M2 - Architecture et Programmation d'accélérateurs Matériels.

(APM 2016-2017)

TP4

Réduction
Stencils

hugo.taboada.ocre@cea.fr
julien.jaeger@cea.fr
patrick.carribault@cea.fr

I Réduction somme en CUDA

Une réduction somme consiste à additionner toutes les valeurs d'une tableau. Une écriture séquentielle d'une réduction pourrait être la suivante :

```
float sum = 0;
for (int i = 0; i < ntot; i++)
        sum += tab[i];
```

On désire écrire une implémentation CUDA de la réduction somme (*tab* se trouve sur le GPU). Pour implémenter la réduction, nous proposons d'opérer en deux étapes :

1. Une première réduction dans chaque bloc. On obtient ainsi à la fin un tableau dimensionné au nombre de blocs et dont les valeurs sont les sommes partielles de chaque bloc.
2. Une seconde réduction sur les sommes partielles. On obtient ainsi la somme totale des éléments du tableau.

Q.1: Implémenter le kernel `reduce_kernel(float *in, float *out)` (voir fichier *reduce.cu*) permettant de faire les sommes partielles par bloc.
in est le tableau de valeurs à réduire dimensionné au nombre total de threads dans la grille, et *out* le tableau de valeurs réduites par bloc, dimensionné au nombre de bloc.
Pour réaliser cette réduction, vous utiliserez une méthode arborescente, ainsi que la fonction `__syncthreads()` qui permet de synchroniser à l'intérieur d'un kernel tous les threads d'un même bloc.
Nous nous placerons sous les hypothèses suivantes :

- Le nombre de blocs et de threads par bloc sont des puissances de 2.
- La taille du tableau est égale au nombre de threads.

Q.2: Utiliser le même kernel pour terminer la réduction (étape 2).

Q.3: Généraliser la réduction à une taille quelconque de tableau.

II Modification d'image : flou et flou iteratif

Le programme *tp4_video1.c* présent dans le répertoire *CODE/Partie2/VIDEO* lit une vidéo, la traduit sous forme de tableaux de pixels, puis modifie ces pixels afin de transformer la vidéo couleur en vidéo en niveaux de gris. Vous pouvez compiler et exécuter le programme, en utilisant la petite vidéo test fournie *Wildlife.wmv*, pour voir l'effet du programme sur la vidéo générée *my_copy.wmv*.

Comme pour le TP précédent, dans le même répertoire se trouve le fichier tp4_video1.cu.

Q.4: Dans ce TP, nous allons implémenter une nouvelle modification d'image : le flou. Le flou est une opération de stencil : pour mettre à jour la valeur de la case, nous avons besoin des donnés des cases voisines. Pour effectuer un flou, il suffit de mettre à jour un pixel avec la moyenne des valeurs de ce pixel et des pixels voisins, pour chaque composante. Dans notre cas, nous allons uniquement considérer les voisin directs. Pour un pixel[i][j], il d'agit des pixels [i-1][j], [i+1][j], [i][j-1] et [i][j+1].

Implémenter le kernel permettant de faire un flou sur les frames 200 à 400.

Q.5: Le flou n'est pas très visible. Nous allons réaliser un flou itératif en appliquant 500 fois notre modification.

Q.6: Pour le moment, notre flou itératif n'est pas correct. Nous appliquons 500 notre flou avec les valeurs d'origine, alors qu'il faudrait réaliser chacun de ces flous sur les nouvelles valeurs. Nous avons donc besoin de nous synchroniser entre chaque itération.

Dans un premier temps, synchrniser uniquement les threads au sein d'un bloc.

Q.7: En vous inspirant de la partie 1, synchrniser tous les threads au sein d'un kernel. Pour éviter les deadlocks, assurez-vous de ne pas mettre plus de blocks que de SMs disponibles sur votre carte graphique.