# Parallélisme

—

# TP6

**Ce travail est individuel et vous avez jusqu'au 13 Janvier 2019 pour le réaliser.**

## 1 Calcul de l'ensemble de Julia sur GPU

Les GPUs (*Graphics Processing Units*) sont des processeurs conçus à l'origine pour le traitement graphique. Ils possèdent un modèle de programmation semblable au SIMD, où de nombreux flux d'exécutions (threads) effectuant les mêmes opérations sur des données différentes vont s'exécuter en parallèle sur de multiples coeurs d'exécutions simples.

Dans ce TP, vous allez implémenter le calcul de l'ensemble de Julia sur GPU avec la technologie CUDA, permettant d'exécuter du code sur les GPUs Nvidia.

## 2 Implémentation

Un programme CUDA se sépare en deux parties. Le code s'exécutant sur le host (CPU) et celui s'exécutant sur le device (GPU). La partie s'exécutant sur le device est appelé code kernel et ne peut pas s'exécuter de façon autonome.

C'est le code host qui donne les ordres d'exécution de kernel et de transfert mémoire sur le device. Vous devez donc écrire un code kernel qui calcule le nombre d'itération avant divergence d'un point donné du plan complexe et stocke le résultat dans une matrice (voir le TP précédent pour les détails sur le calcul de l'ensemble de Julia). Le code host doit préparer la configuration d'exécution (nombre et disposition des threads) et demander l'exécution du kernel. Une fois le kernel exécuté, le host doit récupérer les données et les écrire dans un fichier au format PGM.

Il vous est fortement recommandé de vous inspirer du code effectuant l'addition d'une constante à une matrice. Ce programme possède en effet une structure très proche de ce qu'il vous est demandé d'implémenter (allocation d'une matrice sur le GPU, écriture d'une valeur dans cette matrice depuis le GPU, récupération de la matrice après exécution du kernel).

## 3 Rendu

En plus du code source, vous devez rendre un court rapport présentant le fonctionnement de votre programme et présentant quelques mesures de performances. Vous devez montrer le temps d'exécution de votre programme sur GPU et le comparer au temps d'exécution des versions parallèle du TP précédent. Pensez à mesurer le temps de copie de données ainsi que le temps d'exécution du kernel. L'exécution d'un kernel étant asynchrone depuis le host, pensez à synchroniser le device avec le host avant de terminer la mesure du temps d'exécution.

Votre programme peut schématiquement ressembler à ceci :

```
startCompute = time
kernel <<<dimGrid , dimBlock>>>
deviceSynchronize
endCompute = time
startCopy = time
copyDeviceToHost
endCopy = time
```

# 4 Consigne

- Vous devez rendre votre code et votre rapport dans une archive au format zip portant votre nom d'utilisateur qui doit être structurée de la façon suivante :
  - un répertoire src qui contient vos fichiers sources
  - un fichier Makefile à la racine de l'archive
  - le rapport à la racine de l'archive
- Le fichier Makefile doit contenir au moins les deux cibles suivantes :
  - la cible all compile vos applications dans le répertoire courant et produit l'exécutable nommé julia_cuda
  - la cible clean supprime les fichiers compilés
- Vos applications doivent prendre les arguments suivants, dans l'ordre : point inférieur gauche, point suppérieur droit, c, nombre maximum d'itérations, taille du domaine, nom du fichier de sortie. Vous pouvez prendre exemple sur le fichier julia.cpp sur moodle (répertoire TP5).
- Votre programme doit produire un résultat au format PGM
- L'archive produite doit être déposée sur moodle dans Parallélisme → TP6