UF I5AISE51
Dimensionnement et évaluation des architectures
Introduction à la programmation massivement parallèle

—

Partie 3
Kernel à plusieurs dimensions

—

P.-E. Hladik

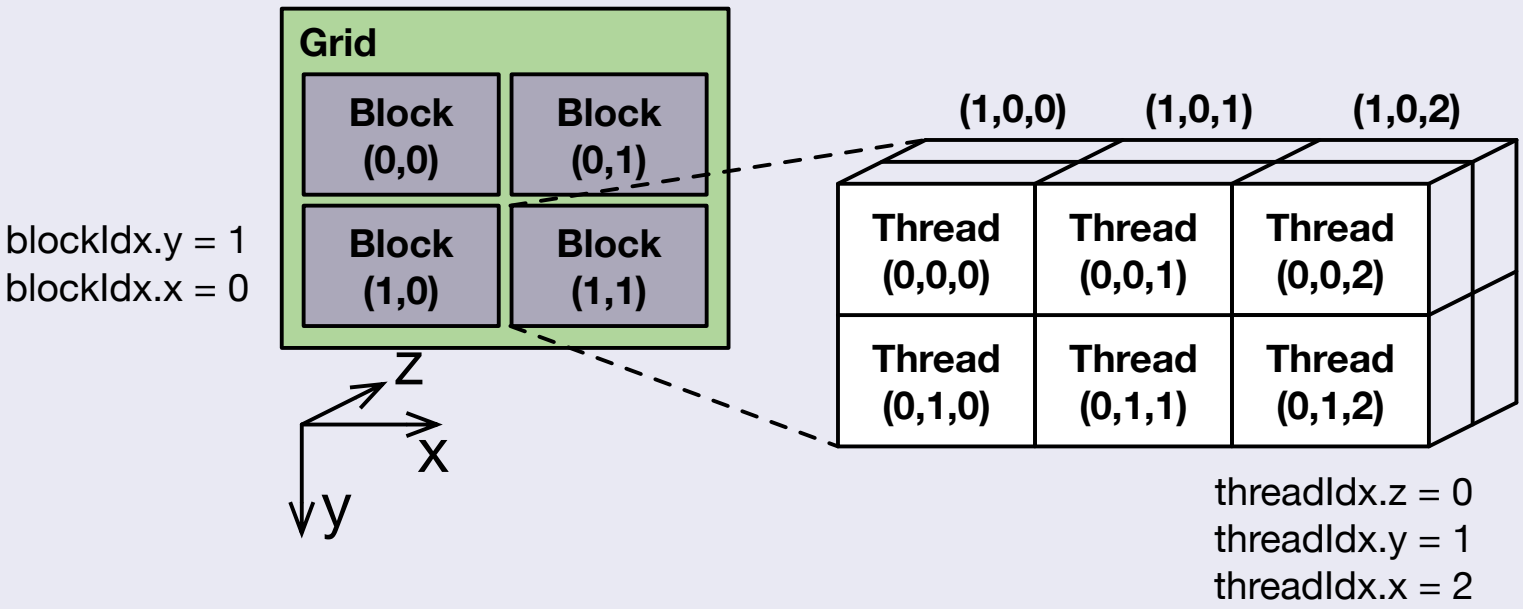
INSA Toulouse

4 janvier 2021

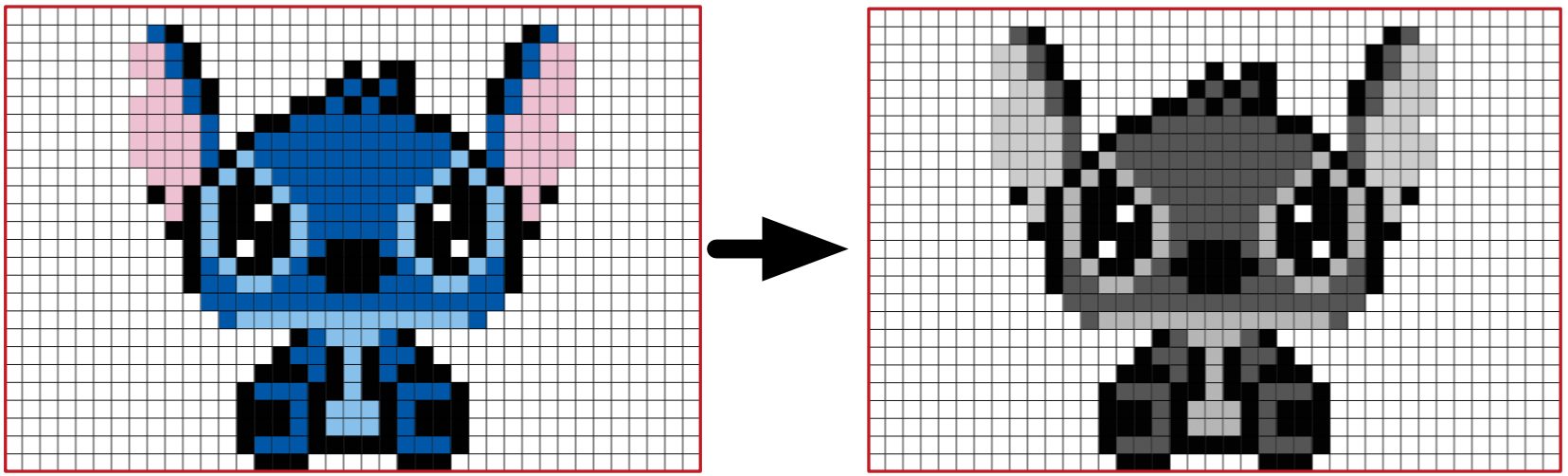
Plan

blockIdx et threadIdx

- les index de block et de thread sont des variables à trois dimensions
- cela simplifie les adresses mémoires quand on travaille avec des données multidimentionelles (image, volumes, etc.)



