1 Rattrapage

1.1 TP 1

Finissez le TP 1.

2 Traitement simple d'images

Dans cette partie, vous allez réaliser un programme en C/C++ ou Python vous permettant d'appliquer un filtre de convolution 3x3 exécuté sur le GPU. Votre programme devra effectuer les opérations suivantes :

1. Chargement de l'image à traiter en mémoire
2. Envoi de l'image chargée dans la mémoire GPU
3. Exécution du traitement sur le GPU (appel d'un kernel CUDA)
4. Récupération de la donnée traitée résidant en mémoire GPU dans la RAM
5. Sauvegarde du résultat dans une image

À chaque étape, pensez à bien vérifier les codes de retour et les erreurs des fonctions que vous utilisez (ainsi que les résultats des allocations mémoire en C/C++).
Pour le chargement et la sauvegarde d'une image en C/C++, vous pouvez utiliser la bibliothèque *STB Image* (header only, simple d'utilisation, license permissive, ...) :

`https://github.com/nothings/stb/blob/master/stb_image.h`
`https://github.com/nothings/stb/blob/master/stb_image_write.h`

Voici un exemple simple d'utilisation de cette bibliothèque pour charger une image :

```
#include ...
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

...
  char const * const filename = "image01.jpg";
  int width = 0, height = 0, nchannels = 0;
  int const desired_channels = 1; // request to convert image to gray

  // load the image
  unsigned char * image_data = stbi_load(filename, &width, &height,
                                         &nchannels, desired_channels);
  // check for errors
  if(!image_data || !width || !height || !nchannels)
  {
    printf("Error loading image %s \n", filename);
    return -1;
  }

  // use the image data
  ...

  // release the image memory buffer
  free(image_data);

...
```

Et par exemple, pour sauvegarder une image :

```
#include ...
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"

...
  char const * const filename = "image01.png";
  int const width = 128;
  int const height = 128;
  int const nchannels = 3;
  int const stride = width * nchannels;

  // suppose we already have a buffer of 128*128*3 bytes
  unsigned char * data = ...

  // save the image
  if(!stbi_write_png(filename, width, height, nchannels, data, stride))
    ...
```

En Python, vous pouvez par exemple utiliser Matplotlib.Image :

```
from matplotlib import image
from numpy import asarray

image = image.imread('input.jpg')
# print(data.shape) --> (1080, 2280, 3) for instance for a color image

# convert to grayscale
imagegray = np.dot(image[...,:3], [0.2989, 0.5870, 0.1140])

data = asarray(imagegray, dtype=np.float32)

# use the image data
...


# Save a processed image
image.imsave('name.png', array)


```

2.1 Implémentation naïve

Ne vous posez pas de question, codez l'application du noyau 3x3 en Cuda simplement, sans se soucier des goulots d'étranglements liés aux accès mémoire.
Ajoutez le nécessaire dans votre code pour obtenir une mesure de performance (par exemple en pixels/seconde et en images/seconde) incluant seulement le temps de traitement, sans le temps d'upload de l'image dans le GPU ni le temps de récupération de la donnée traitée.
Quelles sont les performances ?

2.2 Utilisation de la mémoire partagée

Pour améliorer grandement les performances de votre précédente implémentation, utilisez la mémoire partagée entre les threads d'un même block (`__shared__`) sur les zones mémoire que vous jugerez pertinentes.
Quelles sont les nouvelles performances comparées par rapport aux précédentes ?