Filtro de realce de color R, G ó B

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

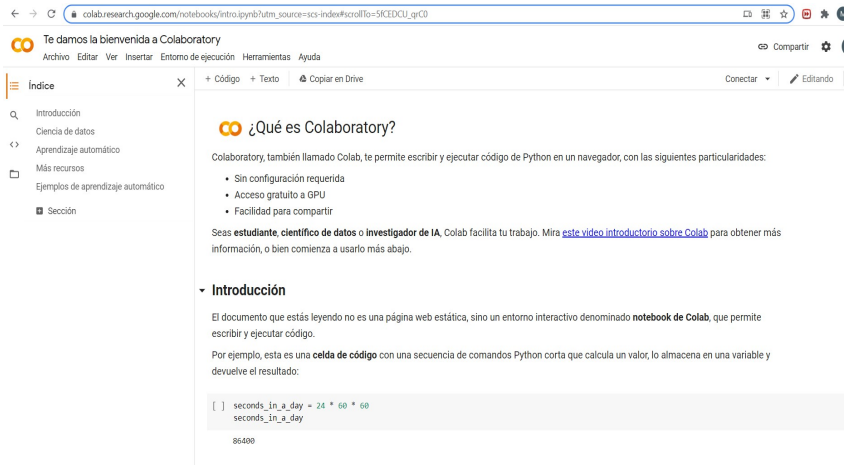
Resumen. El filtro de realce R, G ó B, es un filtro en el que se intenta mostrar el potencial que tiene el procesamiento de gráficos dela GPU utilizando Colab con el lenguaje de programación Python y una biblioteca particular que nos simplificó el trabajo, llamada OpenCL. El objetivo de dicho filtro es realzar el color seleccionado, analizando el valor de la componente R, G ó B y dejando escala de grises para el resto de los colores dando paso a un singular filtro de imágenes.

Palabras claves: Python, GPU, RGB, HPC, Cuda programming, OpenCL.

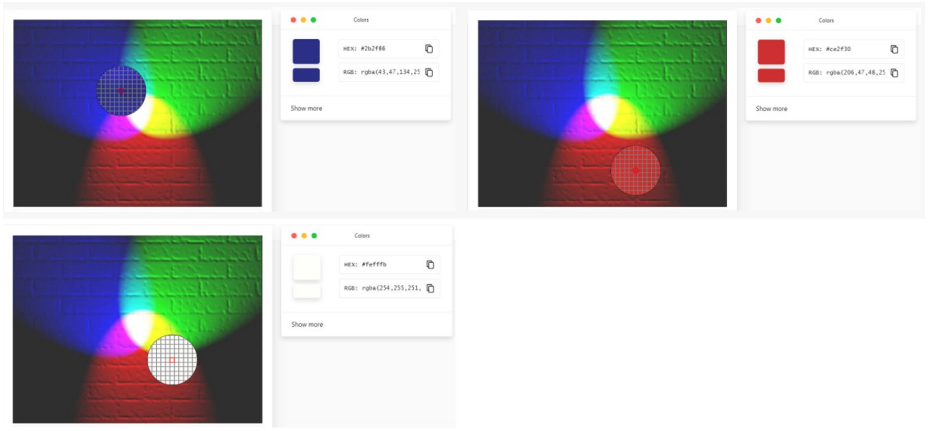
1 Introducción

1.1 Objetivos

En este paper se va a mostrar el funcionamiento de un filtro de relace de color, ya se R, G ó B. 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.



En este trabajo se intentará demostrar el potencial de procesamiento que nos provee la GPU en comparación a la CPU. Por tal motivo, desarrollamos un filtro propio que consiste en detectar los píxeles de color rojo, verde o azul y realzarlos (según elección del usario) mientras que el resto de los píxeles son convertidos a escala de grises.
El algoritmo funciona de la siguiente manera: Supongamos que el usuario eligió el color rojo. El algoritmo recorrerá todos los píxeles de la imagen y por cada pixel analizará si la componente "R" es mayor a 100 y si las componentes "G" y "B" son menores a 120. Éstos datos surgen de una investigación propia en la que se hicieron muestreos de varias imágenes y se analizaron las componentes R, G y B para obtener los valores de umbral.



