RAPPORT PROJET CUDA 2023

OGER Julien
STEAD Enzo

I- Les implémentations

En plus du filtre Sobel, nous avons choisi d'implémenter les différents filtres suivants :

- blur : filtre avec une matrice de convolution de taille 3*3
- line : filtre avec une matrice de convolution de taille 3*3
- edge : filtre avec une matrice de convolution de taille 3*3
- laplacian of gaussian : filtre avec une matrice de convolution de taille 5*5

Chacun de ces filtres a d'abord été implémenté de façon séquentielle en C++ puis avec une version Cuda. La plupart de ces filtres ayant une matrice de convolution de taille 3*3, leur implémentation Cuda a été assez similaire à celle du filtre Sobel. Cependant, le filtre laplacian of gaussian a présenté plus de difficultés de par sa matrice de convolution plus grande.

Du fait de ces similitudes, nous avons implémenté pour ces filtres uniquement la version fusionnée avec mémoire partagée (qui est globalement la version qui semble la plus efficace) et nous avons réalisé les tests de performance entre les différents kernel (2 étapes, mémoire partagée et fusionnée) sur le filtre Sobel.

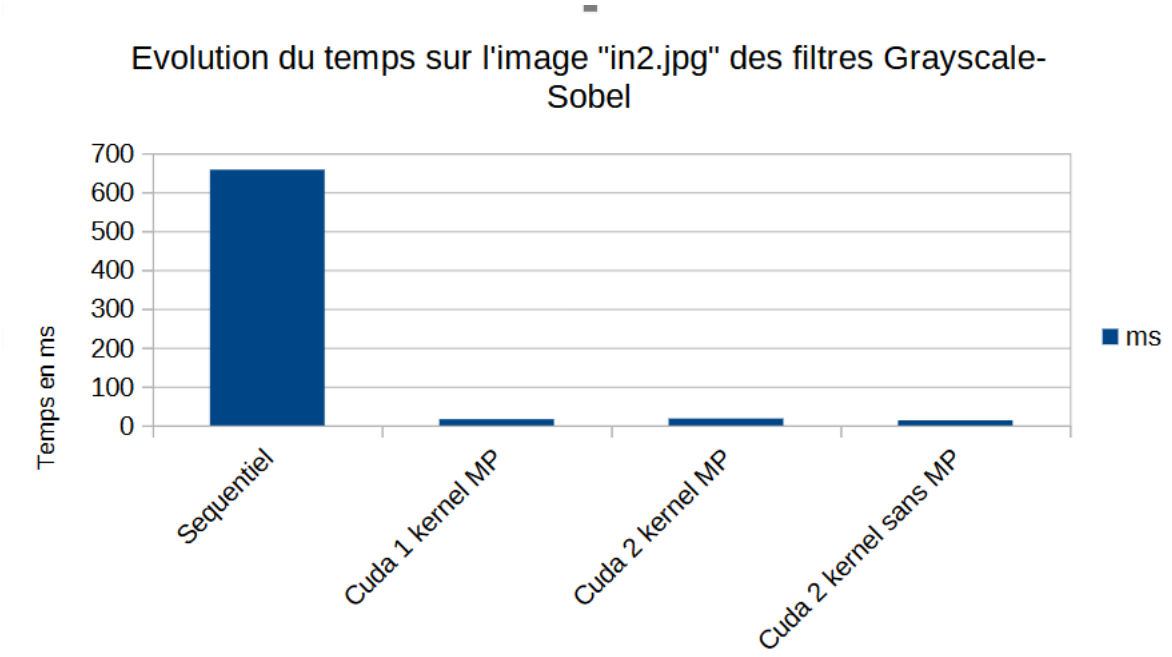
De plus, nous avons implémenté une version de Sobel avec des streams pour pouvoir réaliser des tests de performance sur cette version.

Vous trouverez dans la suite de ce rapport les tests de performance réalisés et nos observations à ce sujet.

