High-pass filter

Rodriguez Angenelo Santiago, Romano Pablo Martín, Grilli Matias Nahuel
DNI: 37109237, DNI: 40389631, DNI: 31915669
Dia de cursada: Miércoles noche, Numero de Grupo: 4

[1]Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina

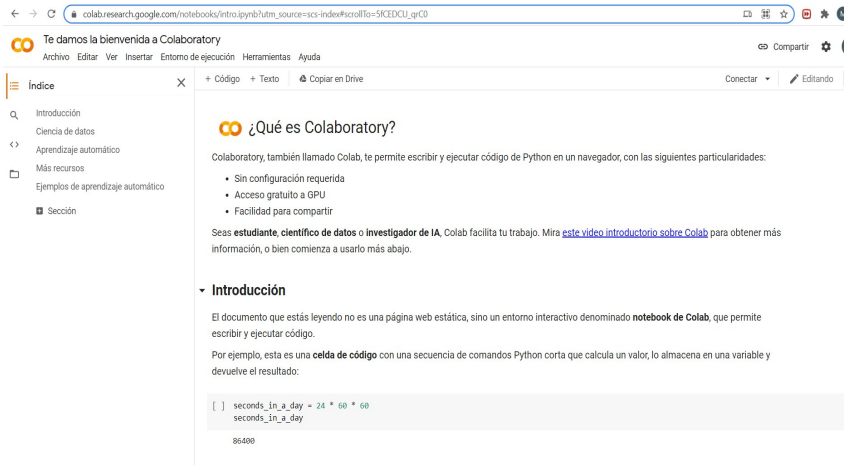
Resumen. High-pass filter es un filtro que se utiliza para detectar los bordes de una imágen donde el color homogéneo es obviado, pero las zonas de detalle se preservan para poder perfilar siluetas A lo largo de este trabajo el objetivo es comparar el funcionamiento de la aplicación del filtro tanto con CPU como con GPU. El procesamiento en GPU se hizo utilizando la biblioteca pycuda. Asimismo se utiliza para aplicar este filtro el concepto de Image Convolution y el lenguaje utilizado fue Python.

Palabras claves: Python, high-pass filter, GPU, Image Convolution, HPC, Cuda programming.

1 Introducción

1.1 Objetivos

En este paper se va a mostrar el funcionamiento de un filtro de imágenes High Pass. Para ello se confeccionó un cuaderno en Colab [1].
Colab es un servicio cloud, basado en los Notebooks de Jupyter, que permite el uso gratuito de las GPUs y TPUs de Google bajo Python 2.7 y 3.6.
Nos permite practicar y mejorar nuestros conocimientos en HPC**,** sin tener que invertir en recursos hardware o del Cloud.
Con Colab se pueden crear notebooks o importar los que ya tengamos creados, además de compartirlos y exportarlos cuando queramos. Esta fluidez a la hora de manejar la información también es aplicable a las fuentes de datos que usemos en nuestros proyectos (notebooks), de modo que podremos trabajar con información contenida en nuestro propio Google Drive, unidad de almacenamiento local, github e incluso en otros sistemas de almacenamiento cloud.



El objetivo de este filtro es detectar los bordes de una imágen donde el color homogéneo es obviado, pero las zonas de detalle se preservan para poder perfilar siluetas.
El objetivo es comparar el tiempo de el funcionamiento del filtro con una ejecución en CPU contra una ejecución en GPU utilizando la biblioteca pycuda [2].

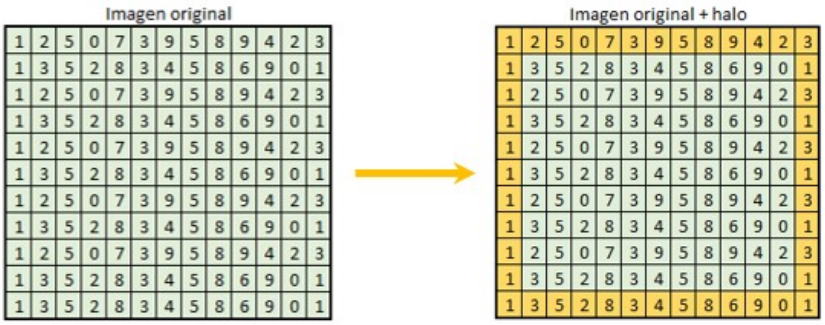
Para este filtro se aplica el concepto de Image Convolution [3] en el que por cada píxel se deben obtener una N cantidad de pixeles a su alrededor y multiplicarlos contra otra matriz de NxN para luego realizar una sumatoria de todos los valores resultantes y con este resultado actualizar el pixel incial.
En este cuaderno se va a utilizar una matriz de N=3. La que mejor resultado nos dió para el filtro fue la siguiente:

0	-1	0
-1	4	-1
0	-1	0

En la cual la celda coloreada representa al píxel actual al que se le está aplicando el filtro.
Por obvias razones, no se multiplicarán ni sumarán los píxeles que correspondan a una celda de la matriz NxN que contenga un cero.