En las imágenes se pueden apreciar los valores RGB de los pixels seleccionados.

Siguiendo el ejemplo, si el pixel analizado corresponde a la gama del color rojo, se procede a realzar el color. Ésto se hace aumentando la componente R en un 25% y disminuyendo las componentes G y B en un 30%. Si el pixel analizado no corresponde a la gama del color rojo, entonces se transforma a escala de grises de la siguiente manera: Se multiplica la componente roja por 0.3, la componente verde por 0.59 y la componente azul por 0.11, luego se suman esos valores y el resultado obtenido será el valor de las commponentes R, G y B del píxel en escala de grises.
.

1.2 GPGPU

GPGPU, General-Purpose computation on Graphics Processing Units, es el uso de un chip GPU, que usualmente procesa sólo gráficos por computadora, para realizar operaciones generales de computación (tarea usualmente exclusiva del CPU). Se trata de aprovechar la capacidad del GPU más allá del simple procesamiento de gráficos.

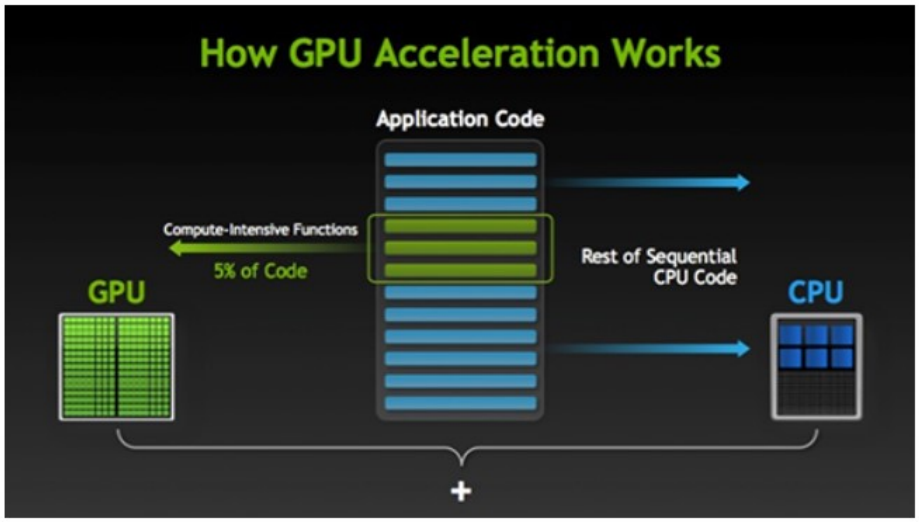
Los GPU son diseñados para procesar cálculos y funciones gráficos, pero potencialmente pueden realizar otras operaciones también. GPGPU maximiza la eficiencia de procesamiento descargando algunas operaciones desde la CPU hacia la GPU. En este caso, en lugar de estar sin hacer nada cuando no procesa gráficos, la GPU estará disponible para otras tareas.

Los sistemas operativos modernos permiten a los programas acceder al GPU junto con la CPU, permitiendo mejorar su rendimiento. Además la GPU está optimizado para el procesamiento en el cálculo de vectores, por lo que pueden procesar ciertas instrucciones más rápidamente que la CPU.

GPGPU es un tipo de procesamiento paralelo, en el que las operaciones se procesan conjuntamente entre la CPU y la GPU.

Cuando la GPU finaliza un cálculo, el resultado puede ser almacenado en una memoria buffer (apenas milisegundos) y luego ser pasado a la CPU.

El popular OpenCL permite programar software para que tanto la CPU como la GPU compartan procesamiento. Otros similares son CUDA, una API de la empresa NVIDIA y APP, un SDK que provee la empresa AMD.



La web de nvidia ofrece una serie de conceptos y herramientas referidas a la programación en GPU [2].