II- Tests de performance

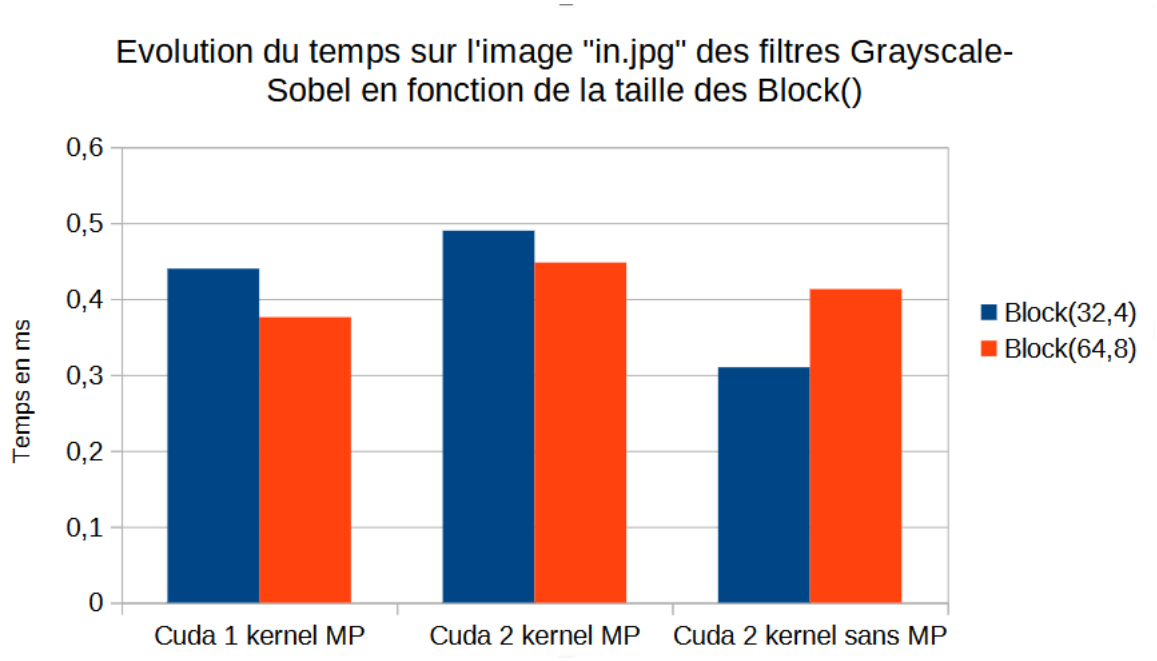
a) Les différents kernels

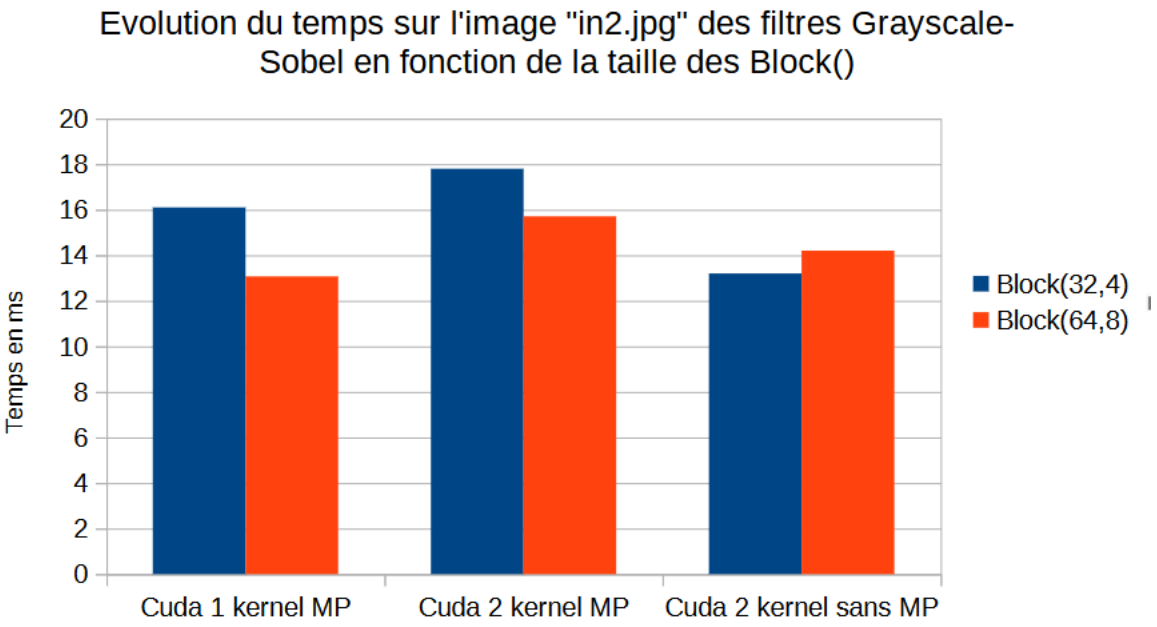
Dans un premier temps, nous avons réalisé une comparaison des performances entre la version séquentielle du filtre Sobel et les différents kernels Cuda à disposition. Ces tests ont été effectué avec des blocks de taille (32, 4).
Vous pouvez retrouver respectivement la version séquentielle, la version fusionnée avec mémoire partagée, la version deux étapes avec mémoire partagée et finalement la version deux étapes sans mémoire partagée.
On observe une efficacité extrêmement plus importante de la version Cuda par rapport à la version séquentielle.