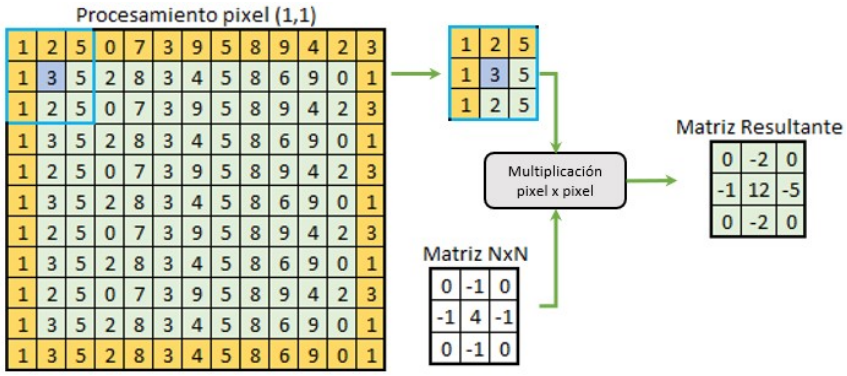
Para comenzar a aplicar este filtro, debemos tomar la imagen original y no tener en cuenta los pixeles que no puedan ser evaluados con la completitud de la matriz. Es decir, en este caso usamos una matriz de 3x3, por lo tanto, los pixeles del borde no podrán ser evaluados ya que se perdería una multiplicación de la matriz de 3x3 y generaría un resultado erróneo. Entonces, lo que debemos hacer es descartar estos pixeles del borde, generando como

una especie de “halo” de pixeles que no se evaluarán. Esto solo genera un cambio a la hora de validar que el idx esté dentro del rango de imagen, el “halo” es solo ilustrativo. Gráficamente:



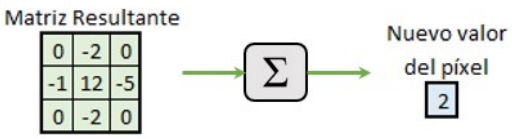
“Halo” ilustrativo. Los pixeles con amarillo no serán evaluados.

Luego, debemos evaluar cada pixel contenido dentro del “halo” (los que están en verde claro). Para ello nos paramos sobre el pixel a evaluar, lo tomamos como centro y extraemos una matriz de NxN (con N igual al N de la matriz con la cual multiplicaremos), realizamos la multiplicación pixel por pixel, y obtendremos una matriz NxN resultante:



Obtención de la matriz resultante para el pixel [1,1] de la imágen

Luego sumamos todos los valores de la matriz resultante y obtenemos un único valor que será nuestro nuevo *pixel [1,1]*. Este proceso se hará sobre cada pixel dentro del “halo”.

