Nom et Prénom : DJAGHLOUL AYOUB AHMED AMINE (IA)

Rapport du Projet MacMahon Solver

Introduction

Ce rapport décrit l'implémentation et les performances de différents algorithmes de backtracking pour résoudre le jeu de MacMahon. Nous avons exploré une implémentation séquentielle et deux variantes de backtracking parallèles en utilisant la programmation multithread de C++11.

Manuel d'Utilisation

Compilation des programmes
Les programmes peuvent être compilés en utilisant les commandes suivantes :
Pour l'algorithme séquentiel:
g++ main1.cpp -Wall -o main.o

Pour l'algorithme avec threads:
g++ -std=c++11 -pthread main_thread.cpp -o main_thread

Pour l'algorithme avec ThreadPool:
g++ -std=c++11 -pthread main_threadpool.cpp -o main_threadpool

Exécution des programmes

Les programmes compilés peuvent être exécutés de la manière suivante :

Pour le plateau 6x6 avec l'algorithme de thread:
./main_thread 6x6_colorv2.txt

Pour le plateau 6x6 avec l'algorithme séquentiel:
./main 6x6_colorv2.txt

Présentation des codes-source :

Déclarations de base:
Déclaration de variables globales ROWS et COLS pour les dimensions du plateau.
Une énumération TileColor définissant les couleurs possibles pour les côtés des tuiles, y compris EMPTY, BLUE, GREEN, RED.
Classe MacMahonTile:
Gère une tuile avec quatre côtés de couleurs qui peuvent être définis et consultés.
Classe MacMahonBoard:
Représente le plateau de jeu comme une grille de tuiles.
Permet de placer une tuile à une position donnée et d'afficher le plateau.

Nom et Prénom : DJAGHLOUL AYOUB AHMED AMINE (IA)

Fonction canPlaceTile:
Vérifie si une tuile peut être placée à une certaine position en tenant compte des contraintes de couleur avec les tuiles adjacentes.
Fonctions de conversion et de lecture:
charToTileColor convertit un caractère en TileColor.
readDataFromFile lit la taille du plateau et les tuiles depuis un fichier.
Fonction récursive solve:
Tente de résoudre le casse-tête en plaçant les tuiles sur le plateau de manière récursive.

Version Multi-threads:

Multi-threading : la fonction solveWithThread est conçue pour être exécutée sur des threads distincts, chaque thread essayant de résoudre le puzzle en commençant par une tuile différente. Une variable std::atomic_bool solutionFound est utilisée pour signaler quand une solution a été trouvée afin que les autres threads puissent s'arrêter.

Version threadpool:

Ajout de la classe **ThreadPool**:
Nous avons intégré la classe ThreadPool pour optimiser le traitement parallèle des tâches. Cette classe offre une abstraction de niveau supérieur pour la gestion de plusieurs threads, permettant ainsi une utilisation efficace des ressources du processeur. Grâce à cette classe, nous pouvons mettre en file d'attente et exécuter plusieurs tâches de manière asynchrone sans avoir à créer, gérer et synchroniser explicitement les threads.
La ThreadPool gère un ensemble fixe de threads qui attendent activement les tâches à exécuter. Lorsqu'une tâche est en file d'attente, elle est encapsulée dans un std::packaged_task afin de lier son futur résultat à un std::future, ce qui permet de récupérer le résultat de manière synchrone si nécessaire.

Implémentations des Algorithmes de Backtracking

Algorithme Séquentiel

L'algorithme séquentiel suit une approche classique de backtracking, explorant l'espace de solution de manière récursive et linéaire.

Algorithme Avec les Threads

Cette implémentation utilise std::thread de C++11 pour créer des threads qui exécutent des parties de l'algorithme de backtracking en parallèle.

Algorithme Avec ThreadPool

Cette variante utilise un pool de threads pour gérer les tâches de backtracking. Les threads sont réutilisés pour différentes tâches, ce qui minimise l'overhead de création et de destruction de threads.

Nom et Prénom : DJAGHLOUL AYOUB AHMED AMINE (IA)

Résultats de Performances

Les mesures de temps pour trouver une solution pour différentes tailles de plateau sont les suivantes :

Taille du Plateau	Algorithme Séquentiel	Algorithme avec Threads	Algorithme avec ThreadPool
4x4	0.003564 sec	0.00551131 seconds	0.00309966 seconds
5x5	0.217861 sec	0.0046872 seconds	0.00424424 seconds
6x6	21.127804 sec	1.91276 seconds	//////////////////

Interprétation des Résultats

4x4 et 5x5 Plateaux: Les temps d'exécution de l'algorithme avec threads sont meilleurs que l'algorithme séquentiel, ce qui suggère que le parallélisme peut efficacement réduire le temps de calcul, en particulier sur des problèmes de taille moyenne comme le plateau 5x5.
6x6 Plateau: Bien que l'algorithme avec threads montre une amélioration significative par rapport à l'approche séquentielle, le temps est plus long que pour les plateaux plus petits en raison de la complexité exponentielle croissante du problème.

Conclusion

Les algorithmes de backtracking parallèles ont montré des améliorations significatives dans les performances par rapport à l'approche séquentielle, démontrant l'efficacité de la parallélisation dans la résolution de problèmes combinatoires complexes comme le jeu de MacMahon.