

Programmation haute Performance - GPU - OpenCL

Marion PAVAUX, Erwan LAMBERT

20 mai 2021

1 Description des filtres

Durant le TP, nous avons codé et optimisé deux filtres: un filtre moyenneur, qui introduit un flou dans l'image, et un filtre gaussien qui introduit un bruit.
Voici le résultat que donnent nos filtres sur l'image initiale:



Figure 1: Image filtrée avec un filtre moyenneur

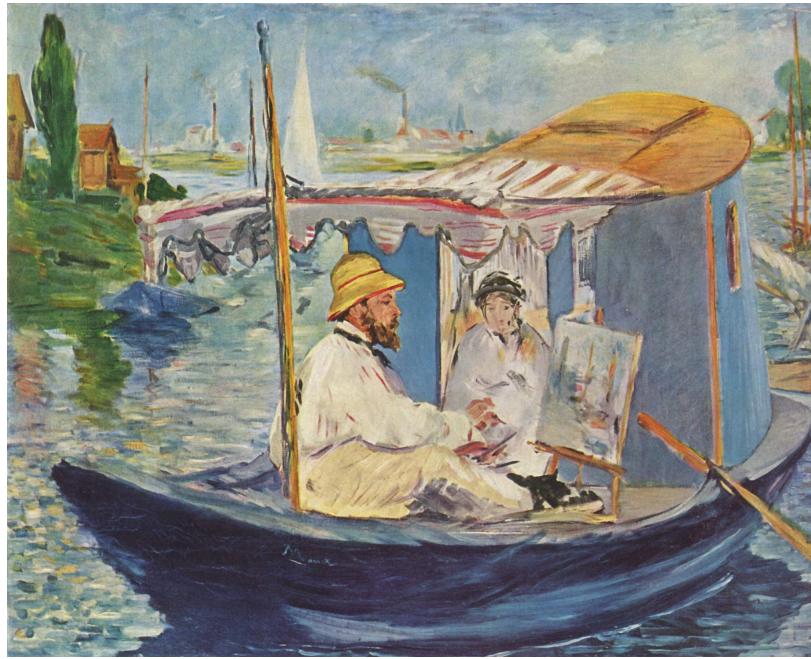


Figure 2: Image filtrée avec un filtre gaussien

2 Optimisation

2.1 Vectorisation

OpenCL nous permet d'utiliser des variables de type `uchar4 *` qui sont l'équivalent de quatre `unassigned char *`.

```
kernel void filtermeanimproved(__global const uchar4 *imageInput, __global uchar4 *imageOutput, __global const float*coef, int width, int height, int F)
```

Cela nous évitera les recopiages inutiles, et les multiplications par 4. À la manière des arrays numpy, la façon dont ils sont gérés optimise grandement le code. De la même manière la moyenne, et la gaussienne sont gérées pour chaque canal en tableau de quatre.

```
float4 mean = (float4)0.0; et float4 gaussian = (float4)0.0;
```

2.2 Utilisation de buffers, limitation des calculs redondants

Lorsqu'on regarde la formule de transformation par le filtre gaussien de l'image originale, on observe un certain nombre de calculs redondants. En effet, pour chaque processeur, le calcul suivant, la fonction gaussienne, est effectué pour l et m allant de -N à N.

