# Programación Paralela

Teórica 02 - Introducción a CUDA

Segundo Semestre, 2025

---

## Introducción

Para los ejercicios vamos a hacer CUDA C que nos permite escribir programas paralelos escalables en sistemas paralelos donde conviven tanto CPUs como GPUs. **En este tipo de sistemas donde hay porciones de código que pueden ejecutarse en paralelo, pero que están gobernadas por un código secuencial que corre en la CPU, los llamaremos *heterogéneos*.** CUDA extiende el lenguaje de programación C con una sintaxis mínima que permite realizar tanto sistemas conteniendo CPUs o GPUs.

Cuando en el software moderno las aplicaciones se ejecutan *lento*, el problema usualmente suele ser que hay demasiados datos para ser procesados. Por ejemplo casos como el procesamiento de imágenes o videos, simulación de dinámica de fluidos, manejo de sistemas complejos como (líneas aéreas), o incluso cosas mucho más sencillas como convertir una imagen de pixels a escala de grises, se pueden fraccionar en tareas más pequeñas que pueden ser ejecutadas de manera independiente.

> **Paralelismo de tareas vs. paralelismo de datos**
>
> El paralelismo de tareas se refiere a la ejecución de múltiples tareas (no necesariamente las mismas) al mismo tiempo. Por otro lado, el paralelismo de datos se refiere a la ejecución de la misma tarea en múltiples datos al mismo tiempo.

## Convertir una Imagen a Escala de Grises

El procesamiento de imágenes es un clásico ejemplo de computación paralela y lo utilizaremos mucho durante la materia para ejemplificar conceptos. En este caso vamos a ver cómo convertir una imagen a escala de grises.

Pero antes de comenzar tenemos que saber cómo se representa una imagen en la computadora. Esencialmente una imagen se puede representar como una matriz de tuplas $(R, G, B)$ donde $R$, $G$ y $B$ son los valores de los colores rojo, verde y azul respectivamente. Cada uno de estos valores puede ir desde 0 a 255 y representan la intensidad de cada color en un pixel. Esa combinación se encuentra dentro del triángulo AdobeRGB (ver Figura 1).

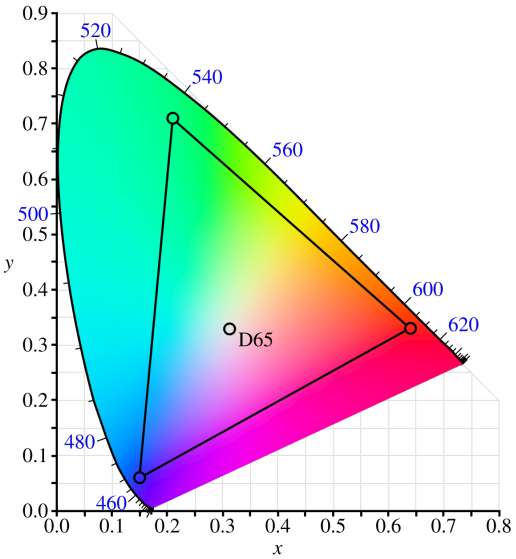


Figure 1: Triángulo AdobeRGB

Estos valores $R$, $G$ y $B$ representan los *canales* y para convertir estos canales a escala de grises se debe

utilizar una fórmula, ya que hay que decidir cuál va a ser la intensidad del pixel en escala de grises. Esto se hace combinando los valores de los canales de color de alguna manera. Lógicamente hay muchas formas de convertir estos canales a escala de grises, por ejemplo podríamos tomar sólo el valor del canal $G$, pero la imagen sólo se vería representada en una escala de grises para el canal verde, lo cual no sería muy verídico. Sin embargo si lo pensamos no es trivial cuál es la fórmula para traducir una imagen de color a escala de grises.

**Fórmulas para convertir a escala de grises**

- **Promedio**: $I = \frac{R+G+B}{3}$
- **Luminosidad**: $I = 0.21R + 0.72G + 0.07B$
- **Desaturación**: $I = \frac{max(R,G,B)+min(R,G,B)}{2}$

En este caso usaremos la fórmula de *luminosidad* para convertir la imagen a escala de grises que es un promedio pesado de los canales de color que representa la percepción humana de la luminosidad, pero como dijimos no es la única forma y cambiar esta fórmula puede dar diferentes resultados que podrían considerarse como "filtros de imagen" (¡podés probarlo luego!).

## Estructura de un programa en CUDA C

La estructura de un programa en CUDA C es similar a un programa en C, lo que refleja su naturaleza heterogénea donde existe un *host* (CPU) y uno o más *devices* (GPU) en la computadora. El código fuente CUDA tiene una mezcla de ambos códigos, uno que se ejecuta en el host y otro en el device. Por defecto, todo el código se ejecuta en el host. Las funciones declaradas para ser corridas en los *devices* son aquellas que exhibirán una gran cantidad de paralelismo de datos.

El código con estas extensiones de CUDA tiene que ser compilado con el compilador de NVIDIA, `nvcc`, que es un wrapper para el compilador de C, `gcc`. El compilador de NVIDIA se encarga de separar el código que se ejecuta en el host del que se ejecuta en el device y de compilarlo con el compilador de C. El código identificado con las *keywords* (palabras reservadas) de CUDA para las funciones paralelas se denominan *kernels*. Estos kernels son funciones que están asociadas a estructuras de datos y que van a ser ejecutadas en paralelo por GPUs. En las situaciones donde no haya una GPU disponible, el código de todas formas se ejecutará en una CPU (uno podría incluso ejecuar el kernel en CPU utilizando herramientas como MCUDA) [1].

## References

[1] H. Sutter and J. Larus, "Software and the concurrency revolution: Leveraging the full power of multicore processors demands new tools and new thinking from the software industry.," *Queue*, vol. 3, no. 7, pp. 54–62, Sep. 2005, ISSN: 1542-7730. DOI: 10.1145/1095408.1095421. [Online]. Available: https://doi.org/10.1145/1095408.1095421.