2 Desarrollo

2.1 Desarrollo

Dirección web del repositorio GitHub: https://github.com/MartinRomano-S/soa-tp2-tp3-grupo4/blob/master/HPC/Cuaderno_2_Mi%C3%A9rcoles__grupo4_2021.ipynb

2.2 Conceptos aplicados.

Para desarrollar este trabajo, nos basamos en conceptos fundamentales de HPC, el cual nos brinda la capacidad de procesar datos y realizar cálculos complejos a velocidades muy altas.
Lo que se hizo fue aprovechar las ventajas de GPU para utilizar hilos simultáneos para recorrer la imagen pixel por pixel, analizarlo y en base a eso realzar el color seleccionado previamente.

Originalmente para manejar esta aplicación se intentó utilizar la tecnología CUDA, pero no dio los resultados esperados, con lo cual se procedió a usar OpenCL.

OpenCL [3] es una de las alternativas libres a la anterior mencionada tecnología CUDA de Nvidia, que intenta aprovechar la potencia de los procesadores gráficos para realizar operaciones intensas repartidas entre el procesador del equipo (CPU) y la GPU de cualquier tarjeta gráfica compatible. Al contrario de CUDA, OpenCL fue creado originalmente por Apple quien luego la propuso al Grupo Khronos para convertirlo en un estandar abierto y libre que no dependa de un hardware de un determinado fabricante (CUDA sólo está disponible en gráficas NVidia).

OpenCL Reference Pages

OpenCL (Open Computing Language) is an open royalty-free standard for general purpose parallel programming across CPUs, GPUs and other processors, giving software developers portable and efficient access to the power of these heterogeneous processing platforms.

OpenCL supports a wide range of applications, ranging from embedded and consumer software to HPC solutions, through a low-level, high-performance, portable abstraction. By creating an efficient, close-to-the-metal programming interface, OpenCL will form the foundation layer of a parallel computing ecosystem of platform-independent tools, middleware and applications.

OpenCL consists of an API for coordinating parallel computation across heterogeneous processors; and a cross-platform programming language with a well-specified computation environment. The OpenCL standard:

- Supports both data- and task-based parallel programming models
- Utilizes a subset of ISO C99 with extensions for parallelism
- Defines consistent numerical requirements based on IEEE 754
- Defines a configuration profile for handheld and embedded devices
- Efficiently interoperates with OpenGL, OpenGL ES, and other graphics APIs

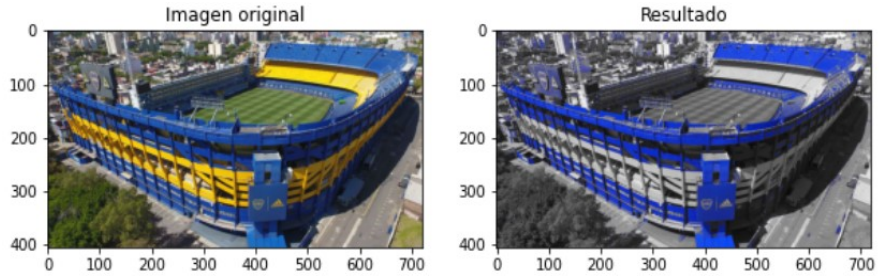
The specification is divided into a core specification that any OpenCL compliant implementation must support; a handheld/embedded profile which relaxes the OpenCL compliance requirements for handheld and embedded devices; and a set of optional extensions that are likely to move into the core specification in later revisions of the OpenCL specification.

Para poder comparar el procesamiento de las imágenes en CPU y GPU, desarrollamos funcionalidades con el uso de ámbas tecnologías.
La primera consta en recorrer con dos ciclos for la imagen a lo alto y a lo ancho, ver sus valores RGB y en base a eso aplicar el filtro.
En el segundo caso, usando GPU, se recorre cada pixel con un hilo diferente, se lee su RGB utilizando OpenCL y en base al color seleccionado, se usará una de las 3 funciones definidas, las cuales consisten en detectar si la componente R, G o B del pixel es superior a 100 y las otras inferiores a 120. Si esa condición se cumple, significa que ese píxel es de tono rojo, azul o verde (según corresponda).
Si pasa el filtro, realzamos la componente un 25% y reducimos el resto de las componentes del pixel un 30%.
Si no pasa el filtro, convertimos el pixel a escala de grises.

