

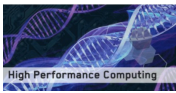


PYTHON



Manual de usuario de High-pass filter

VERSIÓN 1.0



Índice

Introducción	2
Configurar entorno	3
Armado del ambiente	4
Carga de imagen	5
Ejecución CPU	6
Ejecución GPU	7
Métricas y conclusiones	9



Introducción

El cuaderno de Colab de High-pass filter permite realizar pruebas de dicho filtro. Asimismo se puede realizar la comparativa entre una ejecución con CPU y otra con GPU.

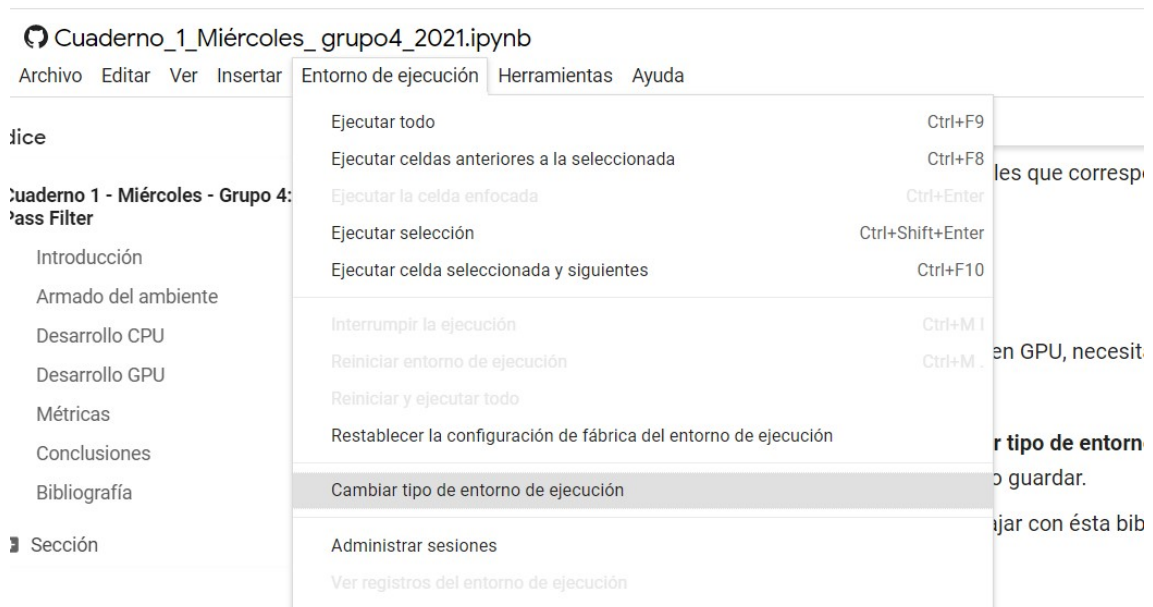
Para poder utilizarlo es necesario ejecutar el cuaderno en Colab. La dirección de github del mismo es:

https://github.com/MartinRomano-S/soa-tp2-tp3-grupo4/blob/master/HPC/Cuaderno_1_Mi%C3%A9rcoles_grupo4_2021.ipynb

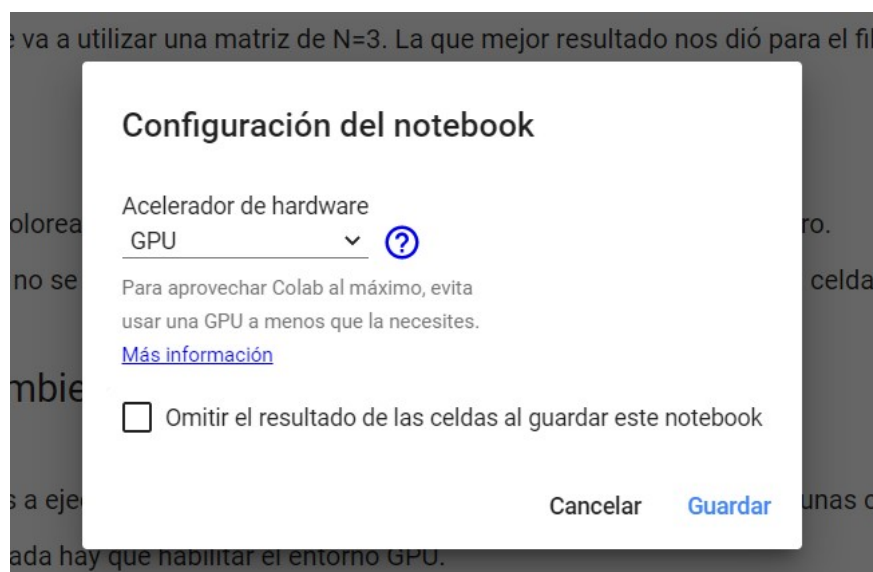


Configurar entorno

Al acceder, lo primero que se debe hacer es configurar el entorno, cambiando el entorno de ejecución a Solo GPU. Primero, clicar “Entorno de ejecución”.