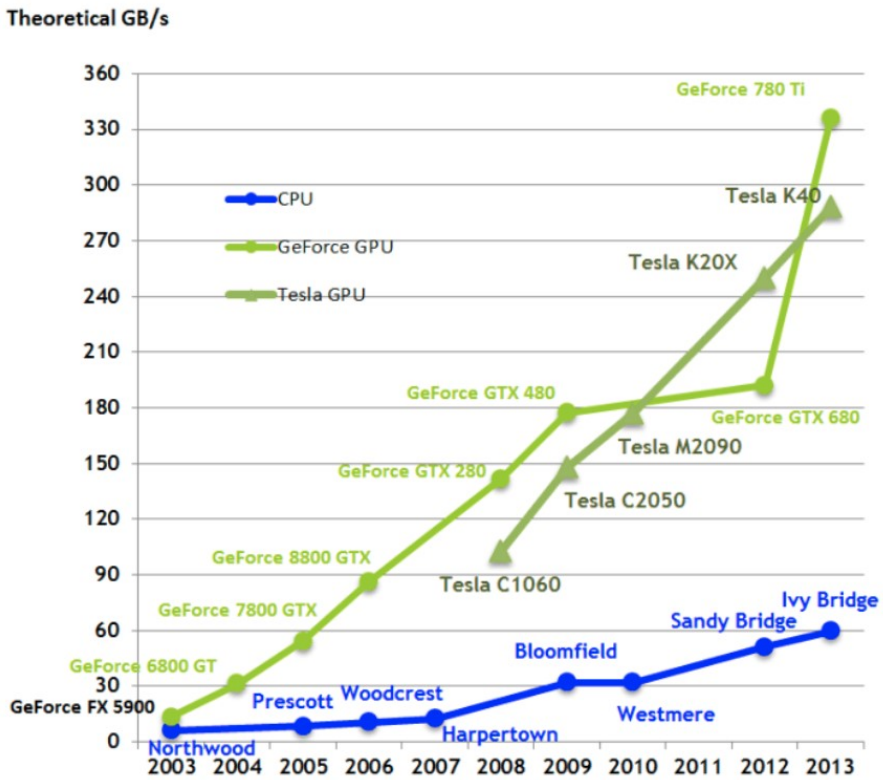


Valor resultante del pixel [1,1]

1.2 CPU vs GPU

Para este trabajo, es necesario comprender el concepto de GPU para contrastarlo contra la CPU.
Es un unidad de procesamiento gráfico con la capacidad de resolver operaciones de coma flotante. Posee procesadores de varios núcleos que ofrecen alto rendimiento.
El paralelismo es una forma de computación en la cual varios cálculos pueden realizarse simultáneamente.
Una forma sencilla de comprender la diferencia entre una GPU y una CPU es comparar la forma en que procesan las tareas. Una CPU tiene unos cuantos núcleos optimizados para el procesamiento en serie secuencial, mientras que una GPU cuenta con una arquitectura en paralelo enorme que consiste de mieles de núcleos más pequeños y eficaces, y que se diseñaron para resolver varias tareas al mismo tiempo.

En el siguiente gráfico podemos ver una comparativa de ancho de banda entre CPU y GPU, donde se aprecia la superioridad de esta última [4].



2 Desarrollo

2.1 Desarrollo

Dirección web del repositorio GitHub: https://github.com/MartinRomano-S/soa-tp2-tp3-grupo4/blob/master/HPC/Cuaderno_1_Mi%C3%A9rcoles__grupo4_2021.ipynb

2.2 Funcionamiento desde punto de vista de la programación CPU/GPU.

El funcionamiento desde el punto de vista de la programación en CPU comienza con la apertura de la imagen y su conversión a array para poder trabajarla. A continuación, se recorre la imagen a lo alto y a lo ancho con dos ciclos for, se le aplica la matriz seleccionada mencionada anteriormente, y se obtiene la imagen en formato de array.
Finalmente se muestra la imagen resultante y el tiempo que se tardó en procesarla por la CPU.

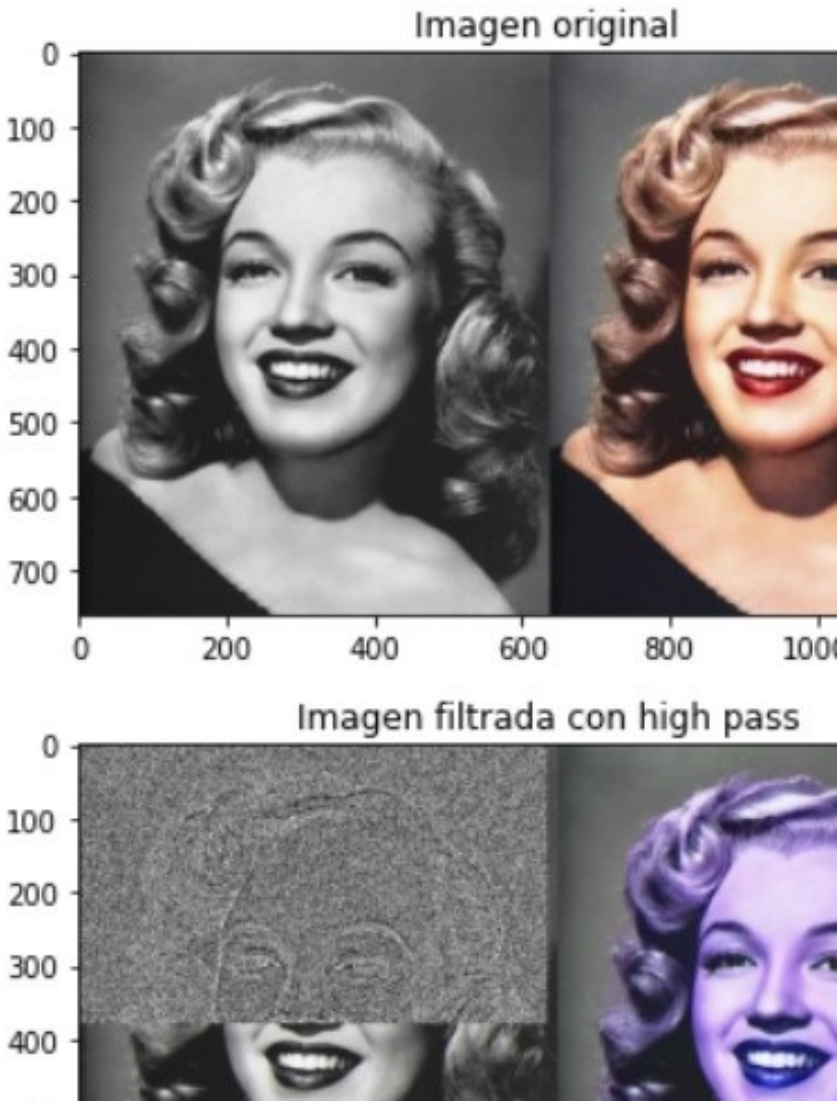
En cuanto al funcionamiento desde el punto de vista GPU, también comienza con con la apertura de la imagen y su conversión a array. Luego se deben reservar en memoria con el método mem_alloc de cuda, los vectores de la imagen. Dicha memoria se copia al GPU y se procede a recorrer con los hilos del GPU las dimensiones de la imagen, a través de una función que definimos, aplicándole la matriz mencionada para obtener los RGB de la imagen final. Se calcula el tiempo y se lo muestra por pantalla.

