Programmation sur Processeur Graphique – GPGPU
TD 4 : mémoire partagée et synchronisation

Centrale Nantes
P.-E. Hladik, pehladik@ec-nantes.fr

—

Version bêta (30 novembre 2022)

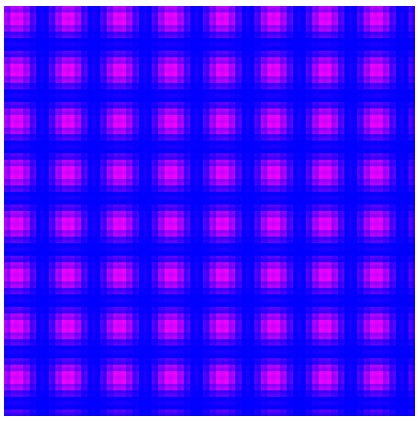
1 Diachronie

Objectif 1.1

— utiliser des synchronisation

(1.1) Travail à faire : C'est trouble

1. Récupérer l'archive `code_td6.zip`.
2. Compilez le code avec `nvcc -o part1 part1.cu`
3. Exécutez le code et observez l'image produite.
4. Pourquoi n'obtient-on pas l'image ci-dessous ? Modifier le code (une ligne) pour obtenir le bon résultat.



2 Stencil 1D

Objectif 2.1

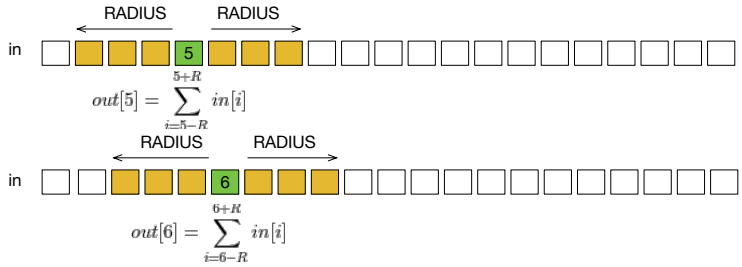
— utiliser une mémoire partagée

(2.1) Travail à faire : Ajout d'une mémoire partagée

Le code du fichier `stencile1D` permet de réaliser la somme des valeurs proches dans un tableau *in* :

$$out[k] = \sum_{i=k-R}^{k+R} in[i]$$

avec R la taille de la fenêtre sur laquelle se fait la somme. Un thread est utilisé pour chaque somme. La figure ci-dessous représente le calcul réalisé pour les valeurs k 5 et 6.



On remarque que des données communes sont utilisées pour le calcul de chaque valeur. Il pourrait être intéressant de ne les charger qu'une seule fois dans un *block* avant de réaliser les sommes.

1. Récupérer le fichier `stencile1D`
2. Compilez le code avec `nvcc -o stencil1D stencil1D.cu`
3. Identifiez les parties du codes qui font le calcul de la somme
4. Modifiez le code pour utiliser la mémoire partagée au niveau d'un *block*. Chargez toutes les données utiles pour les threads du *block* avant de faire les sommes (mettez en place un espace mémoire d'une taille égale au nombre de thread d'un bloc plus deux fois RADIUS)

3 Multiplication de matrice avec tuile

Objectif 3.1

— utiliser une mémoire partagée avec une décomposition en tuiles

(3.1) Travail à faire : Utilisation de tuiles

Reprenez votre programme précédent de multiplication de matrice et faites en une version avec des tuiles. Y-a-t-il un gain de performance ?