poggi & Lautaro Viola**

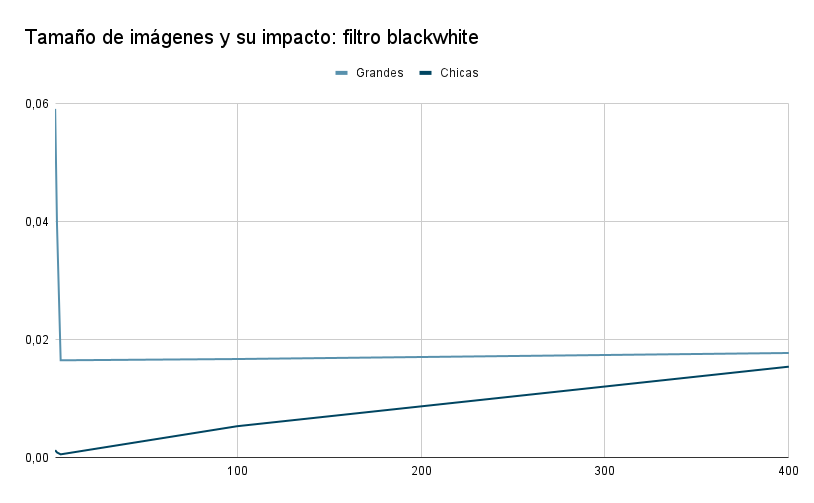
**Considerando un filtro particular, comparar la performance de ejecución en su versión Single-thread y Multi-thread. ¿Qué se puede decir sobre la performance del filtro en función de la cantidad de threads utilizados?**

Nosotros experimentamos con el filtro “Blackwhite”. En el gráfico adjunto se muestra que el rendimiento del procesamiento mejora hasta cierta cantidad de hilos, duplicando las mejoras por cada hilo agregado, hasta llegar al punto en donde empieza a ralentizar.



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

| Hilos | Imágenes grandes | Imagenes chicas |
| --- | --- | --- |
| 1 | 0.059113 | 0.001299 |
| 2 | 0.039545 | 0.000915 |
| 4 | 0.016526 | 0.000593 |
| 100 | 0.016742 | 0.005360 |
| 400 | 0.017764 | 0.015447 |



El gráfico nos muestra que en las imágenes grandes el rendimiento mejora hasta cierto número de hilos, y que después se estabiliza sin notarse la diferencia. Es conveniente hasta cierto punto, luego no afecta. En el caso de las imágenes chicas ocurre al revés. El rendimiento empeora exponencialmente hasta llegar a estabilizarse en el mismo punto casi que las imágenes grandes, pero con resultados contrarios.

El cambio viene por la creación de threads. En las imágenes chicas no es necesario tantos threads, ya que la misma creación tarda más que el procesamiento de las mismas, ralentizando el proceso total. En las imágenes grandes esto da mejoras hasta el punto en que la tarea se divide uniformemente en subtareas cortas de fácil procesamiento, por eso después de cierta cantidad de hilos, no hay cambios prácticamente.

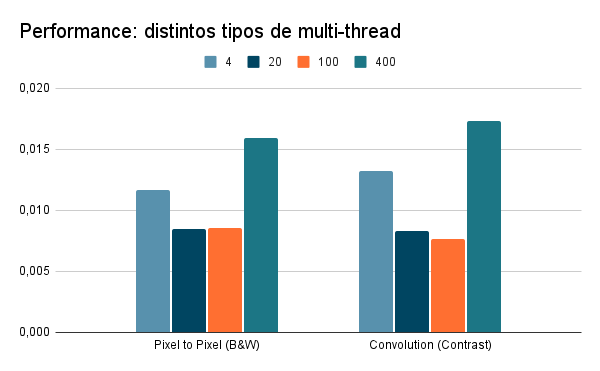
**¿Cuán determinante es la configuración de hardware donde se corren los experimentos y cómo puede relacionarse con lo observado?**

La configuración de hardware es determinante ya que es el que crea los hilos. Si tenemos mejor hardware, podremos crear más hilos, y además, en menos tiempo. Esto va a generar buen rendimiento hasta el punto en el que las tareas no superen a los hilos, y estos no tarden más tiempo en crearse, en lo que tardan en realizarse dos tareas en un mismo hilo, ya que se estarían creando hilos innecesarios.

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

Poniendo todos los filtros al máximo o explotandolos lo más posible en una misma foto, los resultados fueron:

| Hilos | 4 | 20 | 100 | 400 |
| --- | --- | --- | --- | --- |
| Pixel to Pixel (B&W) | 0.011697 | 0.08485 | 0.008594 | 0.015907 |
| Convolution (Contrast) | 0.013185 | 0.008327 | 0.007621 | 0.01732 |

****

Desde nuestro análisis, no hay cambios en los patrones. Probamos con un filtro Pixel to Pixel y con otro de Convolution. Cuando son pocos los hilos, no se notan mucho los cambios. Llegando a los veinte hilos los cambios se notan y se mantienen hasta los cien. Llegados a los cuatrocientos, el rendimiento empeora por la ineficiencia de tantos hilos. En el convolution los rendimientos son en la mayoría de casos peores, mostrando que el multi-thread mejora más el rendimiento en el filtro Pixel to Pixel.

**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.**

No llegamos a hacer este punto, profe, perdón. Hicimos nuestro mejor esfuerzo.

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

El paralelismo busca hacer una tarea más eficiente aprovechando la multiplicidad de recursos y distribuyendo particiones de la tarea en unidades de ejecución distintas. Esta es conveniente cuando las unidades de ejecución trabajan a la misma velocidad y con las mismas proporciones.

El paralelismo no conviene cuando la división de tareas es desproporcionada (más trabajo en unos que otros). Si las tareas son muy pequeñas, tampoco conviene, ya que se pierde más tiempo dividiendo tareas que el tiempo que se tarda en hacerlas directamente, por ende, conviene para tareas con un tamaño significante, y que valgan la pena ser divididas. Conviene cuando hay tareas muy grandes por procesar. El paralelismo se ocupa de agarrar una tarea muy grande, dividirla en varias más pequeñas, procesarlas por separado, y volver a unirlas al final, evitando que un procesador se quede “estancado” en una única tarea.