Exemple : niveau de gris pour une image en 2D



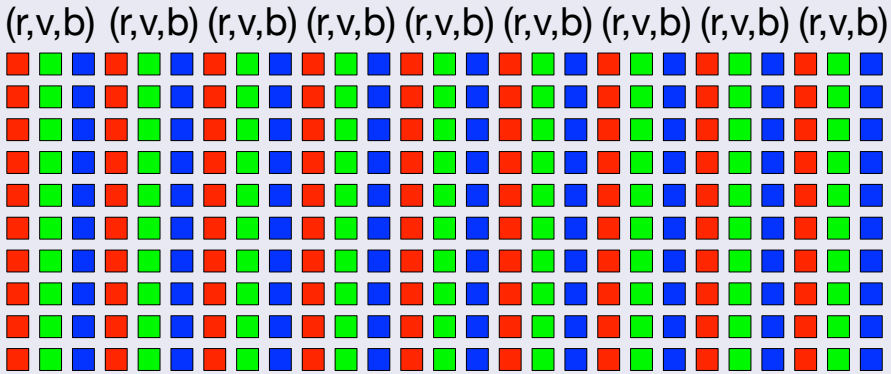
pictures : 39x26 pixels

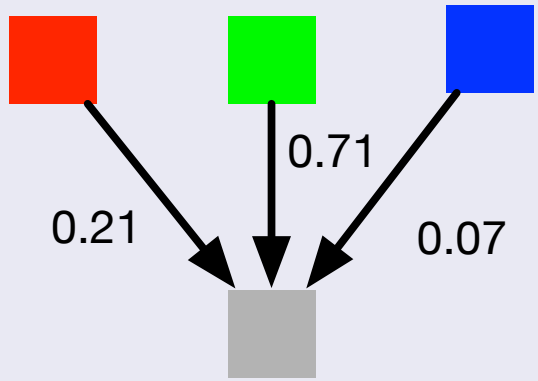
Représentation RVB d'une image et niveau de gris

- Pour chaque pixel (r, v, b) en position (i, j) :

$$grayPixel[i, j] = 0.21 \times r + 0.71 \times v + 0.07 \times b$$

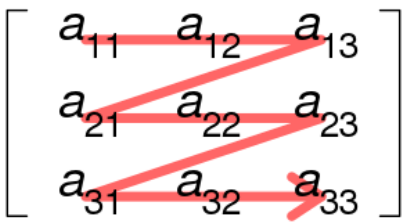
- C'est l'équivalent du produit scalaire $(r, v, b) \cdot (0.21, 0.71, 0.07)$ avec des constantes spécifiques à l'espace RVB



