Rapport BE CUDA ACCOU Martin LALONNIER Lucas

Partie 1

Q 1.1

On doit être vigilant à ce qu'il se passe au bord et donc choisir `numBlocks = (n+BS-1)/BS`.

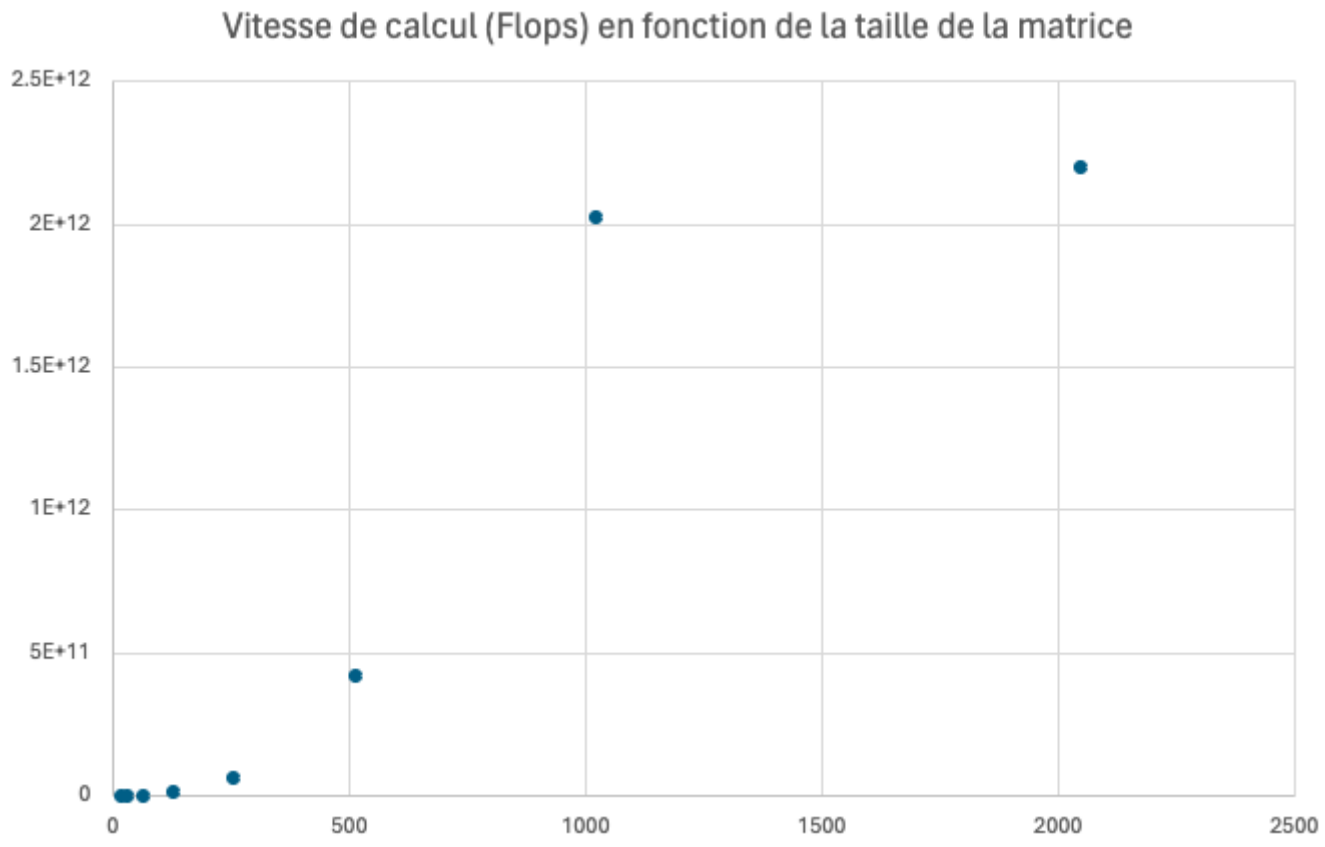
Q 1.6

On a fixé le nombre d'itérations à 1 et BS à 16.

Pour chaque élément de la matrice, on a n multiplications et n additions à effectuer, soit $2n$ opérations à réaliser par élément de la matrice.

Or, on a n^{2} éléments dans une matrice.

On a donc $2n^{3}$ opérations pour réaliser le produit matriciel.



