Projet Cuda

Pierre Zachary - Eoghann Veaute
Mai 2022

Sujet

Réaliser un programme permettant de faire une convolution, sur différentes images et avec différents kernels. Ce programme devra permettre de comparer une version CPU et GPU.

Pour exécuter le projet, voir le fichier README.md, à la racine du projet.

Partie CPU

Pour la partie CPU, le programme est totalement séquentiel : On parcourt chaque octet de chaque pixels de l'image, et pour cet octet on détermine la nouvelle valeur obtenue grâce à la matrice kernel. Cela nous demande de parcourir les pixels qui entourent celui courant.
Si les pixels à parcourir sortent de l'image, on renvoie une valeur par défaut (nous avons choisi 127, ce qui explique que les bordures de l'image peuvent avoir un ton un peu gris).
Sinon on fait la somme des produits de la valeur du kernel à la case parcouru, à la valeur de l'octet du pixel correspondant. Si l'image a plusieurs octets par pixels, (image rgb par exemple), on prendra le même channel que l'octet que l'on est en train de déterminer, donc si je parcours un octet bleu, je prendrai comme valeur l'octet bleu de chaque pixels environnant, etc.

Partie GPU

La version GPU est très similaire, si ce n'est que chaque thread s'occupe individuellement d'un octet : on lance donc autant de threads qu'il y a d'octets dans l'image. De plus, on découpe l'image un certain nombre de fois pour la répartir sur plusieurs streams.

Respect des consignes

Comme il nous l'était demandé, nous avons respecté un certain nombre de consignes :

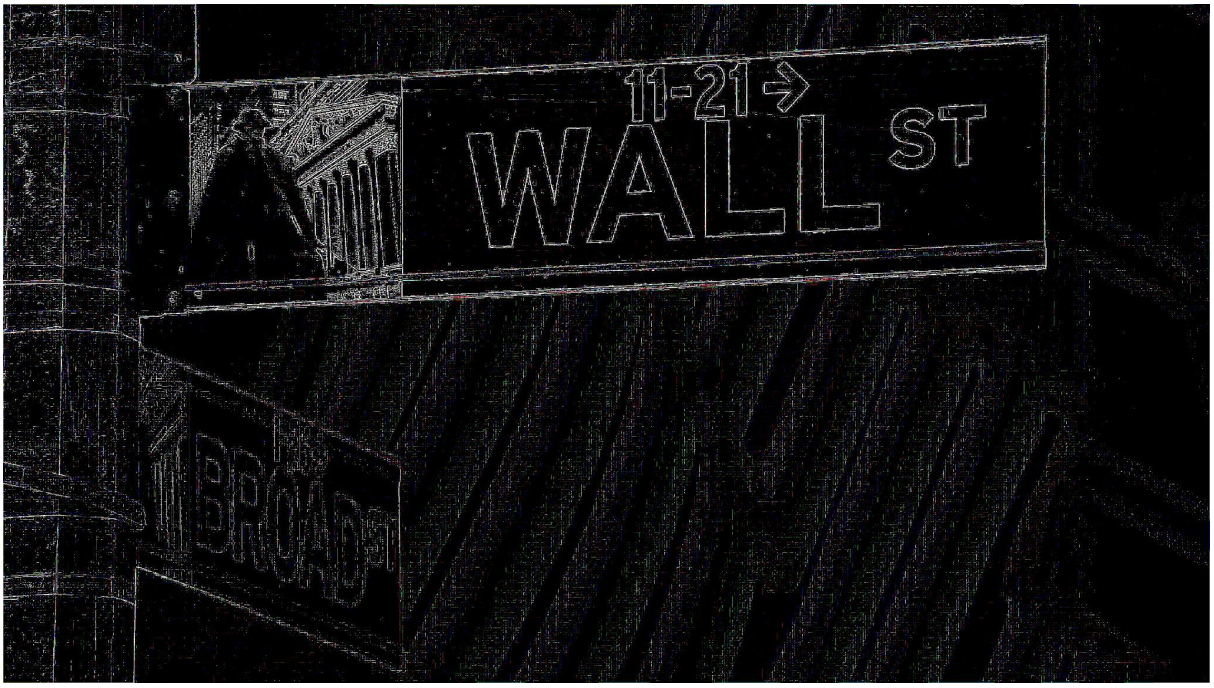
- Nous avons utilisé std::chrono et les cudaEvent pour mesurer le temps d'exécution de la partie cpu et gpu, respectivement.
- Nous avons géré les erreurs des fonctions Cuda et Kernel via une fonction "handleError"
- Nous avons utilisé des streams, leur nombre peut être aisément modifié : il est demandé en paramètre de la fonction execKernelGpu
- Nous avons utilisé la mémoire partagée, notamment pour l'accès aux variables de la matrice kernel : chaque threads a autant d'accès au kernel que la taille de celui-ci, donc si on a un kernel 9x9, on aura 81 accès par threads.
- Nous avons utilisé des blocs 2D, par contre notre Grille de bloc est à une seule dimension (voir difficultés rencontrées).
- Nous avons comparé nos temps d'exécution CPU / GPU sur différentes Images et différentes Configuration (voir ci - dessous)

