# CUDA TP1
# Introduction à CUDA

L'objectif de ce premier TP est de poser les bases de la programmation CUDA. Pour l'instant il n'est pas question de performance.

Ressources :

- Supports pour le module : `https://github.com/sjubertie/teaching-CUDA`
- API CUDA : `http://docs.nvidia.com/cuda/index.html`

## 1 Premiers pas

Tester les exemples basiques disponibles à l'adresse `https://github.com/sjubertie/teaching-CUDA/tree/master/examples/01-basics`. Pour compiler `nvcc -o exemple exemple.cu -std=c++11`.

Modifier la taille des tableaux. Attention, la taille des tableaux ne peux dépasser 1024 pour l'instant car un seul bloc est utilisé et un bloc ne peut contenir que 1024 threads.

## 2 1 sur 2

Créer un programme CUDA qui extrait un élément sur deux d'un tableau et le place dans un tableau de sortie 2 fois plus petit. Plusieurs implantations du kernel sont envisageables:

- Les threads pairs recopient les éléments aux positions paires, les threads impairs ne font rien.
- Chaque thread $tid$ récupère l'élément à la position $2 \times tid$ et le place à la position $tid$ du tableau de sortie.

## 3 Pair ou impair

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

## 4 Gestion des erreurs

Les exemples précédents ne gère pas les erreurs éventuelles d'allocation ou d'exécution sur le GPU. Les fonctions `cudaMalloc`, `cudaMemcpy`, ... retournent une valeur de type `cudaError_t` qui permet de vérifier si l'opération s'est déroulée correctement. Consulter la documentation CUDA, `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#error-checking`, pour obtenir les fonctions permettant de gérer les erreurs et ajouter la gestion des erreurs pour les codes précédents. 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 ).
- Pointeurs incorrects pour les `cudaMalloc`.
- ...

# 5 Mesures

Les appels aux fonctions CUDA par le CPU sont réalisés de manière asynchrones : les appels sont envoyés par le CPU au pilote de la carte graphique qui ajoute ces appels à une file d'attente, et le CPU continue d'exécuter son programme. Il n'est donc pas possible de mesurer le temps d'exécution d'une fonction CUDA avec des outils de mesure CPU classiques (`gettimeofday`, `std::chrono`, ...).

L'API CUDA fournit donc des outils spécifiques pour mesurer le temps d'exécution (CUDA runtime API/Event management). Un exemple est fourni ici : `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#events`.

Instrumenter l'exemple `vecAdd` et mesurer les temps de transfert ainsi que le temps d'exécution du kernel pour un tableau de 1024 valeurs.