TD GPGPU n°1

GPGPU Courses
MII 2 option Sciences de l'Image et IMAC 3 option Prog 3D

Premiers pas en Cuda

Initiation à la programmation parallèle sur processeurs graphiques (GPU) en utilisant CUDA.

CUDA (Compute Unified Device Architecture) est une technologie propriétaire, proposée par NVidia, qui permet d'exploiter la puissance de calcul massivement parallèle intrinsèque aux GPUs.

Commencez par télécharger les sources sur la page de elearning. Tout **devrait** fonctionner sur les machines de la salle 1B110, sous Linux ou Windows. Vous pouvez bien entendu utiliser vos machines personnelles sous l'OS de votre choix. Cependant, assurez-vous qu'elles soient bien configurées et n'attendez pas d'aide de votre chargé de TD pour ce faire (désolé trop chronophage).

Exercice 1 : Un matériel à vérifier

Rien à coder dans cet exercice ! Compilez et exécutez le programme fourni. Il illustre comment récupérer les différentes propriétés des devices pouvant utiliser CUDA sur la machine.

Exercice 2 : Additionner deux (gros) vecteurs

Vous devez paralléliser sur GPU le calcul de la somme de deux vecteurs. **Vous ne devez coder que dans les fichiers** *student*. Utilisez la classe *ChronoGPU* pour mesurer les temps d'allocation, de transferts de données et de calcul sur GPU.

1. Complétez la fonction *studentJob* pour :
   - Allouer la mémoire sur le device
   - Transférer les données depuis l'host vers le device
   - Lancer le kernel en utilisant **1 block de 256 threads**
   - Transférer le résultat depuis le device vers l'host
   - Libérer la mémoire sur le device
2. Implémentez le kernel sumArraysCUDA. Exécutez le code pour vérifier son fonctionnement.
3. Jusqu'ici la taille du vecteur était de 256. Le programme prend en argument la taille du vecteur (-n <taille>). Testez votre code avec (-n 512). Pourquoi le programme ne fonctionne-t-il plus ? Modifiez le nombre de threads donné à l'appel de votre kernel pour résoudre le problème.
4. Essayez maintenant avec (-n 4096). La solution précédente n'est plus applicable. Modifiez l'appel de votre kernel pour résoudre le problème. Déduisez une méthode pour trouver le nombre de blocks nécessaire pour un nombre de threads donné.
5. Testez maintenant avec des tailles supérieures. À partir d'un moment, le programme ne fonctionne plus, pourquoi ? Modifiez votre kernel pour résoudre le problème.

   Indice : lorsque vous codez habituellement, l'exécution se fait sur un seul thread. Comment faites-vous pour traiter toutes les données d'un tableau ?
6. Comparez et analysez les temps pour différentes tailles de vecteurs et avec des nombres de blocks et de threads différents

Exercice 3 : Lena est tellement mieux en Sépia !

Dans cet exercice, vous allez devoir paralléliser un algorithme qui applique un filtre sur une image afin d'en produire une nouvelle ayant un aspect sépia. **Vous ne devez coder que dans les fichiers student.**

L'image source est passée en argument du programme à l'aide de l'option `-f ../images/Lena.png`. Vous testerez votre programme avec les autres images fournies et n'hésitez pas à essayer avec d'autres.

L'algorithme de filtre est simple et consiste à modifier la couleur de chaque pixel tel que :

$$outRed = min(255, (inRed * 0.393 + inGreen * 0.769 + inBlue * 0.189));$$
$$outGreen = min(255, (inRed * 0.349 + inGreen * 0.686 + inBlue * 0.168));$$
$$outBlue = min(255, (inRed * 0.272 + inGreen * 0.534 + inBlue * 0.131));$$

Les TD ont été écrit très majoritairement par Maxime Maria (maintenant Maître de Conférences au XLIM à Limoges)


1. Complétez la fonction studentJob (allouez/transférez/libérez/etc.). Cette fois-ci, vous devez configurer la répartition des threads sur une grille 2D. Pensez à utiliser la classe ChronoGPU pour mesurer les temps.
2. Déclarez et implémentez un kernel appliquant le filtre à une image. Testez son fonctionnement.
3. Comparez et analysez les temps pour différentes images et avec des nombres de blocks et de threads différents.
4. Inversez les boucles (largeur/hauteur) dans votre kernel et comparez les résultats. D'après vous, pourquoi cette différence ? Le pourquoi du comment vous sera donné à la vidéo 97 !

Exercice 4 : J'aime les maths (et j'additionne des matrices)

Dans cet exercice, pas de code fourni !

1. Réalisez un programme qui :
   - Prend deux arguments : une largeur L et une hauteur H
   - Créé et initialise aléatoirement deux matrices de taille L×H
   - Calcule la somme des deux matrices sur CPU
   - Calcule la somme des deux matrices sur GPU avec CUDA
   - Compare les résultats pour s'assurer de leur validité
2. Comparez et analysez les temps pour différentes tailles de matrice et avec des nombres de blocks et de threads différents.