Difficultés rencontrées

Nous n'avons pas fait de grille 2D car nous n'avons pas réussi à décider de la taille de celle-ci. Nous avions pensé à faire $\sqrt{inputsize}$ * $\sqrt{inputsize}$ mais le fait que la racine carrée ne soit pas tout le temps une valeur entière faisait qu'il manquait certains blocs dans la plupart des situations. Nous pensions faire cela car nous avons besoin d'autants de blocs que nécessaire pour couvrir les octets gérée par le stream courant, c'est à dire : nombre d'octets totales divisé par le nombre de stream, divisé par le nombre de thread par bloc (généralement 1024). Cela nous donne un certain nombre de blocs à utiliser pour le stream courant, qui peut ne pas être possible à factoriser.

Une autre difficulté rencontrée aura été de maîtriser la position de l'octet géré par le thread courant dans la fonction gpu, cela nous faisait parfois de bonnes surprises.

Exemple de résultat



Edge détection avec image de taille : 3840x2160

Temps d'exécution CPU : 601 ms
Temps d'exécution GPU : 154 ms

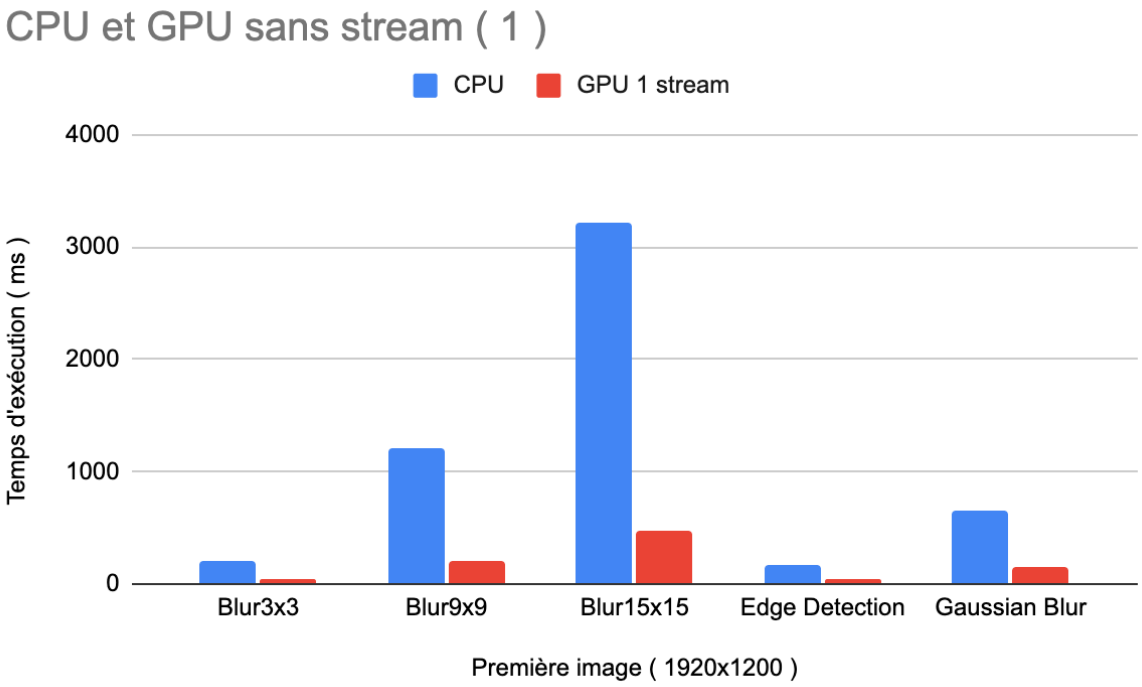
Image originale :

