

# TP1

**Objectifs :** Base de la programmation CUDA

**Ressources :**

- <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)

**Git :**

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

## Exercice 1

- Utilisez l'utilitaire *deviceQuery* afin d'identifier les « Compute Capabilities » .

## Exercice 2

- Testez les exemples du répertoire Ex2
  - *Compilation : nvcc -o exemple exemple.cu -std=c++11*
- Modifiez la taille des tableaux, ceux-ci ne doivent pas dépasser 1024

## Exercice 3

- Créer un programme CUDA qui pour chaque valeur d'un tableau multiplie par 2 cette valeur si elle est paire.

## Exercice 4

- Créer un programme qui initialise deux vecteurs sur le CPU et effectue l'addition de ces vecteurs sur le GPU.

## Exercice 4 – bis (option)

- Créer un programme qui initialise deux tableaux sur le CPU et réalise l'échange des valeurs entre ces deux tableaux sur le GPU (swap).

## Exercice 5

- Les exemples précédents ne gèrent pas les erreurs.
- Les fonctions *cudaMalloc*, *cudaMemcpy*, . . . retournent une valeur de type *cudaError\_t* qui permet de vérifier si l'opération s'est déroulée correctement
  - Rajouter la gestion des erreurs pour le code qui correspond à l'addition de deux vecteurs.
- Produire les erreurs suivantes et vérifier qu'elles sont correctement gérées :
  - Nombre de threads supérieur à 1024 pour le lancement du kernel.
  - Inverser le sens de communication pour les appels à *cudaMemcpy* ( *HostToDevice*, *DeviceToHost* ).

## Exercice 6

- L'API CUDA fournit donc des outils spécifiques pour mesurer le temps d'exécution (*CUDA runtime API/Event management*).
- Instrumenter le code et mesurer les temps d'exécution du kernel pour l'addition de deux vecteurs de taille 1024.