Mini-Projet : exploitation simultanée des processeurs et de l'accélérateur GPU

L'objectif de ce mini-projet est d'élaborer un programme implémentant un code stencil 2D (exactement comme celui vu au TP précédent) capable d'utiliser à la fois l'accélérateur GPU mais aussi les processeurs de la machine.
Vous trouverez des ressources utiles dans le répertoire `/net/cremi/rnamyst/etudiants/opencl/Projet_OpenCL/`.
Par ailleurs, le manuel de référence OpenCL se trouve à cet endroit :

`/net/cremi/rnamyst/etudiants/opencl/Doc/opencl-1.2.pdf`

1 Préambule

Pour réaliser ce mini-projet, vous partirez du code de Stencil2D (même une version naïve du code OpenCL conviendra dans un premier temps) étudié au TP précédent.
À la fin du projet, vous fabriquerez une archive contenant les sources de votre programme ainsi qu'un petit rapport (au format PDF) expliquant de manière concise les problèmes rencontrés et les solutions trouvées, et récapitulant les performances observées à différents stades d'avancement.
L'archive sera de préférence au format TAR compressé et aura un nom formé des noms des binômes et comportant une clé secrète. Par exemple :

`durand-dupont-AbRaCaDaBrA33.tar.gz`.

L'archive devra être déposée dans le répertoire verb+/net/cremi/rnamyst/etudiants/depot+ (répertoire dans lequel vous pouvez écrire mais pas lire).

2 Principe

Pour occuper simultanément les processeurs et la carte graphique, une idée simple est de couper la grille 2D en deux bande horizontales, et affecter l'une des deux à la carte graphique tandis que l'autre sera calculée par les processeurs.
Ajoutez des macros dans le programme permettant de jouer sur le nombre de lignes prises en charge par la carte graphique (et donc par effet de bord sur le nombre de lignes prises en charge par les processeurs). Par exemple :

```
#define YDIM      4096
#define YDIM_GPU  2046
#define YDIM_CPU  (YDIM-YDIM_GPU)
```

Vous ajusterez finement ces valeurs plus tard...
Pour l'instant, on ne cherche pas à optimiser le code s'exécutant sur les processeurs, ni même à le paralléliser. Vous pouvez donc utiliser la fonction séquentielle `stencil` pour calculer la partie séquentielle.
On continuera à utiliser `h_idata` et `h_odata` pour stocker la grille complète en début et fin de programme.
On s'arrangera pour que les calculs sur processeurs travaillent directement sur les premières lignes de ces matrices, et on tranférera sur GPU uniquement la partie « basse » de ces matrices.
Durant une itération, il faut que les processeurs calculent sur le morceau de grille dont ils sont responsables *pendant que* la carte graphique fait de même avec l'autre morceau. Assurez-vous simplement que les calculs soient bien réalisés simultanément à ce stade.

Vérifiez également que le résultat final, une fois le morceau calculé par le GPU rappatrié en mémoire principale, est correct.

3 Plusieurs itérations

Il s'agit à présent d'ajouter une boucle pour réaliser plusieurs itérations.
Modifiez donc le code pour effectuer plusieurs itérations, en vous inspirant de la même façon dont procède le programme `Heat` : alternance entre itérations impaires (lecture de `d_idata` et écriture de `d_odata`) et itérations paires (lecture de `d_odata` et écriture de `d_idata`). Attention, à la fin de ce nombre pair d'itérations, le résultat se trouve dans `h_idata` du côté des CPU...
À ce stade, le résultat final est faux car il n'y a pas propagation des valeurs aux frontières entre les deux sous-grilles. C'est l'objet de la question suivante.

4 Propagation des bords

Pour obtenir un calcul correct de part et d'autre, il faut propager, entre chaque itération, les valeurs limitrophes à la séparation entre les deux sous-grilles.
En supposant que les processeurs calculent la sous-grille du haut et le GPU la sous-grille du bas, cela signifie qu'entre chaque itération, le programme principal envoie la ligne du bas de la sous-grille du haut vers le GPU (en tant que nouveau bord du haut) et qu'il récupère la ligne du haut depuis le GPU pour en faire le nouveau bord en bas de la sous-grille calculée par les processeurs.
Pour effectuer ces copies de « bords », qui écrivent/lisent uniquement une partie de la matrice stockée sur le GPU, vous pouvez vous inpirer du programme `subbuffer` suivant qui illustre l'utilisation de la routine `clEnqueueWriteBuffer` avec un *offset* permettant d'écrire seulement sur une partie du tampon.
Faites varier la taille de la sous-grille traitée par le GPU pour aboutir à des exécutions de même durée de chaque côté.

5 Version CPU parallèle

Parallélisez la version CPU en utilisant, au choix, OpenMP ou OpenCL. Normalement, cette parallélisation ne doit pas avoir d'impact sur le code en charge des échanges avec le GPU.
Évaluez les performances obtenues en nombres d'opérations flottantes par seconde (i.e. Gflop/s). N'hésitez pas à augmenter la taille du problème pour améliorer les performances de la partie graphique.