GPU — TP 1

5 février 2014

1 Préambule

Dans ce TP nous allons aborder :
- La programmation sur GPU avec CUDA NVCC
- Le principe d'allocation mémoire et de transport de données avec CUDA
- Le modèle mémoire d'un GPU et les benefices que l'on peut en tirer

2 Premiers pas

La base de tout langage de programmation commence avec l'écriture d'un hello world. Pour compiler le programme helloworld, entrez la ligne suivante :

nvcc − arch = sm_20 − o hello helloworld.cu

Pour executer, il suffit ensuite de lancer l'exécutable :

./hello

Dans le cas présent, avec un GPU de type Fermi, vous devriez voir un "HELLO WORLD !" qui est affiché avec un caractère par ligne. Maintenant, si vous augmentez le nombre de blocs dans le programme que va t'il se passer sachant que le GPU est une plateforme de calcul parallèle ? Comment pouvez-vous expliquer ce phénomène ?

3 Implémentation

3.1 L'addition vectorielle

L'addition vectorielle est un bon exemple pour comprendre l'allocation mémoire et la distribution des données sur un GPU en gardant un kernel assez basique. Après avoir regarder les notions de `cudaMemcpy, cudaMalloc, blocs et threads` modifier le code addition_vectorielle.cu pour qu'il compile et fonctionne correctement. Utiliser la ligne de compilation suivante :

nvcc − o vector addition_vectorielle.cu

Le resultat à obternir est $c[4194303] = 8388606$.

3.2 Stencil 1D

Dans le fichier 1Dstencil.cu vous trouverez un exemple de filtre permettant d'appréhender la notion de mémoire partagé d'un GPU. Dans un premier temps, faites tourner le code sans utiliser la mémoire partagé et utiliser une fonction de mesure de temps pour timer le code.

Essayez ensuite d'utiliser la mémoire partagé et faites tourner votre code plusieurs fois. Les résultats vous sembles t'ils correct ? Que devez-vous faire pour corriger cela ?

Les performances sont elles les même avec et sans la mémoire partagée ?

Utiliser la ligne de compilation suivante :

nvcc − o stencil 1*Dstencil.cu*

3.3 Produit Matrice vecteur

Ecrire un produit de matrices-vecteur sur GPU et vérifer que le code fonctionne. Essayer maintenant de comparer les performances sur différentes tailles de matrices avec une version CPU que vous écrirez comprenant du openmp et de la vectorization. Que pouvez-vous en déduire de l'utilité d'un GPU dans le calcul scientifique ?