b) Variabillisé la taille des blocks

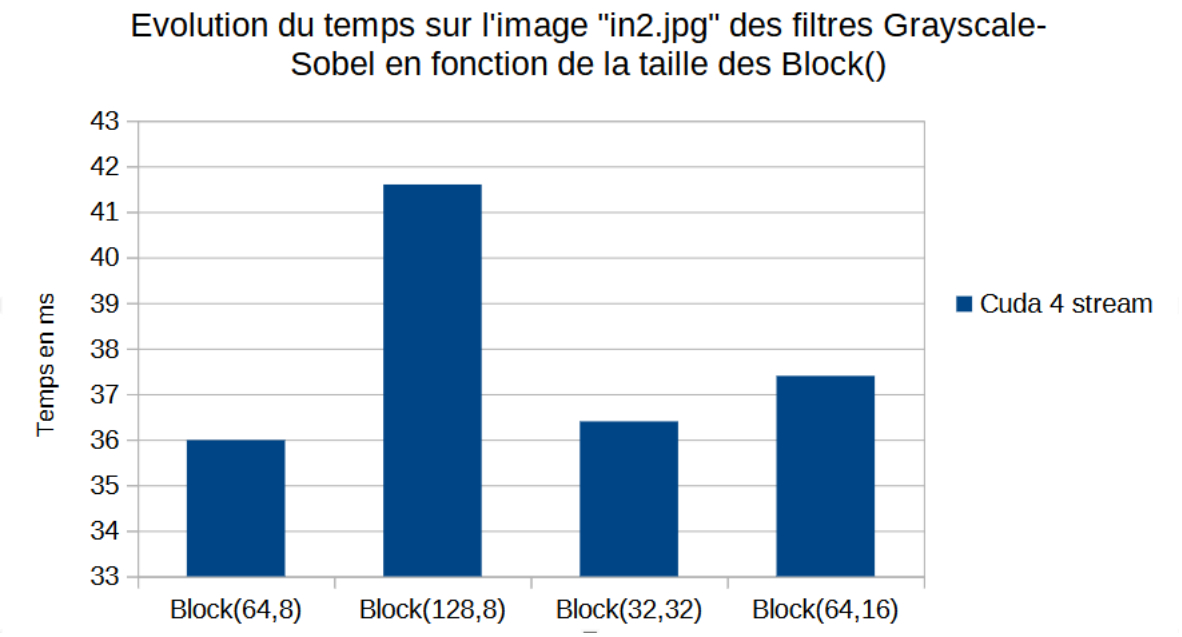
Étonnés comme c'est visible dans le graphe précédent d'observer un temps d'exécution plus rapide de la version deux étapes sans MP (mémoire partagée) par rapport à la version fusionnée, nous avons commencé à faire des tests en changeant la taille des blocks et nous avons pu constater que des blocks de (64, 8) sont plus efficaces que les blocks de (32, 4) précédents et présentent plus de cohérence sur les performances d'un kernel à l'autre.