A continuación se debe cambiar a “Solo GPU” y guardar la configuración.



Armado del ambiente

A continuación, se procede a armar el ambiente. Para ello se debe ejecutar la siguiente línea:

▼ Armado del ambiente

Debido a que vamos a ejecutar código tanto en CPU como en GPU, necesitamos tener unas consideraciones previas.

1. Primero que nada hay que habilitar el entorno GPU.

Para esto hay que ir a **Entorno de ejecución > Cambiar tipo de entorno de ejecución**.

Seleccionar GPU como *acelerador de hardware* y luego guardar.

2. Luego debemos instalar **pycuda** ya que vamos a trabajar con ésta biblioteca.

Para ello debemos ejecutar la siguiente instrucción:

```
!pip install pycuda
```

```
Collecting pycuda  
  Downloading https://files.pythonhosted.org/packages/5a/56/4682a5118a234d15aa1c8768a528aac4858c7b04d2674e18d5  
    | 1.7MB 29.3MB/s  
Installing build dependencies ... done  
Getting requirements to build wheel ... done
```

Carga de imagen

Antes de ejecutar el filtro, primero se debe cargar la imagen, para esto se debe presionar play en la segunda parte del armado del ambiente.

3. Debemos ingresar una imagen a la cual aplicarle el filtro y, además, definimos una función lambda para mostrar las métricas de tiempos.

```
url_imagen = "https://hotpot.ai/images/site/ai/remaster/teaser.jpg" :
if not (type(url_imagen) is str) or (url_imagen == ""):
    raise TypeError("URL inválida")

!wget {url_imagen} -O imagen.jpg

# Definición de función que transforma el tiempo en milisegundos
tiempo_en_ms = lambda dt:(dt.days * 24 * 60 * 60 + dt.seconds) * 1000

--2021-07-03 18:42:05-- https://hotpot.ai/images/site/ai/remaster/teaser.jpg
Resolving hotpot.ai (hotpot.ai)... 104.26.6.94, 104.26.7.94, 172.67.72.40, ...
Connecting to hotpot.ai (hotpot.ai)|104.26.6.94|:443... connected.
HTTP request sent, awaiting response... 200 OK
length: 655360 (640K) [image/jpeg]
```

A continuación, se verá el resultado de la ejecución.

Ejecución CPU

En la opción Desarrollo CPU, al clicar en el botón de “play”, se ejecutará el filtro de high-pass bajo CPU.

▼ Desarrollo CPU

```
[ ] %matplotlib inline
    from datetime import datetime

    # Tiempo total de ejecución
    tiempo_cpu_total = datetime.now()

    import matplotlib.pyplot as plt
    import numpy
    from PIL import Image

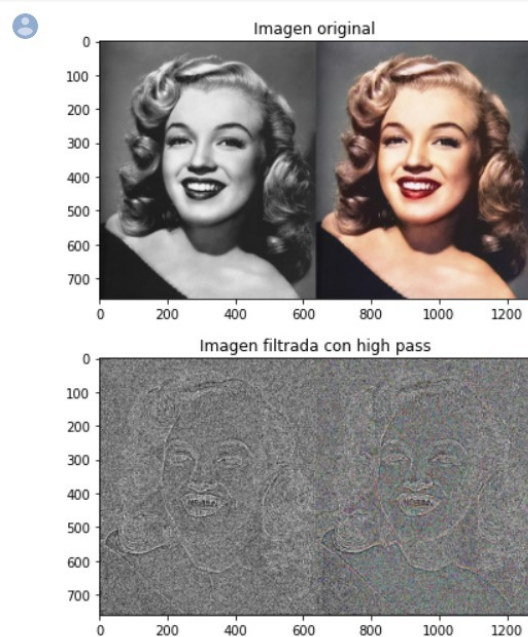
    # Gap que dejamos para aplicar el filtro
    BORDER = 1

    img_nombre = 'imagen.jpg'
    image = Image.open( img_nombre )

    # Obtenemos las dimensiones de la imagen
    img_ancho, img_alto = image.size

    # Convierto la imagen comprimida en JPEG/PNG a array
    img_O_cpu = numpy.asarray(image)
    img_R_cpu = numpy.empty_like( img_O_cpu)
```

Al finalizar la ejecución veremos el resultado final en modalidad CPU.




Ejecución GPU

En la opción Desarrollo GPU, se procederá a ejecutar el filtro de high-pass bajo GPU.