Comparaison des temps d'exécution

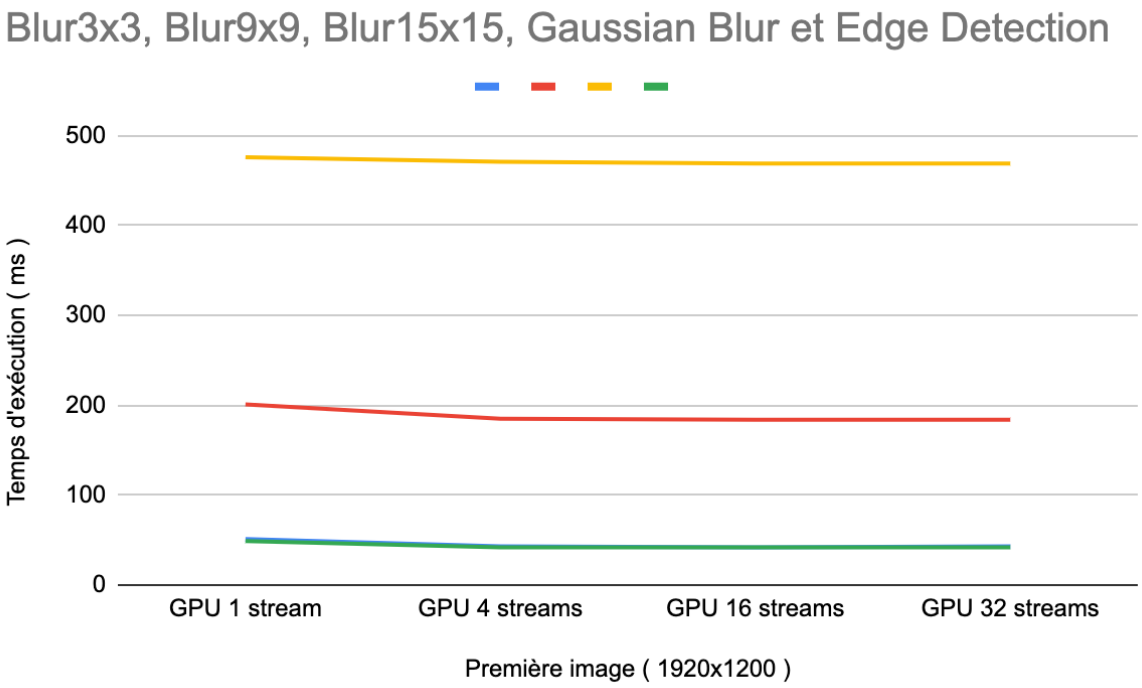
Les tests GPU ont été réalisé avec une carte 940mx qui a la configuration suivante :

- maxBlocksPerMultiProcessor 32
- maxThreadsPerBlock 1024
- warpSize 32
- multiProcessorCount 3
- maxThreadsPerMultiProcessor 2048

Pour une image en 1920x1200 :