Resumiendo, la diferencia sustancial del algoritmo implementado en la GPU respecto al CPU es que se hace uso de hilos para recorrer cada pixel de la imagen de manera paralela, en vez de la manera secuencial implementada con un doble for en CPU.

2.3 Diversos experimentos sobre high-pass filter

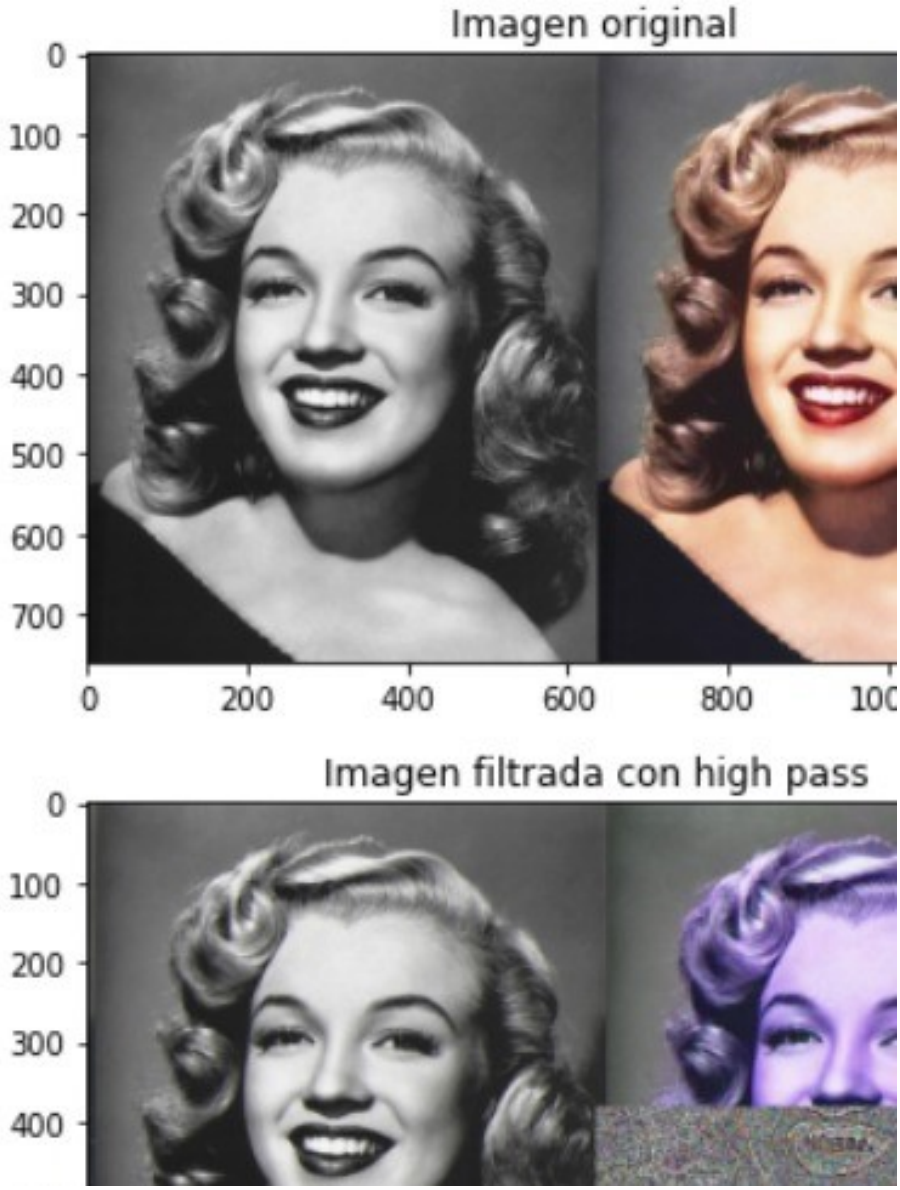
En nuestro trabajo realizamos los siguientes experimentos para comprobar como afectan a los tiempos y al procesamiento de la imagen.