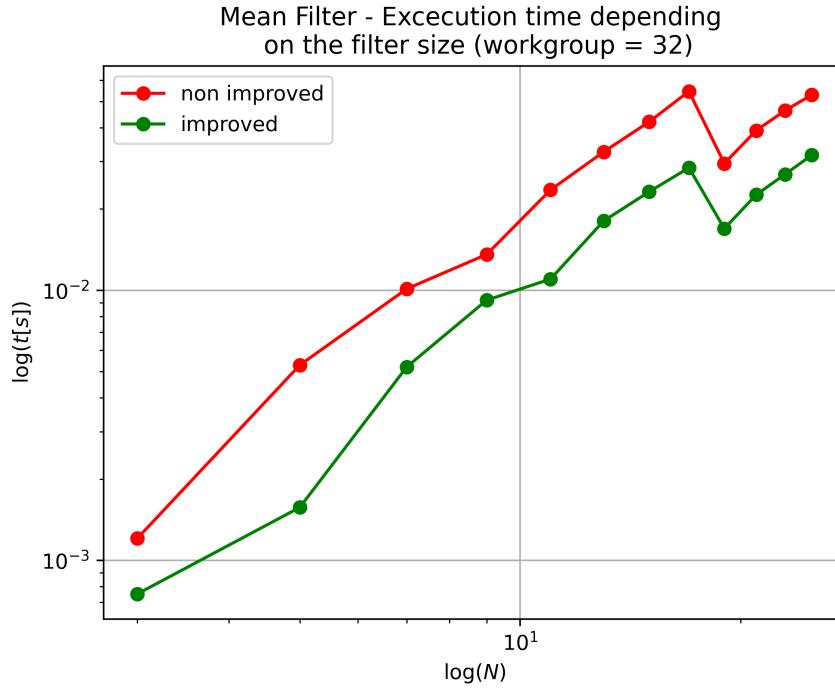


Figure 3: Influence de la taille du filtre avec le filtre moyenneur

e

$$cg(l, m) = \frac{1}{\sqrt{2\pi\sigma^2}} * e^{-\frac{l^2+m^2}{2\sigma^2}}$$

Pour chaque processeur la somme de ces coefficients est réalisée pour constituer le terme de normalisation. Chaque fonction gaussienne et aussi multipliée par un élément de l'image originale, avant encore une fois d'être sommée sur la taille du filtre.

Puisque le terme de normalisation est le même pour chaque processeur, le choix a été fait de le calculer en amont en séquentiel par le CPU, et de le passer en argument des kernel par l'intermédiaire des buffers.

Pour ce qui est des coefficients gaussiens, on peut aussi remarquer deux symétries: $cg(l, m) = cg(m, l)$ et $cg(l, m) = cg(-l, -m)$.

Ainsi, on peut, grâce à la première propriété, calculer en amont des demi-coefficients gaussiens, et grâce à la deuxième propriété, les calculer de 0 à N au lieu de -N à N. On crée alors un tableau de taille N+1 en amont, passé en argument, avec les coefficients:

$$demicg(l) = \frac{1}{(2\pi\sigma^2)^{1/4}} * e^{-\frac{l^2}{2\sigma^2}}$$

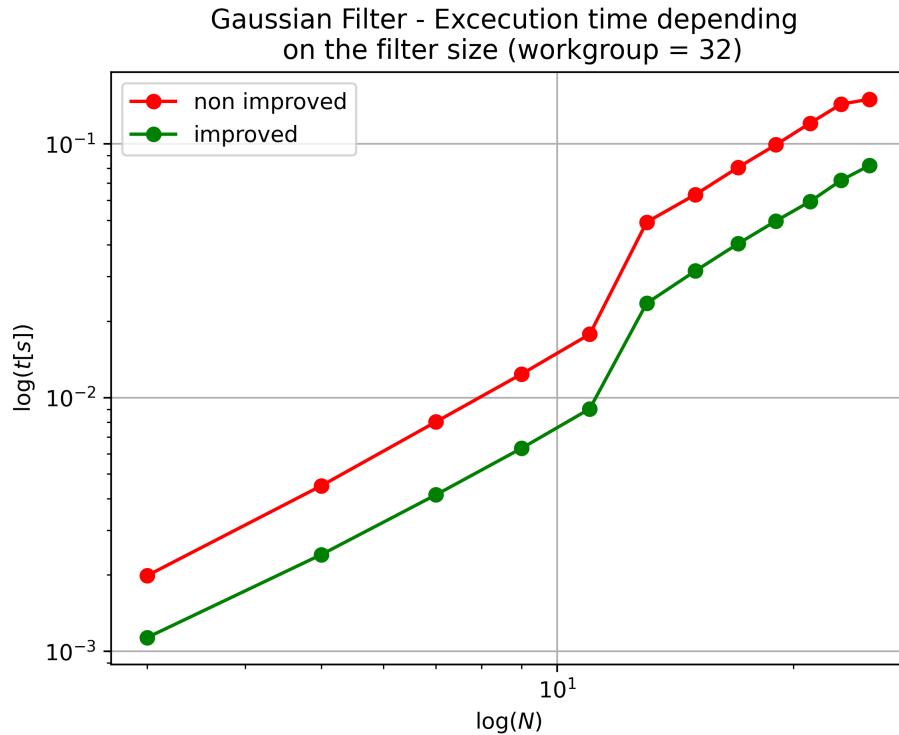


Figure 4: Influence de la taille du filtre avec le filtre gaussien

On voit que l'amélioration finale avec le filtre gaussien (tenant aussi en compte l'amélioration précédente) est très constante.

Pour chacun des filtres, le temps d'exécution augmente avec la taille du filtre [Figures 3 et 4], qu'il soit improoved (calcul vectoriel et utilisation de buffers) ou non improved. En effet plus la taille du filtre est grande plus il y a de terme à calculer pour chaque pixel.

2.3 Taille du workgroup

La dernière amélioration que nous avons essayé était de changer la taille du workgroup.

On peut observer sur les graphiques ci-dessous deux tendances. Augmenter la taille du workgroup jusqu'à 8 permet d'augmenter drastiquement les performances, d'autant plus pour un algorithme non amélioré. Cependant au-delà, la performance n'est pas meilleure, voir peut même décroître très légèrement dans le cas de l'algorithme amélioré.

