TD #3: GPU – CUDA et Patterns

Jonathan Rouzaud-Cornabas (jonathan.rouzaud-cornabas@insa-lyon.fr)

Rendu : Le code source

1 Histogrammes

Le but de cette exercice est d'implémenter une version efficace de l'algorithme de l'histogramme pour un tableau d'entrée contenant des caractères ASCII. Il y a 128 caractères ASCII et chaque caractère sera compté dans son propre sac (et donc 128 sacs au total). Les sacs seront des compteurs (entier 32-bit non signé) pour éviter les dépassements de taille. Pensez à réutiliser l'approche avec privatisation et mémoire partagée pour chaque bloc de thread puis utiliser les atomics pour l'histogramme final. Vous devrez donc:

- Allouer la mémoire sur le device
- Copier de la mémoire depuis l'hôte vers le device
- Définir et initialiser les dimensions de la grille du noyau ainsi que le nombre de threads dans chaque bloc
- Invoquer le noyau CUDA
- Copier les résultats depuis le device vers l'hôte
- Libérer la mémoire du device
- Ecrire le noyau CUDA

Question 1.1 Pour cela, vous allez devoir compléter le code (cuda_pattern_td_1.cu)

2 Stencil

Le but de cet exercice est de réaliser un stencil à 7 points en utilisant des carreaux pour les données en mémoire partagée.

Vous implémenterez un stencil en 7 points sans vous souciez des conditions aux bornes. Le résultat est coupé pour que la valeur reste entre 0 et 255.

```
for i from 1 to height-1:
  # notice the ranges exclude the boundary
  for j from 1 to width-1: # this is done for simplification
    for k from 1 to depth-1:# the output is set to 0 along the boundary
      res = in(i, j, k + 1) + in(i, j, k - 1) + in(i, j + 1, k) +
      in(i, j - 1, k) + in(i + 1, j, k) + in(i - 1, j, k) - 6 * in(i, j, k)
      out(i, j, k) = Clamp(res, 0, 255)
    end
  end
end
```

Avec Clamp défini comme suit:

```
def Clamp(val, start, end):
  return Max(Min(val, end), start)
end
```

Et in(i, j, k) et out(i, j, k) sont définie comme suit:

```
#define value(arry, i, j, k) arry[((i)*width + (j)) * depth + (k)]
#define in(i, j, k) value(input_array, i, j, k)
#define out(i, j, k) value(output_array, i, j, k)
```

- Implémenter le stencil
- Implémenter le code de lancement du stencil avec une grille et un bloc 2D.

Question 2.1 Pour cela, vous allez devoir compléter le code (cuda_pattern_td_2.cu)

3 Réduction

Le but de cet exercice est d'implémenter un code qui provoque la réduction d'une liste 1D. La réduction doit donner la somme de la liste. Vous devez utiliser les différentes optimisations discutées en cours. Vous devez penser aux conditions de bornes et l'utilisation de la mémoire partagée

- Allouer la mémoire sur le device
- Copier de la mémoire depuis l'hôte vers le device
- Définir et initialiser les dimensions de la grille du noyau ainsi que le nombre de threads dans chaque bloc
- Invoquer le noyau CUDA
- Copier les résultats depuis le device vers l'hôte
- Libérer la mémoire du device
- Ecrire le noyau CUDA
- Implémenter la boucle CPU qui effectue la somme finale

Question 3.1 Pour cela, vous allez devoir compléter le code (cuda_pattern_td_3.cu)