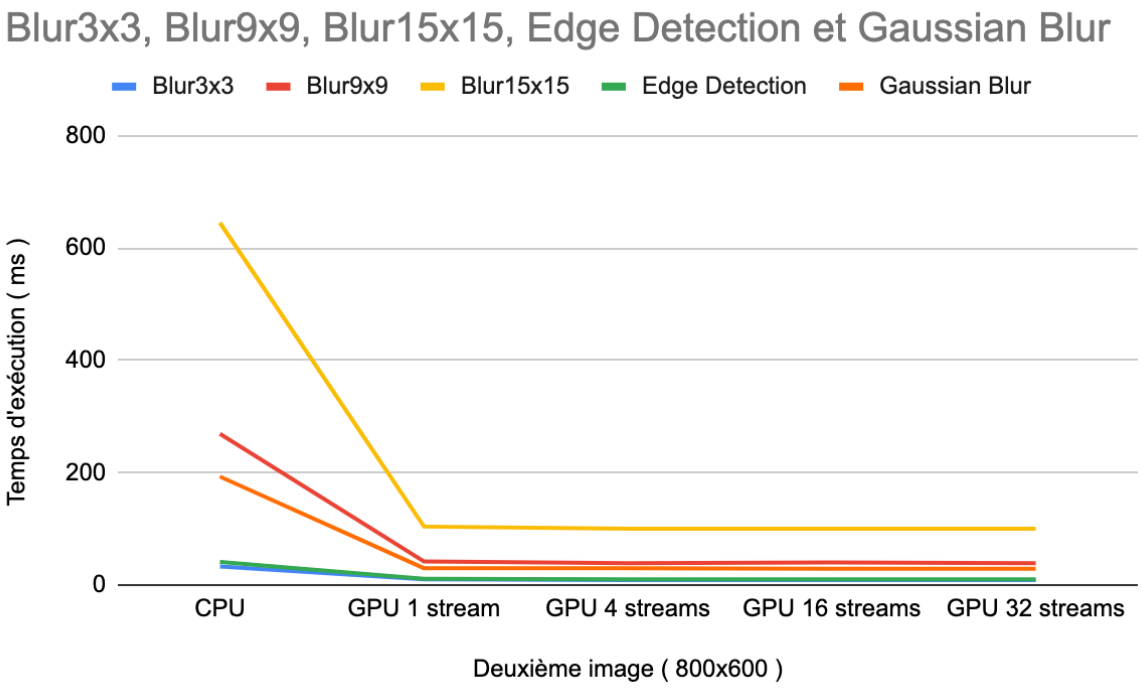


Comparaison CPU vs GPU pour une image de taille moyenne



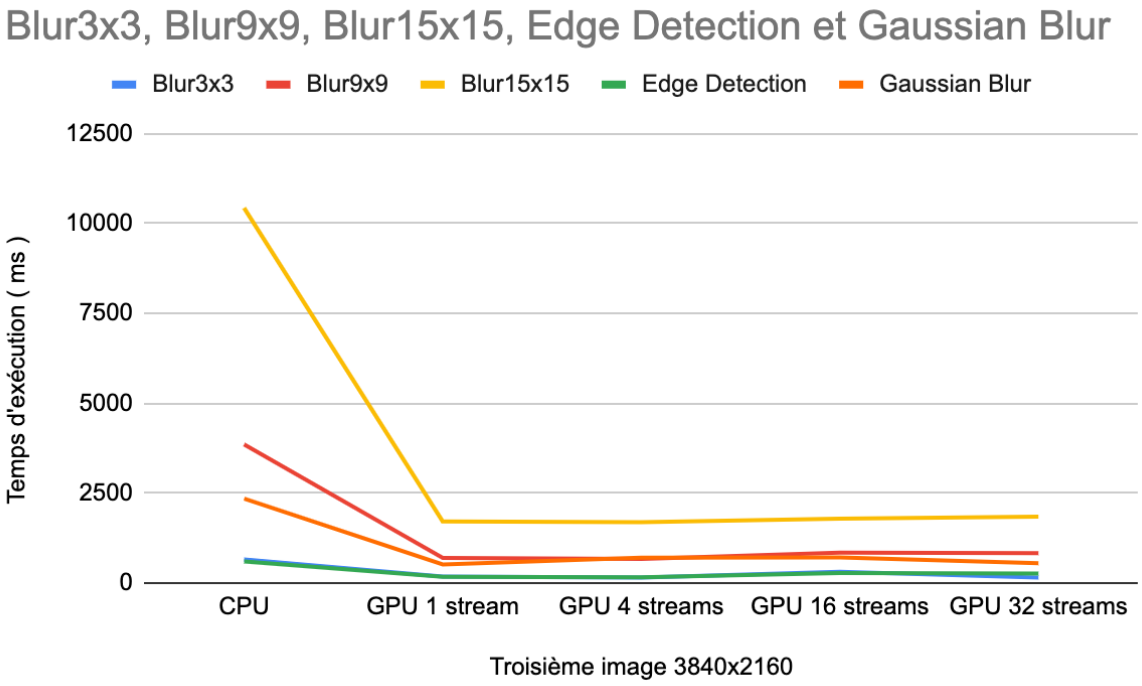
Comparaison de l'exécution GPU avec différents streams, on remarque une baisse de quelques ms entre la version avec un seul stream et la version à 4 streams. La courbe verte et bleue sont au même niveau.