+ Código + Texto

▼ Desarrollo GPU

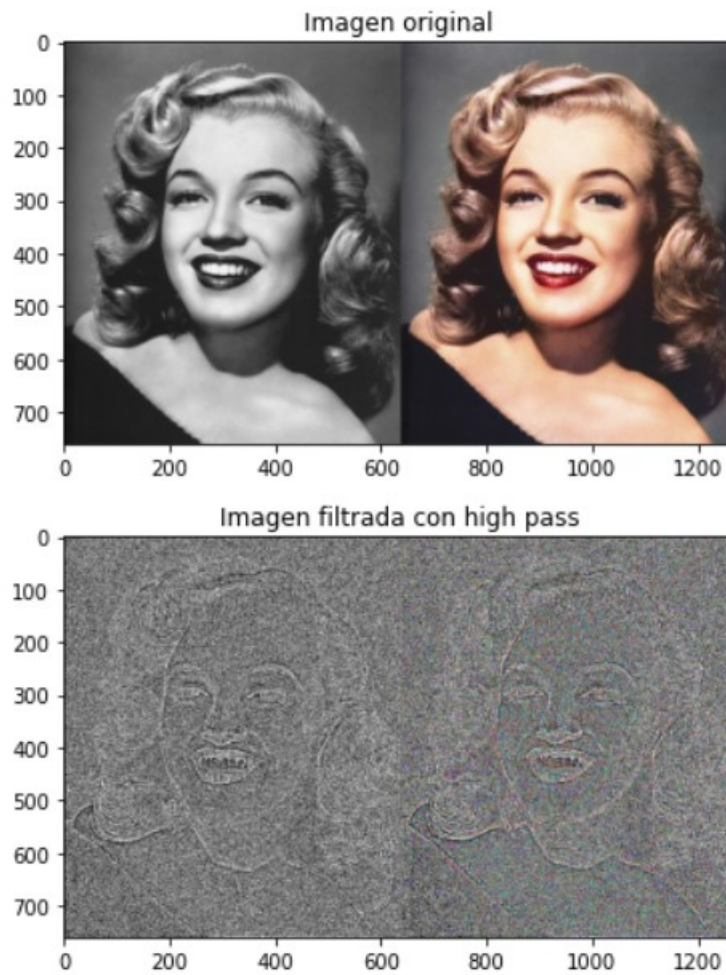
```
 %matplotlib inline
from datetime import datetime

# Tiempo en CPU + GPU
tiempo_gpu_total = datetime.now()

import matplotlib.pyplot as plt
import numpy
from PIL import Image
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule

img nombre = 'imagen.jpg'
```

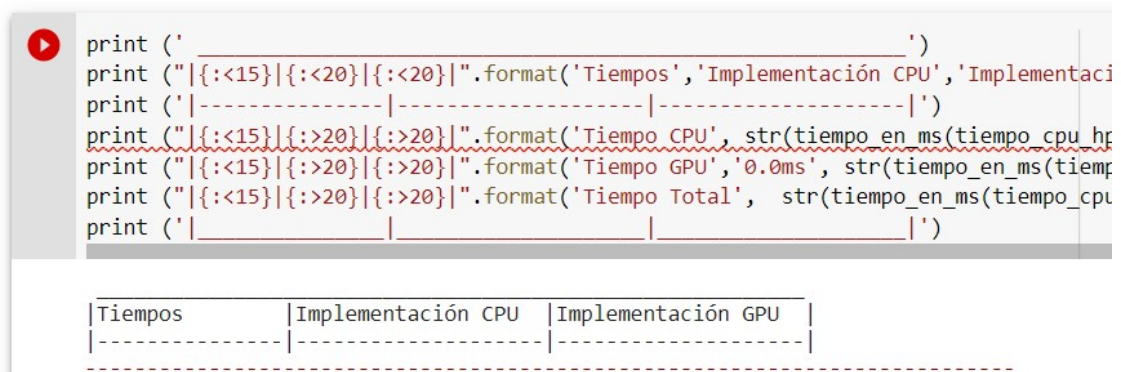

Al terminar, se verá el resultado de la imagen.



Métricas y conclusiones

Finalmente, se pueden ver las métricas a presionar el botón de “play” en dicha sección para poder comparar los tiempos tanto de CPU como GPU.

▼ Métricas



```

print ('_____')
print ("|{:<15}|{:<20}|{:<20}|".format('Tiempos', 'Implementación CPU', 'Implementación GPU'))
print ('|-----|-----|-----|')
print ("|{:<15}|{:>20}|{:>20}|".format('Tiempo CPU', str(tiempo_en_ms(tiempo_cpu_h)), str(tiempo_en_ms(tiempo_gpu_h))))
print ("|{:<15}|{:>20}|{:>20}|".format('Tiempo GPU', '0.0ms', str(tiempo_en_ms(tiempo_gpu_h))))
print ("|{:<15}|{:>20}|{:>20}|".format('Tiempo Total', str(tiempo_en_ms(tiempo_cpu_h + tiempo_gpu_h))))
print ('|_____|_____|_____|')

```

Tiempos	Implementación CPU	Implementación GPU
-----	-----	-----

Al finalizar se visualizarán las conclusiones del trabajo.

▼ 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 imagen, mayor será la brecha de tiempos entre ambas implementaciones. Esto se debe a la gran complejidad computacional que presenta la implementación de CPU.

▼ Bibliografía

[1] PyCUDA Documentation: [doc](#)

[2] GPU Computing: Image Convolution, Dipl.-Ing. Jan Novak, Dipl.-Inf. Gabor Liktor, Prof. Dr.-Ing. Carsten Dachsbacher: [PDF](#)