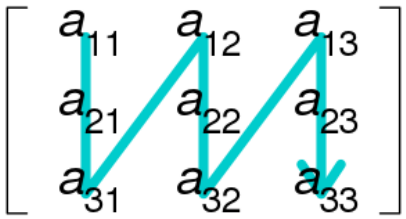


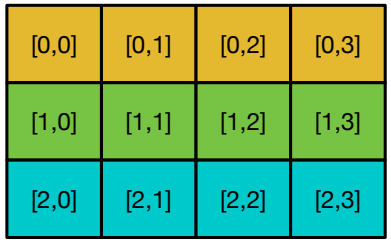
Row-Major Order en C/C++

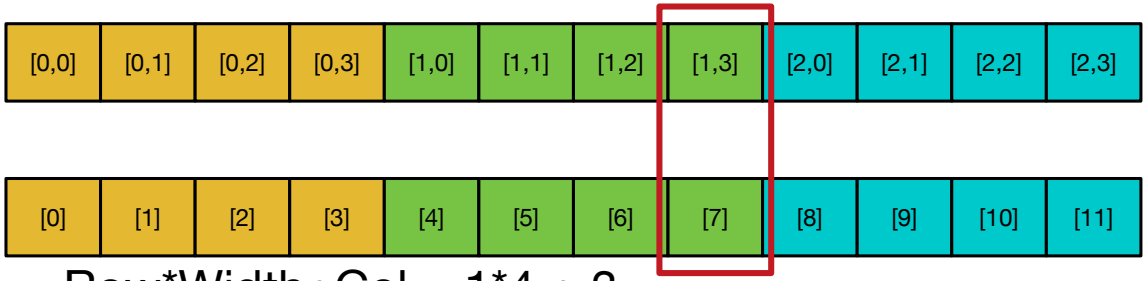
Row-major order



Column-major order





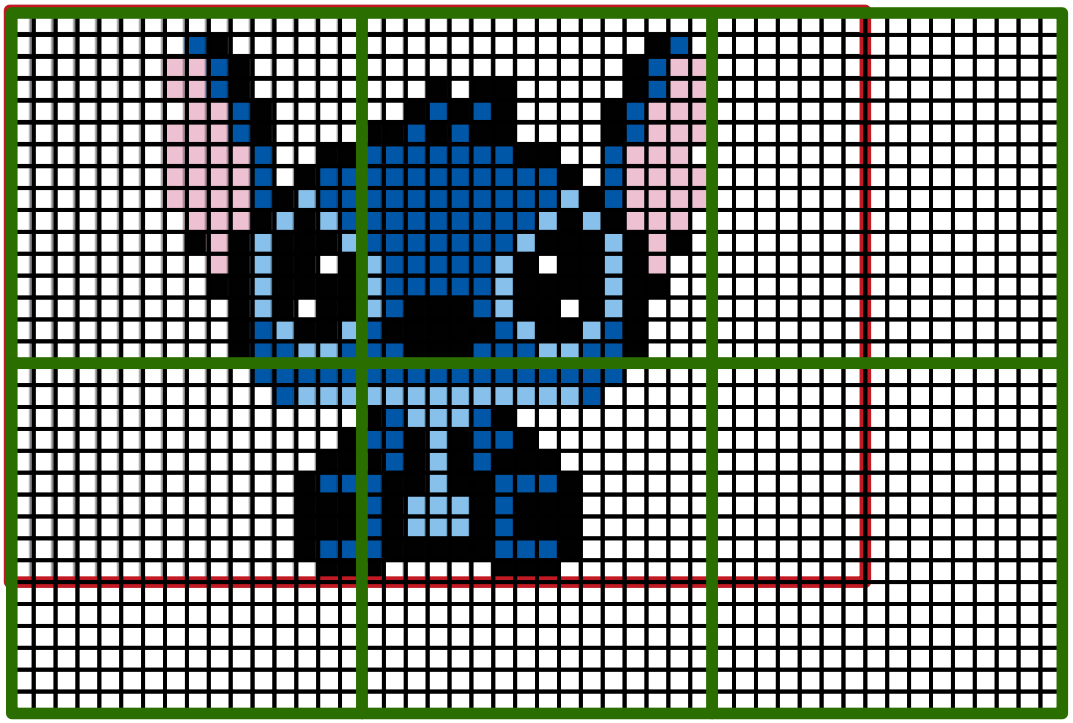


Row*Width+Col = 1*4 + 3

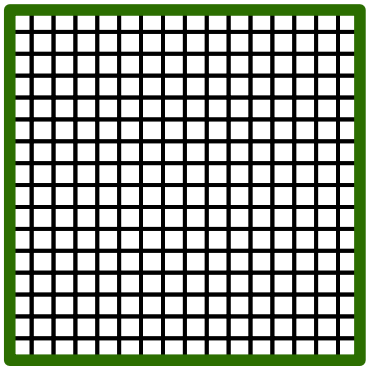
Pseudo-algo CPU

```
for ii from 0 to height do
  for jj from 0 to width do
    idx = ii * width + jj
    r = input[3*idx]
    g = input[3*idx + 1]
    b = input[3*idx + 2]
    grayImage[idx] = (0.21*r + 0.71*g + 0.07*b)
  end
end
```

Création de la *grid*



grid : 3x2 blocks



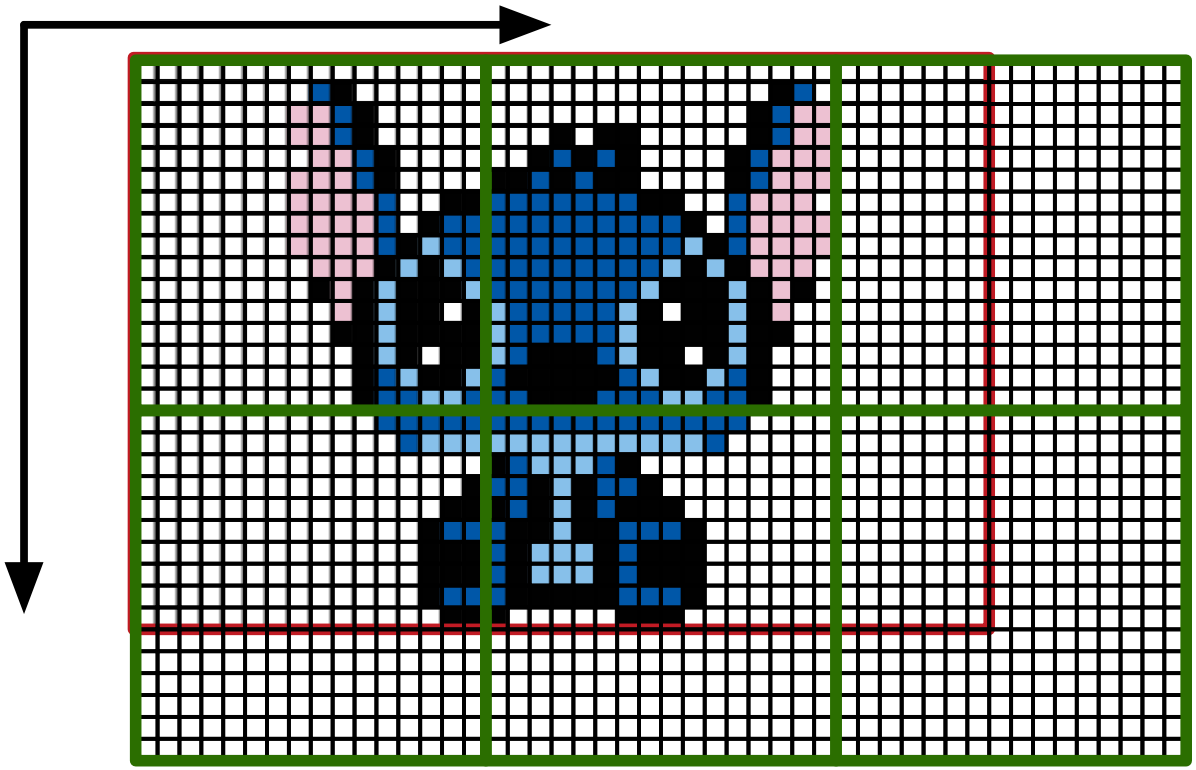
block : 16x16 threads

Kernel

```
#define CHANNELS 3 // we have 3 channels corresponding to RGB
// The input image is encoded as unsigned characters [0, 255]
__global__
void colorConvert(unsigned char * grayImage,
                  unsigned char * rgbImage,
                  int width, int height) {
  int x = threadIdx.x + blockIdx.x * blockDim.x; // Col
  int y = threadIdx.y + blockIdx.y * blockDim.y; // Row

  if (x < width && y < height) {
    // get 1D coordinate for the grayscale image
    int grayOffset = y*width + x;

    // one can think of the RGB image having
    // CHANNEL times columns than the gray scale image
    int rgbOffset = grayOffset*CHANNELS;
    unsigned char r = rgbImage[rgbOffset    ]; // red value for pixel
    unsigned char g = rgbImage[rgbOffset + 2]; // green value for pixel
    unsigned char b = rgbImage[rgbOffset + 3]; // blue value for pixel

    // perform the rescaling and store it
    // We multiply by floating point constants
    grayImage[grayOffset] = 0.21f*r + 0.71f*g + 0.07f*b;
  }
}
```