Pour la deuxième image : 800x600

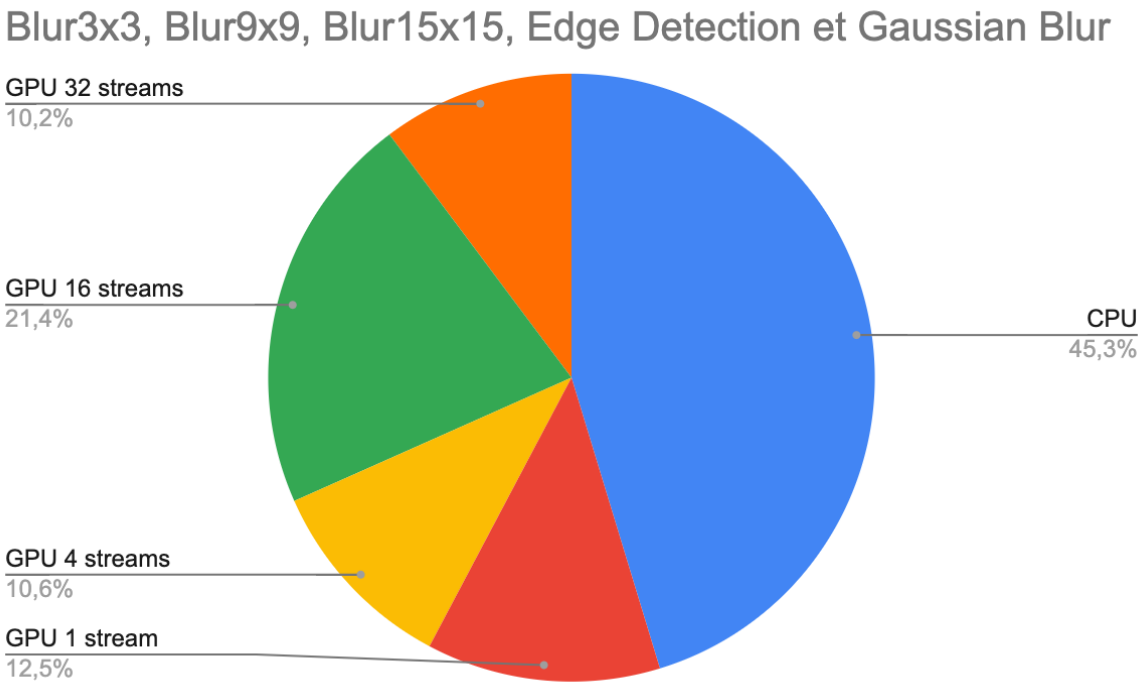


Pour la deuxième image, pas de différence notable avec la première, on gagne quelques millisecondes entre la version sans stream et la version avec.

Pour la troisième image : 3840x2160



Pour cette troisième image, on remarque une légère augmentation entre la version 4 streams et la version 16 streams, nous expliquons cela par une augmentation de la charge de travail sur l'ordinateur qui réalisait les différents tests.



On peut un peu mieux se rendre compte de cela en comparant le temps total passé entre les différentes versions : la version 16 streams a passé environ 2 fois plus de temps que la version 4 streams.