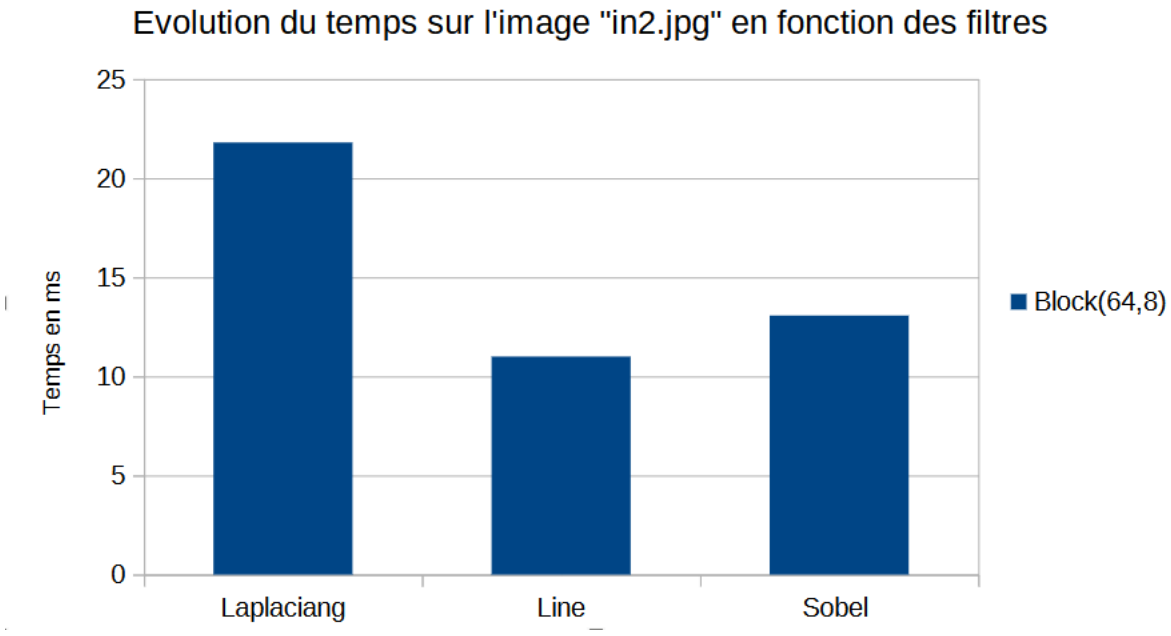
c) Vers une version avec streams

Curieux d'observer l'efficacité du filtre Sobel avec des streams, nous avons implémenté ces derniers. Cependant, malgré nos tests, la version avec streams semble globalement moins performante que la version fusionnée, même si elle reste bien plus efficace que la version séquentielle. Notre théorie est que la communication nécessaire d'un stream à l'autre (pour communiquer les voisins, via des blocks de plus grande taille) entraîne une perte de performance. Leur utilisation semble moins adapté à l'application d'une matrice de convolution. Voici le résultat de nos différents tests :



d) D'un filtre à l'autre

Enfin, nos tests étant concluants sur le filtre Sobel, nous avons réalisés des mesures sur les performances de nos implémentations des différents filtres dont voici le résultat ci-dessous :



III- Conclusion

L’application de filtre à l’aide de matrice de convolution présente la difficulté de partager des informations pour ne pas perdre de données. En effet, chaque pixel a besoin des informations des pixels voisins, plus ou moins selon la taille de la matrice de convolution. Nous avons dû en effet jouer sur la taille des blocks et le nombre de blocks notamment pour le filtre laplacian gaussian pour ne pas avoir de perte de données (matrice de convolution de taille 5*5). Cependant, réaliser les calculs sur la carte graphique permet une amélioration très importante des performances et il est nécessaire d’optimiser à minima la taille des blocks pour obtenir les meilleures performances possibles.