Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

CEG 4536 Architecture des ordinateurs III
Automne 2024

A soumettre le 23 novembre à 23h59.

Lab3 : Optimisation de la hiérarchie de la mémoire CUDA

1. Introduction

Dans le domaine du calcul haute performance, en particulier avec les accélérateurs GPU, une gestion efficace de la mémoire est cruciale. CUDA fournit un modèle de mémoire robuste qui optimise la latence et la bande passante en gérant efficacement la mémoire de l'hôte et celle du périphérique. Ce projet explore la hiérarchie de la mémoire CUDA, en se concentrant sur les rôles des différents types de mémoire (par exemple, les registres, la mémoire partagée, la mémoire globale) et en optimisant les schémas d'accès à la mémoire afin d'obtenir une programmation GPU efficace. En exploitant les capacités de CUDA, les étudiants acquerront une expérience pratique de l'optimisation du débit de la mémoire, de la réduction de la latence et de l'amélioration des performances des applications GPU.

2. Objectif

L'objectif de ce projet est de développer une application basée sur CUDA qui exploite la hiérarchie de mémoire CUDA pour maximiser le débit de la mémoire et optimiser les performances. Plus précisément, les étudiants concevront et mettront en œuvre des solutions pour minimiser la latence d'accès à la mémoire et le gaspillage de la bande passante, en explorant des techniques telles que la coalescence de la mémoire, la distribution des banques de mémoire dans la mémoire partagée et la minimisation du transfert de données entre l'hôte et l'appareil.

3. Réalisation

Le projet sera réalisé en trois phases principales :

1. **Optimisation de l'accès à la mémoire** :
   Analyse et optimisation des schémas d'accès à la mémoire dans un noyau CUDA. Il s'agit notamment de garantir l'alignement et la coalescence des accès à la mémoire globale et de réduire les accès mal alignés à la mémoire.
2. **Utilisation de la mémoire partagée** :
   Mettre en œuvre des techniques d'utilisation de la mémoire partagée pour gérer efficacement la communication intra-bloc et réduire l'accès à la mémoire globale. Cette phase implique de comprendre et de traiter les conflits entre les banques en configurant les schémas d'accès à la mémoire partagée et en mettant en œuvre un rembourrage de la mémoire si nécessaire.
3. **Profilage et optimisation des performances** :
   Profilage des performances du noyau CUDA à l'aide d'outils tels que `nvprof` ou similaires, en mesurant des paramètres tels que l'efficacité du chargement/stockage global, le comportement du

cache et le nombre de transactions en mémoire. En se basant sur les résultats du profilage, les étudiants affineront leur code de manière itérative afin d'améliorer l'efficacité de la mémoire.

4. Exemple d'application : Optimisation de la multiplication matricielle à l'aide de la hiérarchie de mémoire CUDA

La multiplication matricielle est une opération fondamentale dans de nombreux calculs scientifiques et techniques. Dans cet exemple, nous visons à optimiser un noyau de multiplication matricielle dans CUDA en utilisant efficacement différents niveaux de mémoire et en optimisant les schémas d'accès à la mémoire. L'objectif principal est d'obtenir un débit mémoire élevé et une faible latence, afin d'améliorer les performances globales du calcul.

Étapes de mise en œuvre

1. **Définir le problème**
Implémenter un noyau CUDA pour effectuer la multiplication matricielle de deux matrices *A* et *B* afin de produire la matrice *C. Chaque*
matrice sera stockée dans la mémoire globale. Chaque matrice sera initialement stockée dans la mémoire globale, les étapes suivantes visant à réduire la dépendance à l'égard de la mémoire globale en utilisant la mémoire partagée.

2. **Optimisation de l'accès à la mémoire globale** :
Veiller à ce que les schémas d'accès à la mémoire pour les matrices *A* et *B* coïncident. Cela peut se faire en alignant les accès à la mémoire sur des limites de 128 octets, de sorte que chaque chaîne récupère les données en morceaux contigus. Cela minimise le nombre de transactions en mémoire, ce qui réduit la latence de la mémoire.

3. **Utilisation de la mémoire partagée** :
Charger des parties des matrices dans la mémoire partagée afin de réduire la fréquence d'accès à la mémoire globale à forte latence. Cette approche est bénéfique car la mémoire partagée est plus rapide et accessible à tous les threads d'un bloc. Chaque bloc de threads chargera une tuile (sous-matrice) de *A* et *B* dans la mémoire partagée, effectuera la multiplication sur ces tuiles, puis stockera le résultat dans *C*.

4. **Optimiser les schémas d'accès à la mémoire partagée** :
Gérer les schémas d'accès à la mémoire partagée afin d'éviter les conflits de banque. Cela peut inclure le remplissage des tableaux de mémoire partagée pour s'assurer que les différents threads accèdent simultanément aux différentes banques de mémoire. Un remplissage et un alignement corrects sont essentiels pour maximiser la bande passante de la mémoire partagée.

5. **Profilage des performances** :
Profiler le noyau de multiplication de matrice en utilisant `nvprof` ou un outil de profilage similaire. Les mesures clés sont les suivantes :
Efficacité globale de la charge et du stockage : Pour s'assurer que les transactions de mémoire sont coalescentes et alignées.
Efficacité de la mémoire partagée : Vérifier qu'il n'y a pas de conflits de banques.
Temps d'exécution du noyau : pour mesurer l'amélioration des performances après les optimisations.

6. **Optimisation itérative** :
En se basant sur le retour d'information du profilage, affiner le noyau. Par exemple, utiliser le déroulement de boucle pour maximiser le parallélisme au sein de chaque bloc de threads ou ajuster la taille des tuiles pour améliorer l'utilisation de la mémoire partagée.

5. Les produits à livrer

1. **Base de code** :
Une base de code CUDA entièrement fonctionnelle mettant en œuvre les optimisations de l'accès à la mémoire

et de l'utilisation de la mémoire partagée.

2. **Rapport de performance** :
Un rapport détaillant les schémas d'accès à la mémoire utilisés, les résultats du profilage et les optimisations spécifiques appliquées pour améliorer les performances.

3. **Démonstration** :
Une démonstration enregistrée ou en direct présentant l'application optimisée fonctionnant sur un GPU, avec des explications sur la manière dont chaque technique d'optimisation a contribué aux gains de performance.

6. L'évaluation

L'évaluation sera basée sur :

Fonctionnement (30 %) : L'application doit répondre aux exigences et produire des résultats exacts.
Techniques d'optimisation (30%) : Mise en œuvre efficace des optimisations de l'accès à la mémoire et de la mémoire partagée.
Gains de performance (20 %) : Améliorations de l'efficacité et du débit mesurées par des mesures de profilage.
Documentation et rapport (20 %) : Documentation claire et complète, comprenant des explications sur les optimisations et l'analyse des données de profilage.