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

À remettre 28 septembre 23h 59

Lab1 : Optimisation de la simulation de feux de forêt avec CUDA

1. Introduction

Dans ce laboratoire, vous allez travailler sur un code de simulation de feux de forêt qui utilise une grille de taille 1000 ×1000. Le feu se déclenche à 100 emplacements distincts dans la forêt. Le programme, fourni dans sa version de départ, est implémenté de manière séquentielle. Il simule la propagation du feu, la combustion des arbres et leur extinction. La grille est affichée à l'aide de la bibliothèque OpenGL, où chaque cellule représente un arbre ou un espace vide.

L'objectif de ce laboratoire est de paralléliser le code existant en utilisant CUDA C afin de tirer parti de la puissance des processeurs graphiques (GPU) et ainsi rendre la simulation plus rapide et plus efficace. Les étudiants devront identifier les parties du code où l'optimisation par parallélisation est la plus pertinente, telles que la mise à jour de l'état de chaque cellule de la forêt.

2. Objectif

L'objectif principal de ce laboratoire est de transformer le code séquentiel en une version optimisée utilisant CUDA C pour accélérer la simulation. Vous allez apprendre à :

- Identifier les sections du code qui peuvent être parallélisées.
- Utiliser CUDA C pour exécuter des calculs en parallèle sur un GPU.
- Mesurer les gains de performance obtenus grâce à la parallélisation.

3. Plateforme de développement

Le développement et l'optimisation du programme se feront sur des machines équipées de GPU compatibles CUDA. Les outils à utiliser incluent :

- CUDA Toolkit (12.6 ou supérieur) pour la compilation des programmes CUDA.
- Visual Studio 2022 pour l'édition et le débogage du code.
- CUDA Debugger pour tester et profiler vos kernels CUDA.

Vous utiliserez OpenGL pour l'affichage de la simulation, et le travail se fera sur des stations de travail avec des GPU NVIDIA prenant en charge CUDA.

4. Réalisation

Étape 1 : Comprendre le code de départ

- Analysez le code fourni. Il s'agit d'une simulation de feux de forêt où chaque cellule de la grille représente un arbre ou un espace vide. Un feu se déclenche à 100 endroits aléatoires, se propage à des cellules voisines, et les arbres en feu finissent par s'éteindre après un certain temps.
- Étape 2 : Identifier les opportunités de parallélisation
- La mise à jour de la grille est la section du code qui peut être parallélisée de manière significative. Chaque cellule de la grille peut être mise à jour indépendamment des autres.
- Analysez la fonction updateForest() qui est responsable de la mise à jour de l'état des arbres en feu et de la propagation du feu aux cellules voisines. C'est cette partie qui doit être optimisée en utilisant CUDA.

Étape 3 : Implémenter la parallélisation avec CUDA C

- Initialisation de CUDA : Allouez la mémoire pour la grille (forest) et le temps de combustion (burnTime) sur le GPU à l'aide de cudaMalloc().
- Kernel CUDA : Implémentez un kernel qui met à jour l'état de chaque cellule de la forêt en parallèle.
- Exécution parallèle : Assurez-vous que chaque cellule de la grille est mise à jour en parallèle à l'aide de plusieurs threads sur le GPU.
- Gestion des blocs et des threads : Découpez la grille en blocs de threads CUDA pour une exécution optimisée.

Étape 4 : Mesurer les performances

- Mesurez le temps d'exécution du programme séquentiel et comparez-le à la version optimisée avec CUDA. Utilisez les outils de profilage CUDA pour identifier les gains de performance et toute optimisation supplémentaire possible.

5. Livrables

Chaque équipe devra soumettre un rapport contenant les éléments suivants :

- Une explication des parties du code qui ont été parallélisées.
- Le code source modifié avec l'implémentation CUDA.
- Une analyse des performances montrant les temps d'exécution avant et après l'optimisation.
- Les captures d'écran du programme en cours d'exécution avec des résultats visuels de la simulation.

6. Critère d'évaluation

- Correctitude : Le programme doit fonctionner correctement après l'optimisation. La simulation doit se comporter de manière identique à la version séquentielle.
- Parallélisation efficace : Le code doit montrer une utilisation correcte et efficace de CUDA, avec une parallélisation significative des parties appropriées du programme.
- Amélioration des performances : Des gains de performance mesurables doivent être démontrés avec la version CUDA. La différence de temps d'exécution entre la version séquentielle et la version parallèle doit être clairement expliquée.
- Qualité du code : Le code doit être bien structuré, commenté et respecter les bonnes pratiques de programmation.

Remarque : Ce laboratoire est une première introduction à la parallélisation avec CUDA, il est donc important de bien comprendre les concepts de base de CUDA avant de commencer à coder.