TP1 A5 - GPU

Parallélisation d'algorithmes sur GPU embarqué

M2 SETI

N. Gac

Carte Jetson nano

Lors de ce TP, vous programmerez un algorithme de seuillage couleur à l'aide de trois langages de programmation associés chacun à des objectifs distincts mais complémentaires :

1. Etude algorithmique (Matlab sur PC)
2. Implémentation de référence sur cible embarquée (C sur ARM)
3. Accélération sur cible embarquée (CUDA sur GPU)

Les images et codes de démarrage se trouvent sur https://gitlab.com/nicolas.gac/TP_GPU/ -> TP1_image.

1 Etude algorithmique (DSL [1] Matlab)

Objectif : Programmer l'algorithme de seuillage couleur avec un langage haut niveau (matlab) afin d'en avoir une validation fonctionnelle (et un premier temps d'exécution).

L'image couleur *ferrari.jpg* est une image RGB (Red Green Blue). Cette image RGB est stockée dans trois tableaux 2D, chacun correspondant à une des trois couleurs Rouge, Vert et Bleu.

Travaillez sous Matlab dans le répertoire *seuillage_ Matlab*. Utilisez comme script de départ le fichier *seuil_ Matlab.m*.

1.1 Ferrari rouge

Seuillez l'image I en fonction de η_R niveau de rouge de chaque pixel de l'image I :

$$I_s(xn, yn) = \begin{cases} I(xn, yn) \text{ si } \eta_R > 0.7 \\ 0 \text{ sinon} \end{cases}$$

$$\eta_R = \frac{I_R}{\sqrt{I_R^2 + I_G^2 + I_B^2}}$$

1.2 Ferrari jaune

Changez les pixels rouge ($\eta_R > 0.7$) en jaune.

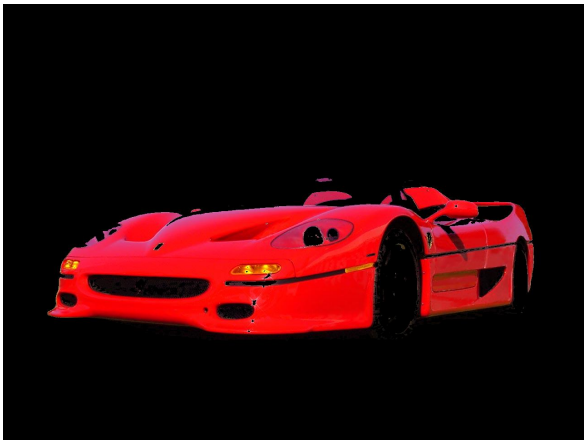
Indice : Jaune=Rouge+Vert...

1.3 Optimisation de code Matlab

Contrairement au C qui est un langage compilé (apres compilation, le code binaire est prêt à être exécuté), Matlab est un langage interprété. Chaque ligne de code est traduite en un mini exécutable. L'analyse de chaque ligne de code implique un surcoût en temps d'exécution. Optimisez votre code en supprimant au maximum les boucles for qui ralentissent les performances des codes Matlab.

1. DSL : Domain Specific Language

(a) I image d'origine



(b) I_{seuil} Image seuillée

(c) I_{jaune} Image avec le changement de couleur

2 Exécution sur CPU (langage C)

Objectif : Programmer l'algorithme de seuillage couleur sur une cible embarqué avec CPU (processeur ARM de la carte Jetson nano) avec un langage standard (C) afin d'obtenir une validation fonctionnelle du code C et un temps d'exécution de référence sur cible embarquée.

2.1 Code source et compilation

Vous travaillerez dans le répertoire *TP1_ image/CUDA* qui contient trois fichiers sources :

- **seuillage_main_student.cu** qui contient le *main* exécuté sur le CPU et le kernel *seuillage_ kernel* exécuté sur le GPU.
- **seuillage_C_student.cpp** qui contient la fonction *seuillage_ C* exécuté sur le CPU.
- **seuillage.h**

Pour compiler :

1. Depuis le répertoire *TP1_ image/CUDA*, créez un répertoire build ('**mkdir build**') puis aller dedans ('**cd build**')
2. Créez le fichier Makefile à l'aide de cmake('**cmake ..**/')
3. Compilez avec make ('**make**')
4. Lancez l'exécutable ('**./seuillage_CUDA ../../Image**'). L'exécutable a un argument correspondant au chemin contenant l'image à traiter.

2.2 Code à trou

Codez la fonction seuillage_C contenu dans *seuillage_ C.cpp*. Vérifiez si vous obtenez le bon seuillage en observant sous ImageJ, l'image de sortie *Image\ferrari_ out_ CPU.raw*. Comparez la avec celle obtenue sous Matlab *Image\ferrari_ out_ Matlab.raw*.

Notez le temps de calcul, comparez le avec celui obtenu sous Matlab.

2.3 Optimisation de code sur CPU

Inversez l'ordre des boucles (boucle selon les lignes et colonnes de l'image). Notez le temps de calcul. Comparez le temps de calcul pour les deux ordres de boucle. Pourquoi existe t il une telle différence ?

3 Parallélisation sur GPU (langage CUDA)

Objectif : Programmer l'algorithme de seuillage couleur sur un GPU embarqué (processeur ARM de la carte Jetson nano) avec le langage (CUDA) afin d'obtenir une accélération notable sur cible embarquée par rapport à l'implémentation de référence sur CPU.

3.1 Puissance de calcul de la carte

Utilisez l'executable **'deviceQuery'** pour connaitre les caractéristique du GPU de la carte utilisée (nombre de coeurs, fréquence, ...). Cette exécutable fait partie des *CUDA samples* habituellement disponibles dans le répertoire */usr/local/cuda/samples/bin*.

3.2 Code à trou

En vous inspirant du code de multiplication de matrice vu en cours, complétez dans le fichier source **seuillage_main_student.cu** :

1. la fonction main où il manque les étapes d'allocation mémoire, les transferts mémoire entre le PC hôte et la carte GPU, le découpage en threads et le lancement du kernel.
2. le kernel *seuillage_kernel*

qui définit le programme exécuté par chaque thread CUDA.

Vérifiez si vous obtenez le bon seuillage en observant sous ImageJ, l'image de sortie *Image \ferrari_out_GPU.raw*. Comparez la avec celle obtenue sous Matlab *Image \ferrari_out_Matlab.raw*.

3.3 Accélération obtenue

Mesurez le temps de calcul et comparez le avec celui obtenu en C et Matlab. Quel serait le facteur d'accélération avec une carte A100 ayant 6192 coeurs @ 1.41 Ghz ?