TP2

**Objectifs :** Concepts avancés

**Ressources :**

* [*http://docs.nvidia.com/cuda/index.html*](http://docs.nvidia.com/cuda/index.html)
* [*https://docs.nvidia.com/cuda/pdf/CUDA\_C\_Programming\_Guide.pdf*](https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf)

**Setup :**

* haswell-cuda ou 192.168.80.64
* Login : nom.prenom (minuscule)
* Passwd : toto (à changer)
* Vérification bashrc, PATH et LD\_LIBRARY\_PATH

**Depot Git :**

* <https://github.com/hpc-apps/Cuda.git>

**Exercice 1**

1. Créer un code permettant de récupérer le nombre de carte graphique disponible en utilisant ***cudaGetDeviceCount()***
2. *En utilisant* ***cudaGetDeviceProperties()*** *récupérer :*

* le nombre maximum de threads par bloc,
* les dimensions maximales pour les blocs et les grilles,
* le nombre de multiprocesseurs.

**Exercice 2**

1. Reprendre l’exemple d’addition de vecteur du TP1 et l’adapter pour que le nombre de blocs et de threads par bloc soit déterminé automatiquement à partir de la taille des données.
2. Optimiser la décomposition en fonction de la taille des données et les caractéristiques du GPU.

**Exercice 3**

* Modifier l’exemple d’addition de vecteurs afin de d’utiliser une version utilisant les **streams** CUDA
  + Chaque **stream** opère sur une sous-partie du vecteur (e.g deux streams).
  + Utiliser les fonctions *cudaMallocHost, cudaStreamCreate* + destructeurs et *cudaMemcpyAsync*

**Exercice 4**

1. Utiliser le squelette de programme proposé afin de créer deux matrices de taille N x N et de les additionner en utilisant des blocs 2D de threads.
2. Optimiser en fonction de la taille des données ainsi que des caractéristiques du GPU.