Ce résultat est quelque peu surprenant puisque pour d'autres executions, nous avons observé une nette accélération en passant la taille du workgroup de 16 à 32 GPU.

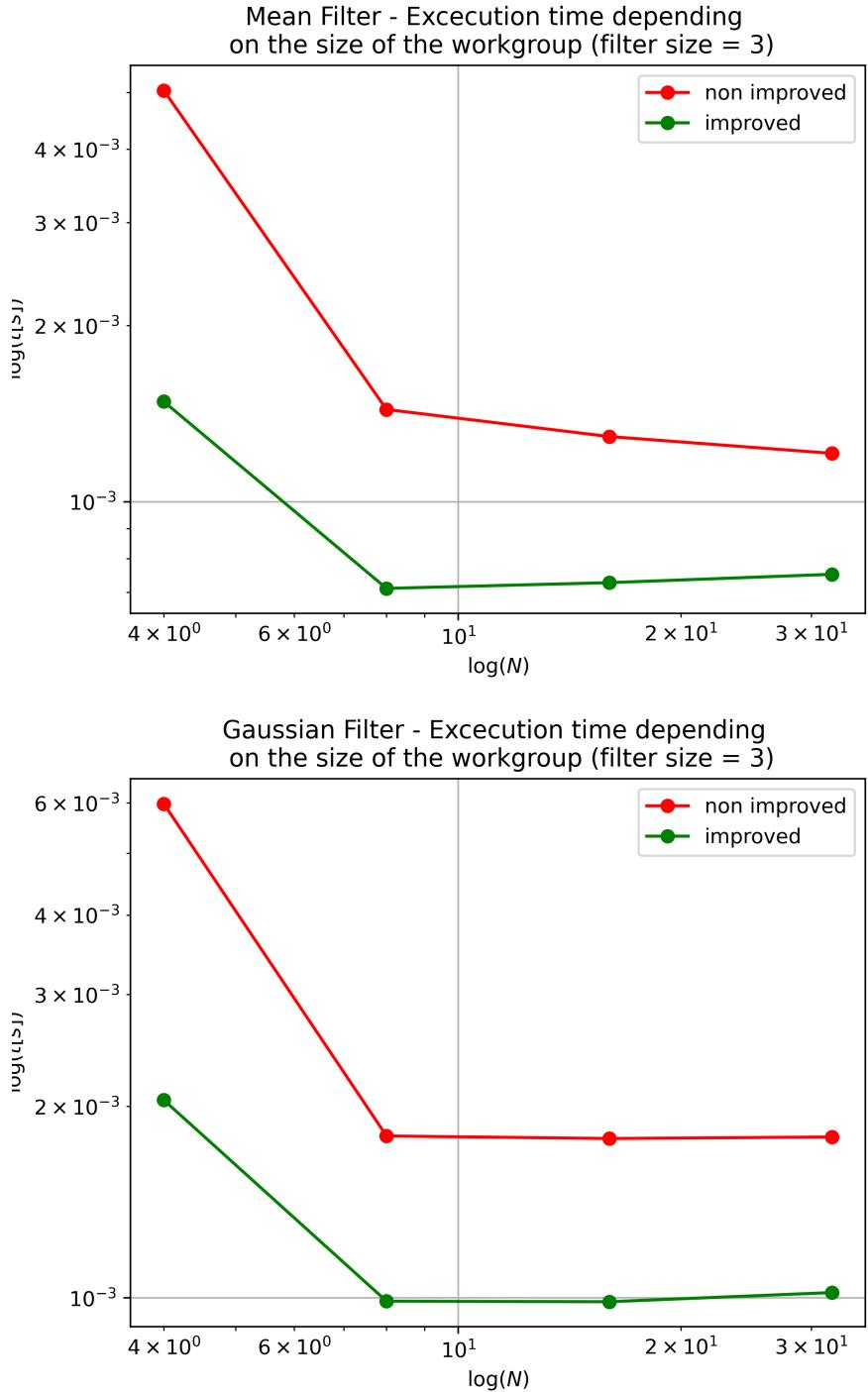


Figure 5: Influence de la taille du workgroup

3 Conclusion

Des améliorations ont été effectuées concernant:

1. la gestion des données en tableaux de quatre 4 constituant d'un pixel)
2. l'évitement des calculs redondants en passant des données en argument
3. la taille du workgroup

Concernant le filtre moyenneur, une amélioration aurait encore pû être effectuée en essayant d'éviter une concurrence à l'accès en lecture par les processeurs. En effet, chaque processeur essaye de lire, pour un filtre de 3*3, neuf pixels de l'image. Les processeurs voisins essayant de faire de même, sont mis en attente de lecture. Cela a donc une influence négative sur le temps d'exécution. Nous aurions pû éviter cela en réalisant des copies locales de l'image, faites par des processeurs dédiés à cette tâche.