L'utilisation de GPUs est donc surtout justifiée pour de grandes valeurs de tailles de matrices.

Partie 2

Q 2.1

Cet algorithme permet d'améliorer le temps de calcul en exploitant les mémoires partagées de chaque bloc.

En effet, on a d'un côté (pour notre première implémentation) un nombre d'accès à la mémoire globale égal à $2n^{3}$.

De l'autre côté, avec la version mémoire partagée, le nombre d'accès à la mémoire globale est égal à $NombreDeBlocs * NombreAccèsParBloc$ soit $n^{2}/BS^{2} * 2*BS*n$ soit $2n^{3}/BS$.

On réduit donc d'un facteur BS le nombre d'accès à la mémoire globale, ce qui est souhaitable car on sait que l'accès à la mémoire partagée au sein d'un blocs est bien plus rapide.

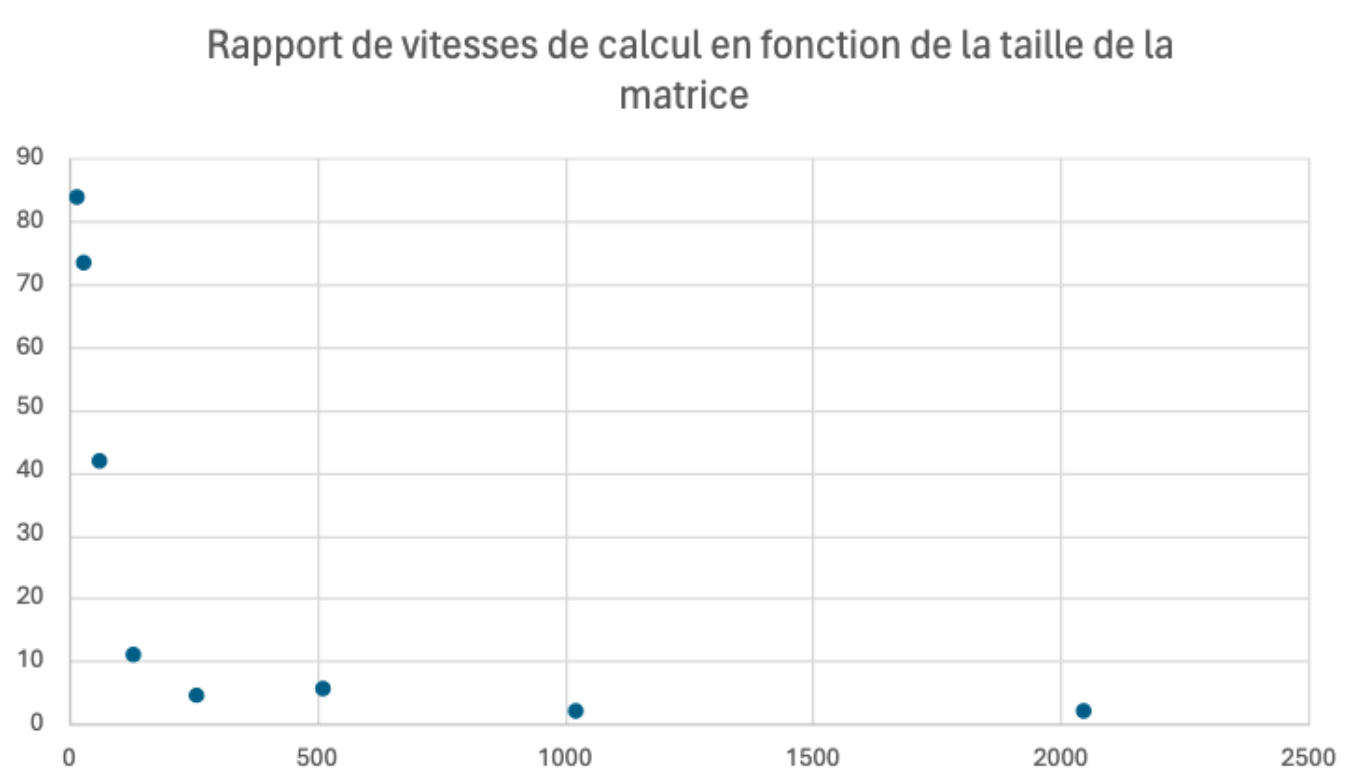
Q 2.2

On implémente l'algorithme proposé dans le cours.