Por ejemplo, el procesamiento original de la imagen bajo GPU nos dio un promedio aproximado de 70,12 ms.
Al ejecutar con la mitad de los hilos, tuvo un ligero aumento a 71,28 ms.
Mientras que al ejecutar con el doble de hilos el tiempo bajó muy poco, a 69,68 ms. Si bien es una mejora, no es considerable.
Lo mismo sucede cuando se planifica con el máximo de hilos, el tiempo baja a 69,23 ms, un tiempo menor, pero no demasiado significativo respecto al original.
Lo siguiente sucede cuando se procesa la mitad inicial de la imagen original, donde se obtiene como resultado la siguiente imagen.



Se puede apreciar que se le aplicar el filtro completo apenas a la primera mitad de la primera imagen.
Asimismo el tiempo de procesamiento nos dio similar, unos 69,77 ms.

Si por el contrario, decidimos procesar la mitad final de la imagen destino, se obtiene el siguiente resultado:



En ella se puede ver que se procesó la mitad final de la segunda imagen con un tiempo ligeramente superior de 71,47 ms.

2.4 Manual de Usuario de high-pass filter.

Disponible y adjuntado en el trabajo.

3 Conclusiones

- Como conclusiones, viendo las métricas obtenidas, vemos que el rendimiento al ejecutar el algoritmo del filtro con un entorno GPU y aprovechando la capacidad de los threads mejora considerablemente en comparación a la implementación CPU.

  Hay que tener en cuenta que, para imágenes pequeñas, es posible que el algoritmo de CPU demore menos tiempo de ejecución. Esto se debe a que la implementación de GPU requiere un tiempo prácticamente fijo para la planificación de los threads (overhead) que en imágenes grandes pasa desapercibido.

  Mientras más grande sea la imágen, mayor será la brecha de tiempos entre ambas implementaciones. Esto se debe a la grán complejidad computacional que presenta la implementación de CPU.

- Problemas surgidos en el desarrollo:
  El primer problema que se nos presentó fue comprender de que se trataba el filtro. Para ello debimos interiorizarnos con el concepto y buscar información para poder realizar las ejecuciones tanto de CPU como GPU.

  El segundo problema fue que el resultado del procesamiento de la imagen al principio distaba ampliamente de lo esperado. Luego de varias revisiones del código sin poder encontrar el error, detectamos que el mismo era ocasionado por una línea en la que estaban invertidos el ancho y el alto de la imagen. Con cambiar su orden, se llegó a resultado que queríamos.

  También nos encontramos con inconvenientes al probar la matriz que mejor se adapte al filtro de high-pass. Comenzamos a buscar y a probar y finalmente llegamos al resultado óptimo con la matriz declarada con ceros, unos y un cuatro en el centro.

  Para poder realizar este trabajo también debimos aprender nociones básicas de Python [5], lo cual si bien fue sencillo por el alcance de lo que necesitábamos hacer, tuvo una ligera curva de aprendizaje.

4 Referencias

Bibliografía

[1] Google, «https://colab.research.google.com/,» [En línea]. Available: https://colab.research.google.com/.

[2] python, «https://documen.tician.de/pycuda/,» [En línea]. Available: https://documen.tician.de/pycuda/.

[3] I. G. L. I. C. D. Ing. Jan Novak, «https://cg.ivd.kit.edu/,» [En línea]. Available: https://cg.ivd.kit.edu/downloads/assignment3_GPUC.pdf.

[4] S. Shah, «https://medium.com,» [En línea]. Available: https://medium.com/@shachishah.ce/do-we-really-need-gpu-for-deep-learning-47042c02efe2.

[5] python, «https://www.python.org/,» [En línea]. Available: https://docs.python.org/3/.