Création de la *grid*

```
int y = threadIdx.y + blockIdx.y * blockDim.y;
```



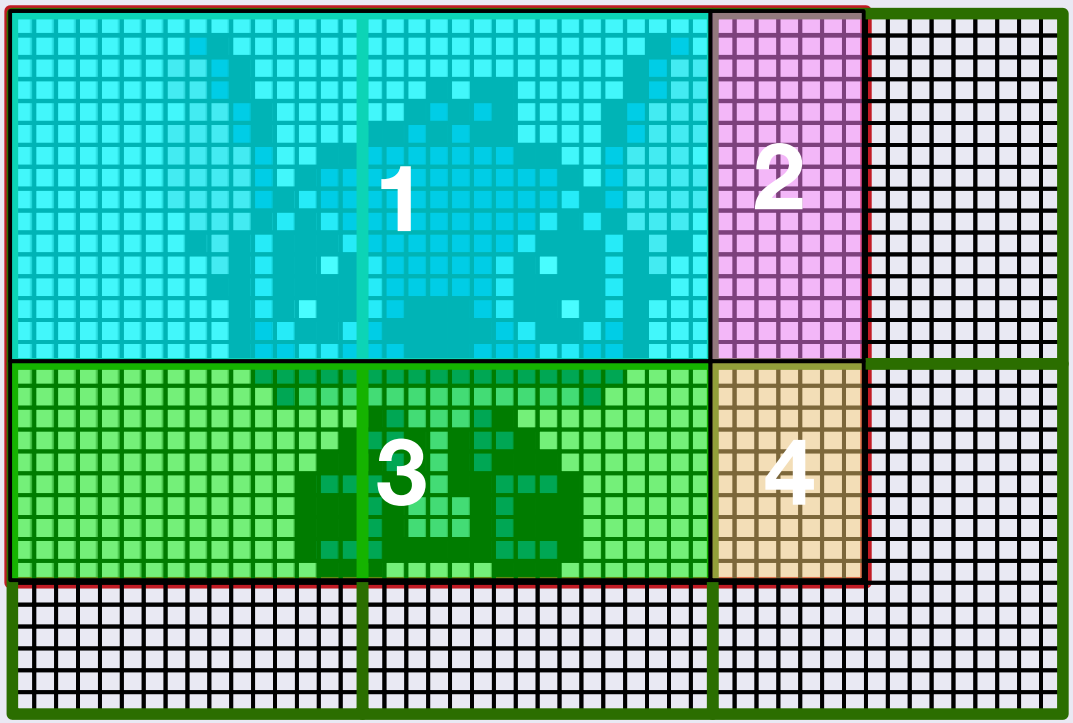
```
int x = threadIdx.x + blockIdx.x * blockDim.x;
```

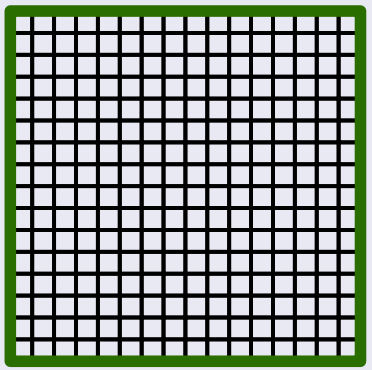
Code de l'*host* pour lancer l'exécution

```
// assume that the picture is m X n,
// m pixels in y dimension and n pixels in x dimension
// input d_Pin has been allocated on and copied to device
// output d_Pout has been allocated on device
...
dim3 DimGrid((n-1)/16 + 1, (m-1)/16+1, 1);
dim3 DimBlock(16, 16, 1);
colorConvert<<<DimGrid,DimBlock>>>(grayImage, rgbImage, width, height);
...
```

Couverture de l'image par les blocs

Tous les threads dans un bloc n'ont pas le même chemin dans leur flot de contrôle.





block : 16x16 threads