



**fcfm**

**ESCUELA DE  
INGENIERÍA Y CIENCIAS**  
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS

## GPGPU - Laboratorio 2 - Memoria constante

Maxime MARIA

Laboratorio XLIM, UMR CNRS 7252 - Universidad de Limoges, Francia

Marzo 2024

### 1 Convolución de imágenes

En este laboratorio, vais a utilizar una nueva zona de memoria en vuestra GPU : la memoria constante. El objetivo es aplicar un núcleo de convolución a una imagen<sup>1</sup>.

#### Zona de memoria constante

Como su nombre indica, esta zona de memoria se utiliza para almacenar datos que no se modificarán durante la ejecución del *kernel*. En otras palabras, sólo se puede acceder a los datos cargados en la memoria constante en modo de lectura. Hay 64 KB de memoria constante optimizada para el acceso de lectura. El uso de esta área de memoria puede aumentar el rendimiento de un programa al reducir el ancho de banda necesario para ejecutarlo.

La memoria constante se declara utilizando la palabra clave `__constant__`. Tenga en cuenta que esta área de memoria se asigna estáticamente. Por lo tanto, no hay necesidad de preocuparse por su asignación (`cudaMalloc`) o su liberación (`cudaFree`). El principal cambio está en la inicialización de los datos. En lugar de utilizar `cudaMemcpy` como para la memoria global, se debe utilizar `cudaMemcpyToSymbol`.

#### Convolución de imágenes

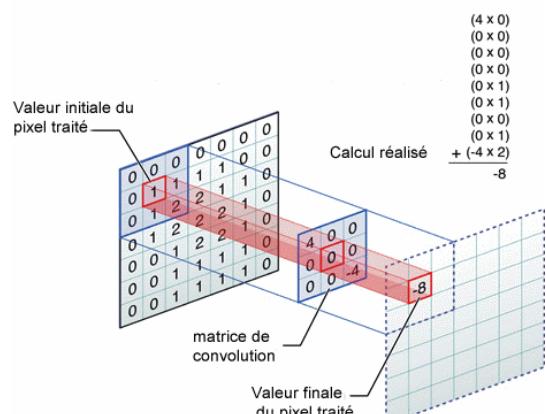
En el tratamiento de imágenes, la convolución consiste en aplicar un filtro a una imagen para resaltar efectos o recuperar información : desenfocar una imagen, detectar contornos, *etc.*

Cada filtro está asociado a una matriz  $m$  (núcleo de convolución), de tamaño  $w \times h$ , que se aplica a cada píxel de la imagen del siguiente modo :

$$f_r[x, y] = \sum_{j=0}^h \sum_{i=0}^w m[i, j] \cdot f[x + i - \frac{w}{2}, y + j - \frac{h}{2}]$$

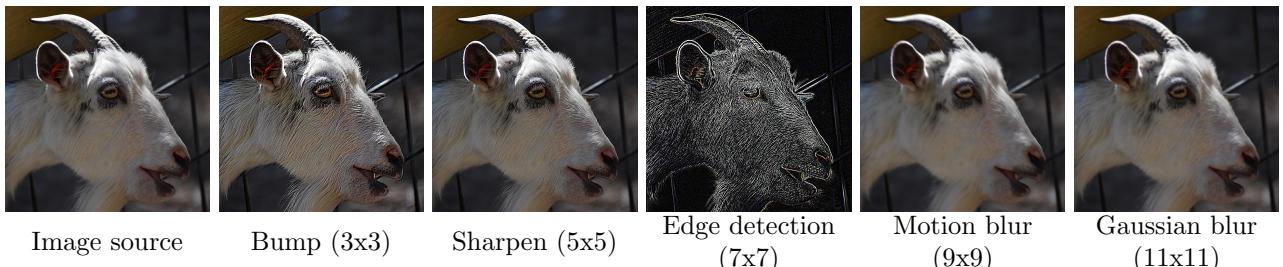
où :

- $f[x, y]$  corresponde al valor del píxel inicial (entre 0 y 255) en las coordenadas  $x$  y  $y$ ;
- $f_{res}[x, y]$  corresponde al valor del píxel del resultado (entre 0 y 255) en las coordenadas  $x$  y  $y$ ;
- $m[i, j]$  corresponde al valor de la matriz de convolución en las coordenadas  $i$  y  $j$ .



1. [https://es.wikipedia.org/wiki/Núcleo\\_\(procesamiento\\_digital\\_de\\_imágenes\)](https://es.wikipedia.org/wiki/Núcleo_(procesamiento_digital_de_imágenes))

El archivo `convolutionKernels.hpp` disponible en la plataforma contiene cinco núcleos de convolución utilizados para obtener las imágenes que se muestran a continuación.



En este laboratorio, vais a paralelizar este procesamiento en una GPU. Utilizaréis las clases de los ejercicios anteriores ([Image](#), [ChronoCPU](#), [ChronoGPU](#)). Para cada ejercicio, tendréis que analizar el rendimiento de vuestros algoritmos para varios tamaños de matriz de convolución y varias imágenes.

**NB :** En el borde de la imagen, los núcleos de convolución accederán a píxeles situados fuera de los límites de la imagen. Para superar este problema, vamos a utilizar un método por extensión : el píxel más cercano al borde de la imagen es « extendido » para proporcionar los valores necesarios para la convolución (*cf.* Figura 1). En otras palabras, conservamos el valor del píxel del borde.

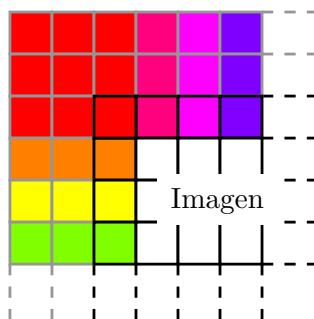


FIGURE 1 – Procesamiento de píxeles en el borde de la imagen.

## 1.1 Paralelización

A partir de la versión secuencial que figura en los archivos `reference.hpp` y `reference.cpp`, implementa el kernel CUDA ([kernelConvolution](#)) para aplicar el núcleo de convolución a cada píxel de la imagen. Asigne, libere y transfiera memoria `host ⇔ device` en la función [convolutionGPU](#).

## 1.2 Uso de la memoria constante

Ahora pasaremos el de convolución en memoria constante.

1. Declare la zona de memoria constante para el núcleo de convolución y transfírelo a la GPU.
2. Adapte el *kernel* para que utilice la memoria constante (conserva la versión anterior para comparar).