Projets CUDA 2021

Les projets sont à réaliser par groupes de 2 étudiants (un groupe de 3 si nombre impair dans le groupe).

Vous avez le choix entre les 4 sujets ci-dessous avec des difficultés variables.
Pour chaque projet il faudra créer une implantation CPU de référence et comparer les résultats et performances.

Il est demandé de faire une implantation correcte et ensuite d'envisager des optimisations sur le GPU pour améliorer encore les performances (mémoire shared, streams, etc).
Des codes sources sur CPU sont fournis pour ces différents projets.

Le support pour les projets se fera via le salon M1 cm-prog-graphique-cuda sur Discord.
Il ne faut pas hésiter à poser vos questions et à échanger des informations sur vos problèmes et solutions entre groupes pour résoudre vos problèmes rapidement.

Chaque groupe doit mettre son projet sur un dépôt Github/Gitlab et m'ajouter en tant que « Reporter » pour me permettre d'accéder au code (mon login sur les 2 plateformes : sjubertie).

Un rapport de quelques pages présentant l'algorithme implanté, les optimisations choisies, les difficultés rencontrées, et les résultats obtenus devra accompagner le projet.
L'évaluation tiendra compte de la difficulté du projet, de la qualité du code, des optimisations utilisées, des justifications et de l'analyse des résultats.
La date de rendu est fixée habituellement à la date de l'examen du module.

1 Traitement d'images

Implanter un traitement d'images sur GPU parmi les exemples proposés dans les liens suivants :
http://gmic.eu/gallery/arrays.html#menu
http://www.cimg.eu/
https://docs.opencv.org/master/d7/da8/tutorial_table_of_content_imgproc.html
https://en.wikipedia.org/wiki/Digital_image_processing
https://docs.gimp.org/2.10/fr/gimp-filter-convolution-matrix.html

Il peut s'agir de :
- redimensionner une image
- effectuer une rotation sur une image
- appliquer un flou anisotropique, Gaussien, etc
- du débruitage
- de la détection de contours
- calculer l'histogramme d'une image
- etc.

La majorité de ces algorithmes consistent à interpoler des valeurs ou à considérer le voisinage de chaque pixel (stencil/convolution) comme dans l'exemple de la détection de contour de Sobel.

2 Transformée de Fourier rapide (1D)

La transformée de Fourier rapide (FFT : Fast Fourier Transform) est très utilisée dans le traitement du signal pour passer du domaine temporel au domaine fréquentiel. L'objectif est de l'implanter en CUDA et de comparer cette implantation à celle de référence sur GPU : cuFFT :
https://fr.wikipedia.org/wiki/Transformation_de_Fourier_rapide
https://developer.nvidia.com/cufft

L'objectif n'est pas d'aller plus vite que l'implantation de Nvidia mais de tester différentes optimisations et voir l'effet sur les performances.

L'algorithme de la FFT est de complexité log(n) et consiste principalement en des multiplications de nombres complexes et à échanger des données suivant un schéma dit « butterfly ». Les optimisations à considérer sont principalement l'utilisation de la mémoire shared, et la synchronisation des blocs.

3 Simulation : Différences finies – Stencils (2D-3D)

Les stencils ou convolution sont très utilisés pour le traitement d'images ou dans les simulations (équation de la chaleur, dynamique des fluides, etc).
Le principe de base est de discrétiser un domaine sous forme de grille 2D ou 3D, de parcourir ce domaine, et de déterminer pour chaque point sa nouvelle en fonction du point et de ses voisins. Ce calcul est généralement donné par une matrice de convolution qui détermine les voisins considérés ainsi que les coefficients à leur appliquer.
Par exemple, la matrice suivante indique que pour déterminer la valeur du point il faut considérer les voisins au-dessus, au-dessous, à gauche et à droite et les multiplier par 1/8 et ajouter ces valeurs à 1/4 de la valeur du point courant :

	1/8	
1/8	1/4	1/8
	1/8	

Ce qui se traduit par le code C++ suivant (version 2D) :

```
for( std::size_t j = 1 ; j < h - 1 ; ++j )
{
  for( std::size_t i = 1 ; i < w – 1 ; ++i )
  {
    out[ j*w + i ] = 0.25f * in[ j*w+i ]
                   + 0.125f * ( in[ (j-1)*w+i ] + in[ (j+1)*w+i ] + in[ j*w+i-1  ] + in[ j*w+i+1 ] );
  }
}
```

Évidemment, la forme du stencil peut varier en fonction du problème considéré : 3D, plus de voisins (diagonales, plus large) et coefficients différents.

Bibliographie :
https://www.researchgate.net/publication/317487457_A_Multi-level_Optimization_Strategy_to_Improve_the_Performance_of_Stencil_Computation

4 Simulation : Éléments finis spectraux

La méthode des éléments finis est utilisée pour étudier la propagation d'ondes sismiques, ou la résistance de matériaux. Elle consiste à discrétiser un domaine en éléments de forme hexaédrique (cube) composés de 5x5x5 points (à l'ordre 4) et à calculer des forces à l'intérieur de chaque hexaèdre. Les points sur les faces d'un élément sont partagés avec les éléments voisins. Cette présentation laisse penser que le code est complexe mais en fait il se résume en 3 étapes :

1. Constitution de l'élément
2. Calcul des forces internes
3. Assemblage des accélérations

Une version du code écrite en C est disponible. Deux approches pour le portage sur GPU sont possibles. Le point le plus important à considérer est le portage sur GPU, les optimisations sont à considérer dans un second temps.

Bibliographie :

https://www.researchgate.net/publication/323232151_Vectorization_of_a_spectral_finite-element_numerical_kernel