TD GPGPU n°2

Cours de GPGPU
MII 2 option SIS / IMAC 3ème année

GPGPU grâce à Cuda

Utilisation de l'extension Cuda pour des calculs parallèles

Le but de ce TD est de réaliser dans un premier temps des calculs d'algèbre linéaire très simple pour expérimenter la programmation parallèle. Il est important de noter que lorsque l'on souhaite profiter de l'accélération qu'offre Cuda, il faut pouvoir amortir les temps de transfert de mémoire CPU – GPU. Ainsi Cuda devra s'employer plutôt pour des calculs coûteux et qui se parallélisent bien, en prenant soin de laisser tant que faire se peut la mémoire sur la carte graphique.

Ressource :

Du code source vous est fourni sur le site internet http://www-igm.univ-mlv.fr/~biri/. Vous y trouverez l'application cuda_blank qui sert de « canevas » aux développements que l'on vous demandera. Les fichiers sont séparés en un « main.cpp » qui est le programme principal, et deux fichiers « blank.cu » et « blankKernel.cu » qui sont respectivement, un fichier permettant de piloter Cuda et un fichier contenant le noyau (kernel) à envoyer aux processeurs du GPU. Enfin vous y trouverez également le fichier TimerManager permettant de gérer des timers.

Exercice 1 : Calcul vectoriel simple

Sur le code source récupéré (renommez les fichiers blank.*!), vous allez faire des opérations simples sur un vecteur. Dans notre exemple, nous allons faire, sur chaque élément du vecteur *v*, initialisé à $v[i]=i$, l'opération suivante :

$$v[i]=i*\sin(v[i])$$

Nous travaillerons sur un vecteur de taille variable *n*, décomposé en $n=nt*nb$ où *nt* correspondra au nombre de thread par bloc, et *nb* au nombre de blocs. Dans un premier temps, vous travaillerez avec un vecteur de taille réduit (par exemple *nb = 1* et *nt = 32*)

1. Dans un premier temps, créez trois tableaux en C (sur le host), le premier stockera les données sources sur lequel nous ferons les calculs. Le second stockera le résultat du calcul réalisé par le CPU. Le troisième stockera le résultat issu du calcul GPU.
2. Réalisez la fonction de calcul sur CPU. En C, la fonction sinus s'écrit *sinf*.
3. Insérer des timers pour calculer le temps passé par la version CPU.
4. Dans la fonction *runTest* (fichier *blank.cu*), créez deux tableaux sur GPU avec la fonction *cudaMalloc*. Puis recopier le tableau source dans le premier tableau GPU grâce à la fonction *cudaMemcpy*. Notez que la fonction *runTest* doit prendre des arguments supplémentaires (notamment les tableaux d'entrée et sortie coté host). Vous utiliserez de la mémoire globale coté device.
5. Faites l'appel au kernel et implémentez la fonction kernel du GPU.
6. Calculez grâce aux *event* CUDA, le temps passé par le kernel pour faire son calcul.
7. Dans le fichier *main.cpp*, comparez les deux résultats obtenus (CPU & GPU)
8. Implémentez le kernel en utilisant de la mémoire shared. Attention à allouer de la mémoire shared à l'appel du kernel.

Exercice 2 : Traitement d'images simple

Réaliser en Cuda une application permettant de convertir une image en luminance. Pour ce faire, vous reprendrez une partie du code vu dans le TD1 (pour la partie CPU & le chargement des images). Pour mémoire la luminance L d'un pixel coloré se calcule par :

$$L = 0.299\,R + 0.587\,V + 0.114\,B$$

1. Récupérer le calcul de la luminance en CPU du TP 1 et récupérer les temps de calcul de son exécution
2. Calculez en Cuda la luminance de votre image. Récupérer également les temps de calculs. Le kernel ressemble à

```
__global__
void rgba_to_greyscale(unsigned char* const rgbaImage,
                       unsigned char* const greyImage,
                       int numRows, int numCols)
```

Exercice 3: Calcul matriciel

Réalisez, en cuda, l'addition de 2 matrices. Comparez avec l'implémentation (si disponible de CUBLAS).