L'utilisation de `__syncthreads()` permet de s'assurer que tous les threads ont terminé les instructions précédentes avant de réaliser les suivantes. C'est utile ici pour s'assurer que la mémoire partagée est bien écrite avant de pouvoir y accéder.

Q 2.3

On trace le rapport entre la vitesse de calcul (en Flops) avec la version mémoire partagée et la vitesse de calcul avec la version précédente.



On remarque que l'utilisation de la mémoire partagée permet d'améliorer la vitesse de calcul. Le rapport semble stationner à 2 pour de grandes tailles de matrices.

Q 2.4

On obtient les résultats suivants:

Taille de blocs	Temps d'éxécution (s)
8	0.008449
16	0.003978
32	0.004916

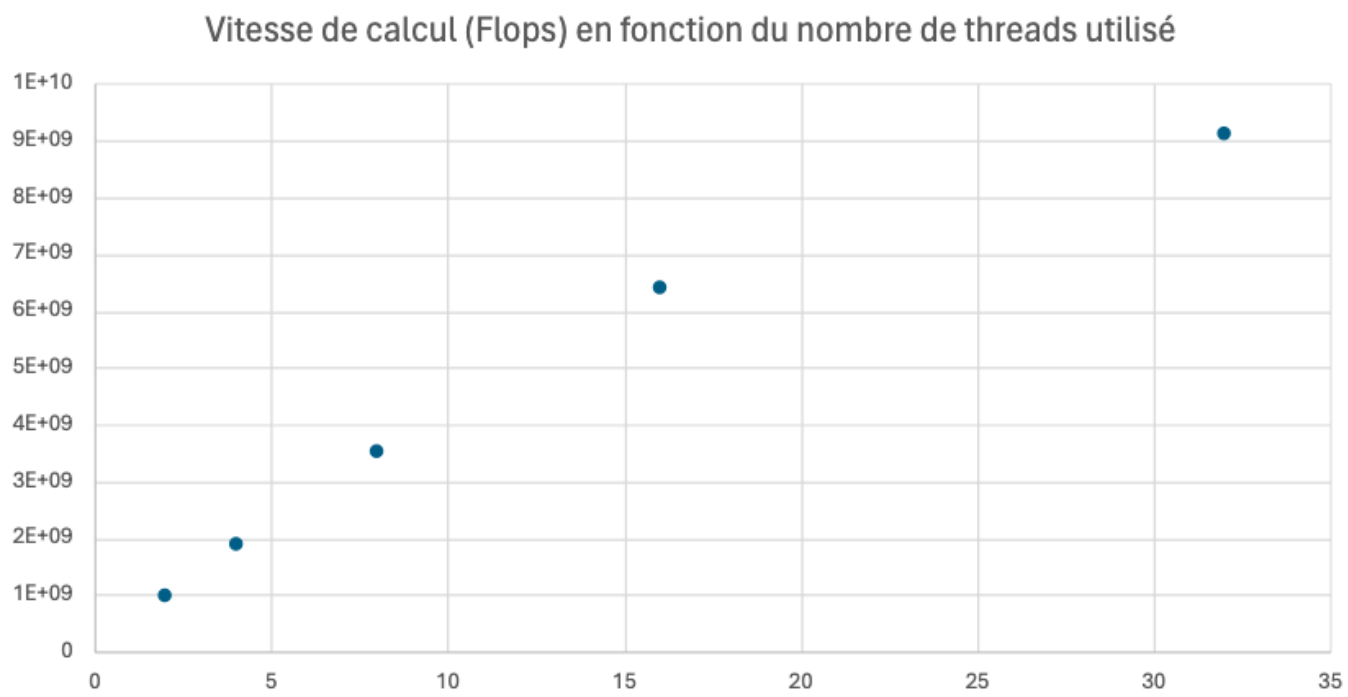
On ne peut pas prendre une taille de blocs de 64 car on dépasse alors le nombre maximal de threads par bloc.

Les meilleures performances sont obtenues pour une taille de blocs égale à 16.

Q 2.5

On utilise ici une clause `omp for` avec l'option `collapse` pour paralléliser le calcul de chaque élément de la matrice.

On fixe aussi $n=2048$.



La vitesse de calculs est bien corrélée au nombre de threads utilisé.

Ce qui est surtout important à remarquer est l'ordre de grandeur (3) obtenu par l'utilisation d'un GPU sur la vitesse de calcul.