Parallélisation de code Stencil via Kokkos & OpenMP

CHPS0801

Alexis LABILLE – Kilyan SLOWINSKI

Sommaire

Introduction

Le but de ce projet est de développer une méthode de stencil en 5 points et d'en améliorer les performances à l'aide d'OpenMP et de Kokkos. Nous commençons par une implémentation itérative classique, puis nous développons des versions optimisées pour le CPU et le GPU en utilisant OpenMP et Kokkos. Ces différentes versions seront comparées en termes de performances.

Notons que tous les tests de performances ont été effectués sur l'image "lena.jpg" avec un stencil de 100 itérations. La machine sur laquelle les tests ont été effectués n'est pas Roméo (qui n'était pas disponible) mais un ordinateur portable disposant d'un ryzen 5 5600H (3.3GHz, 6 coeurs, Threads).

Utilisation

Les différentes versions de l'application peuvent être compilé à l'aide de l'outil cmake:

> cmake .

> make

Développement de la méthode en itératif

La méthode développée est celle du stencil en 5 points.

Ajout d'un padding

Pour préparer le code au parallélisme et éviter l'utilisation d'instructions conditionnelles (IF) coûteuses, nous ajoutons une marge extérieure (padding) à la matrice de travail. Ainsi, il n'est plus nécessaire de vérifier les conditions de bord, ce qui simplifie et accélère les calculs. Cette bordure n'est pas enregistrée à la fin de la boucle.

Résultats

```
iamako@iamako-IdeaPad-Gaming-3-15ACH6:~/Documents/CHPS0801/projet/Stencil-Kokkos-OpenMP$ ./stencil base.exe
Usage: stencil base.exe [params] input


        input (value:img/lena.jpg)
                input image
-----------------------------------
Execution time: 1072 milliseconds
-----------------------------------
Image successfully written to output image.jpg
```

Kokkos

Parallélisation de la boucle principale

Pour améliorer les performances, nous ciblons la boucle principale qui applique le stencil. Nous utilisons l'instruction parallel_for de Kokkos avec une politique de plage (range

policy). Dans notre cas, nous avons décidé de paralléliser les parties horizontales et verticales du stencil sur l'image.

Utilisation de vues

Kokkos permet l'utilisation de "vues" pour optimiser la gestion des données sur différentes architectures. Les données traitées sont les matrices dérivées de l'image d'origine. Nous créons donc des vues spécifiques pour ces données.

Espace de mémoire

Pour la parallélisation sur CPU, l'espace mémoire utilisé reste celui de l'hôte. Pour l'accélération matérielle via le GPU, il est nécessaire de transférer les données vers le device en utilisant l'espace mémoire CudaSpace et l'instruction deep_copy. Nous récupérons les données après traitement avec la même instruction.

Résultats (CPU)

On remarque une claire baisse des performances par rapport à la version séquentielle, indiquant que l'optimisation pour CPU avec Kokkos nécessite des ajustements supplémentaires.

```
Usage: stencil kokkos cpu.exe [params] input


        input (value:img/lena.jpg)
                input image
-----------------------------------
Execution time: 4827 milliseconds
-----------------------------------
Image successfully written to output_image.jpg
```

Résultats (GPU)

Des erreurs de compilation ont empêché l'évaluation des performances sur le GPU.

OpenMP

Implémentations des tâches

OpenMP propose un parallélisme par tâche via la directive **#pragma omp task**. Celle-ci génère alors le code correspondant qui pourra être exécuté n'importe quand par l'un des threads. À noter qu'il n'y a pas de dépendances entre les tâches dans une même exécution de fonction, vu qu'on lit dans une image et écrit dans l'autre.

Le découpage de l'image proposé est tel que chaque tâche disposera de 100 lignes chacune afin de maximiser les accès au cache.

Niveaux de parallélisme

On découpe donc les tâches à l'intérieur de la fonction jacobi_task, il aurait pu être intéressant de créer les tâches au niveau de la boucle dans le programme principal, celle-ci lançant la fonction d'abord sur l'image vers une image2 puis vice versa, cependant les dépendances à gérer serait problématiques.

En effet, pour chaque itération de cette boucle principale, une ligne de plus rentrerait dans les dépendances étant donné que l'on utilise un calcul de flou sur les voisins directs.
C'est pour cela que l'on se contente de créer des tâches à l'intérieur de la fonction.

Résultats (CPU)

```
iamako@iamako-IdeaPad-Gaming-3-15ACH6:~/Documents/CHPS0801/projet/Stencil-Kokkos-OpenMP$ ./stencil omp.exe
Usage: stencil omp.exe [params] input


        input (value:img/lena.jpg)
                input image
-----------------------------------
Execution time: 7177 milliseconds
-----------------------------------
```

A noter que le code ne produit pas le résultat escompté.

Conclusions

Les différentes méthodes d'optimisation ont montré des résultats variés. La version Kokkos pour CPU nécessite des ajustements supplémentaires pour améliorer les performances, et des erreurs de compilation ont empêché l'évaluation des performances sur le GPU. L'implémentation OpenMP n'a pas produit les résultats escomptés, indiquant que des ajustements sont nécessaires pour tirer pleinement parti du parallélisme. Des ajustements et des optimisations supplémentaires sont nécessaires pour exploiter pleinement les capacités des deux bibliothèques.