TP Partie 2 : flou et convolutions

Pendant ce TP vous allez devoir flouter une image. Pour ce faire, imaginez que nous avons un tableau réprésentant les valeurs des pixels. Pour chaque pixel de l'image, imaginez que l'on superpose un tableau de poids centré sur le pixel courant. Pour calculer la valeur du pixel flouté, on multiplie chaque paire de nombre qui se font face. C'est-à-dire, on multiplie chaque pixel avec le poids "au dessus". Pour finir, on somme tous les résultats des multiplications et on donne cette valeur au pixel en cours. On répète ce procédé pour tous les pixels de l'image.

Pour vous aidez à commencer, voici quelques notes :

Etape 0

Transformation de la structure

Une image en couleur possède plusieurs canaux de couleurs (RGBA). Nous allons les séparer pour avoir des tableaux continus de valeur plutôt qu'entrelacé. Cela simplifiera le code.

Au lieu d'avoir RGBARGBARGBARGBARGBA... nous préférons avoir 3 tableaux distincts (on ignore le canal alpha):

1. RRRRRRRR...
2. GGGGGGGG...
3. BBBBBBBB...

La structure initiale se nomme Array of Structures (AoS) tandis que la seconde est Structure of Arrays (SoA).

En guise d'échauffement, vous allez devoir écrire un kernel qui effectue cette séparation. Ensuite, le coeur du problème est le flou (ou convolution). Un kernel qui recombine les canaux est déjà fourni.

Convolutions

Vous devez compléter le kernel `gaussian_blur` pour effectuer le floutage de inputChannel en utilisant le tableau de poids puis stocker le résultat dans outputChannel.

Voici un exemple de comment calculer le flou en utilisant une moyenne pondérée, pour un seul pixel de l'image :

Tableau des coefficients :

0.0	0.2	0.0
0.2	0.2	0.2
0.0	0.2	0.0

Image (notez qu'on aligne le tableau des coefficients au centre de la "boîte") :

1	2	5	2	0	3			
	---	---	---					
3	|2	5	1|	6	0		0.0*2 + 0.2*5 + 0.0*1 +	
4	|3	6	2|	1	4	->	0.2*3 + 0.2*6 + 0.2*2 +	-> 3.2
0	|4	0	3|	4	2		0.0*4 + 0.2*0 + 0.0*3	
	---	---	---					

9	6	5	0	3	9			
		(1)					(2)	(3)

Par où commencer

Comme auparavant, vous aurez à lier chaque thread à un pixel de l'image. Ensuite, chaque thread peut faire les étapes 2 et 3 du diagramme ci-dessus indépendamment des autres.

Notez que le tableau des coefficients est un carré, sa hauteur est égale à sa largeur. On nomme le tableau des coefficients `filter` et sa largeur `filterWidth`.

De plus, vous pouvez utiliser la fonction `checkCudaErrors`. Elle vous permettra de vérifier qu'une fonction s'est bien déroulée. Voici un example de comment l'utiliser : Avant :

```
cudaMalloc(&d\_red, sizeof(unsigned char) * numRows * numCols);
```

Après :

```
checkCudaErrors(cudaMalloc(&d\_red, sizeof(unsigned char) * numRows * numCols));
```

Écrire du code propre et sûr est toujours un peu plus long, mais c'est indispensable pour corriger ses erreurs rapidement. Si vous ne faites pas cette vérification et qu'il y a une erreur, tous les appels suivants ne vont rien faire, et se sera plus difficile de se rendre compte de pourquoi. Ecrire du code sûr vous permettra d'avoir l'information dès qu'il y a une erreur.

Enfin, n'oubliez de libérer la mémoire de ce que vous avez alloué.

Questions : 1. Une fois que vous avez vérifié que votre programme fonctionne sur la petite image `cinque_terre_small.tiff`, donnez le temps de calcul sur la grande: `cinque_terre_large.tiff`. 2. Encore une fois, essayez de trouver une taille de block / grille qui maximise la performance.

Etape 1

Pour cette nouvelle étape, il faut mettre en place une optimisation sur notre code. Vous pouvez repartir de ce que vous avez déjà en étape 0.

Le filtre de convolution va maintenant être directement stocké dans la mémoire constant plutôt qu'en mémoire globale.

La variable a déjà été déclarée `filter_constant` mais il faut transférer les données sur cet espace grâce à cudaMemcpyToSymbol. Vous pouvez utiliser la documentation de CUDA pour vous aider.

Questions : 3. Quel est le speedup obtenu avec cette amélioration sur la grande image ? ((Speedup = Ancien_tps_de_calcul - Nouveau_tps_de_calcul) / Nouveau_tps_de_calcul)

Etape 2

Encore une fois, vous pouvez repartir du travail fait dans les étapes précédantes.

Maintenant, on veut chercher a réutiliser les valeurs de l'image lues dans le bloc. Pour cela, le bloc va tout d'abord écrire dans la mémoire partagée ce dont il aura besoin.

Questions : 4. Pour une convolution avec un filtre de largeur et hauteur 5, combien de fois chaque pixel de l'image doit-il être lu ? 5. Quel est le speedup par rapport à la partie 2 ? 6. (Bonus) Pour quelle(s) raison(s) le speedup n'est pas si élevé ?