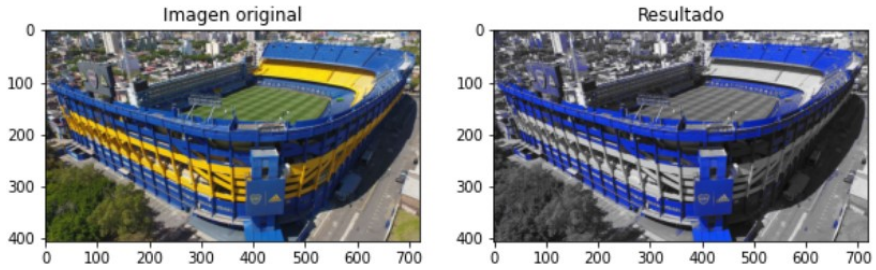
2.3 Pruebas realizadas para el filtro de realce de color R, G ó B.

Al realizar los tests podemos ver como el resultado de la imagen es idéntico, pero también se puede ver claramente que el tiempo de procesamiento GPU es significativamente inferior al de CPU.

Resultado CPU



Resultado GPU



Resultados visualmente idénticos, pero con tiempos con enormes diferencias.

Tiempos	Implementación CPU	Implementación GPU
Tiempo CPU	3617.422ms	222.31ms
Tiempo GPU	0.0ms	35.969ms
Tiempo Total	3674.834ms	258.279ms

2.4 Manual de Usuario de Filtro de realce R, G ó B.

Disponible y adjuntado en el trabajo.

3 Conclusiones

- A partir de los resultados obtenidos en las métricas, se ve que el tiempo de ejecución total del programa utilizando la GPU para la imagen propuesta es de tan sólo el 7.02% del tiempo de ejecución total utilizando sólo la CPU. Es decir, aproximadamente 14 veces menos. Ésto nos da una clara demostración de la mejora en tiempo que nos proveen los threads.
  Similar a lo concluído en el ejercicio 1, si se trata de imágenes pequeñas, por ejemplo una imagen de 256x256 píxeles, la diferencia entre ambos algoritmos será despreciable, ya que el overhead de la planificación de threads termina equiparando los tiempos totales. Para la imagen propuesta (Estrellas) se obtuvieron los siguientes tiempos: CPU: 270.92ms, GPU: 238.978ms
  Pero también es importante destacar que cuanto mayor es la imagen, mayor también será la brecha entre ambos. Para una imagen de 1851x1041 píxeles, la diferencia es casi 100 veces menor para el algoritmo con GPU. Para la imagen propuesta (Salzburgo) se obtuvieron los siguientes tiempos: CPU: 23335.426ms, GPU: 365.

- Problemas surgidos en el desarrollo:
  El primer problema presentado fue pensar en como poner de manifiesto en un trabajo la diferencia sustancial que existe entre el procesamiento de imágenes entre CPU y GPU. Después de un tiempo de debate nos fijamos de hacer un filtro novedoso que muestre el poderío de la programación con paralelismo a nivel de datos.

  Otro problema era comprender un poco más el lenguaje Python [4] con sus sintaxis propias. También en más de una ocasión el incorrecto uso de la tabulación nos provocó errores de compilación.

  Al principio queríamos utilizar CUDA para leer los valores RGB, los cuales varían entre 0 y 255. Sin embargo por alguna extraña razón nos daba valores negativos, con lo cual debimos buscar otra alternativa. Para canalizar dicha situación finalmente utilizamos OpenCL.

4 Referencias

Bibliografía

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

[2] | OpenCL, «https://www.khronos.org/,» [En línea]. Available: https://www.khronos.org/registry/OpenCL/sdk/1.2/docs/man/xhtml/.

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

[4] | Nvidia, «https://www.nvidia.com/,» [En línea]. Available: https://www.nvidia.com/es-la/drivers/what-is-gpu-computing/.