Student: Abdoulaye SAYOUTI SOULEYMANE
Mail: abdoulaye.sayouti-souleymane@alumni.univ-avignon.fr
Group : IA-IL-CLA
ID : uapv2104389

Rapport : MiniProjet Tetravex

1. Objectif :

L'objectif est d'implémenter trois algorithmes pour la résolution du jeu Tetravex.

a) Le premier algorithme un de Backtracking récursive.

b) Le second algorithme utilise des Thread pour accélérer la résolution du jeu.

c) Le troisième algorithme utilise un thread pool pour gérer automatiquement les threads.

2. Architecture de la solution :

La solution est composée de trois dossiers et d'un fichier **main.cpp** comme indiquer à la Figure 2. Le dossier **data** contient les données du jeu Tetravex à résoudre. Le dossier **game** contient la modélisation du jeu Tetravex. Le dossier **solver** contient les trois différents algorithmes pour la résolution du jeu. La Figure 3 illustre le contenu du fichier **main.cpp**. La variable **nbThread** permet d'initialiser le nombre de threads pour le thread pool.

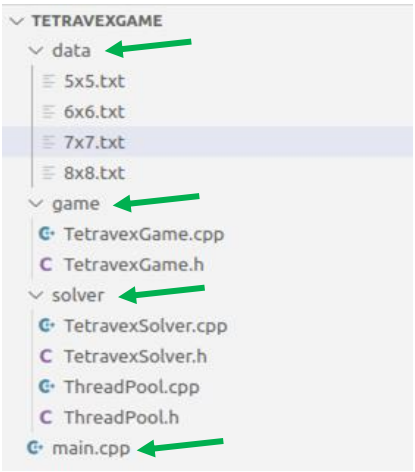

Figure 2 : Architecture de la solution

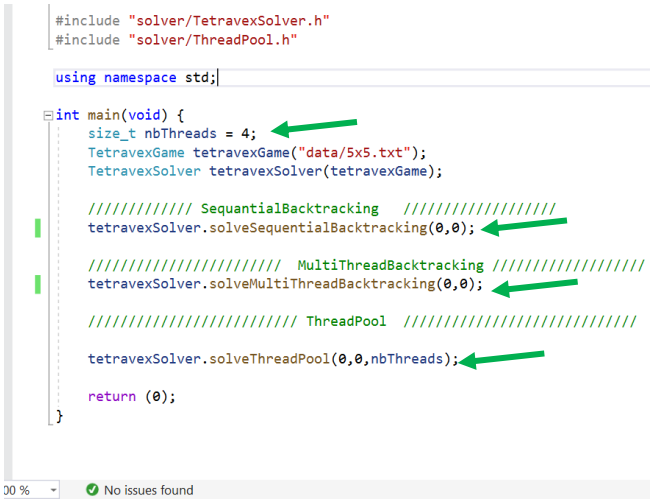

Figure 3 : fonction main

La classe **TetravexGame** permet de modéliser le jeu en y indiquant le chemin vers le fichier à lire. La classe **TetravexSolver** prends en paramètre le jeu et implémente les trois algorithmes de résolutions.

3. Fonctionnement des algorithmes :

a) Backtracking récursive :

i. L'algorithme backtracking récursive sélectionne une pièce non utilisée dans la liste des pièces.

ii. Il vérifie ensuite si la pièce peut être déposé dans la case suivante.

iii. Si oui, l'algorithme la dépose et retourne à l'étape i.

iv. Si non, l'algorithme retourne la pièce dans la liste des pièces non utilisé et retourne à l'étape i.

v. Si aucune pièce ne peut être déposer, l'algorithme retire la pièce déposée précédemment et retourne à l'étape i.

vi. Si la grille est pleine, alors une solution a été trouvé.

vii. Si toutes les combinaisons ont été essayé et que la grille n'est pas pleine alors aucune solution n'a été trouvé.

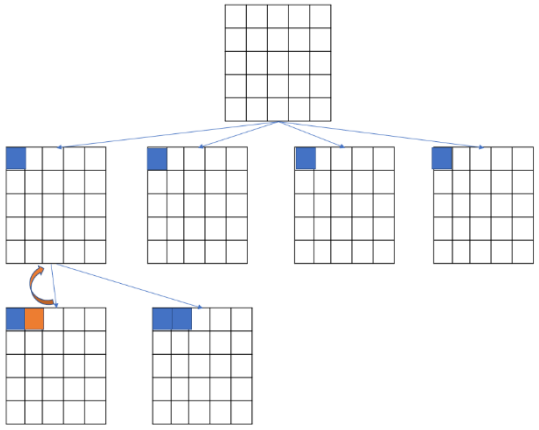

Figure 4 : Backtracking récursive

b) Backtracking parallèle :

Dans ce cas, plusieurs threads sont créés. Chaque thread se charge de résoudre une fraction du problème. La décision sur la première pièce à déposer est partagé entre les threads parmi la liste des pièces. Dans la solution proposée, N threads sont créés une grille NxN.

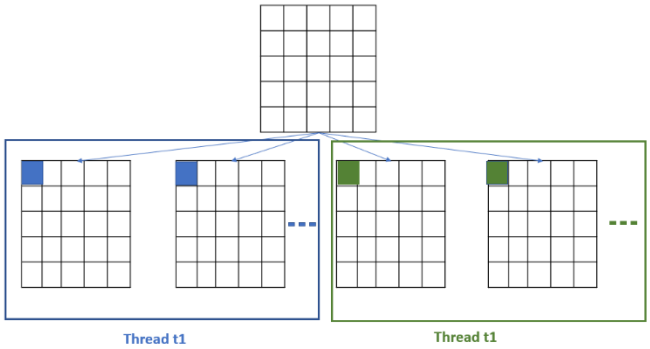

Figure 5 : Backtracking parallèle

c) Thread pool :

Dans le backtracking parallèle, les threads sont gérés manuellement. Avec le Thread pool, les threads sont gérés par une classe spécifique. Le thread pool se charge d'attribuer les tâches aux différents threads.

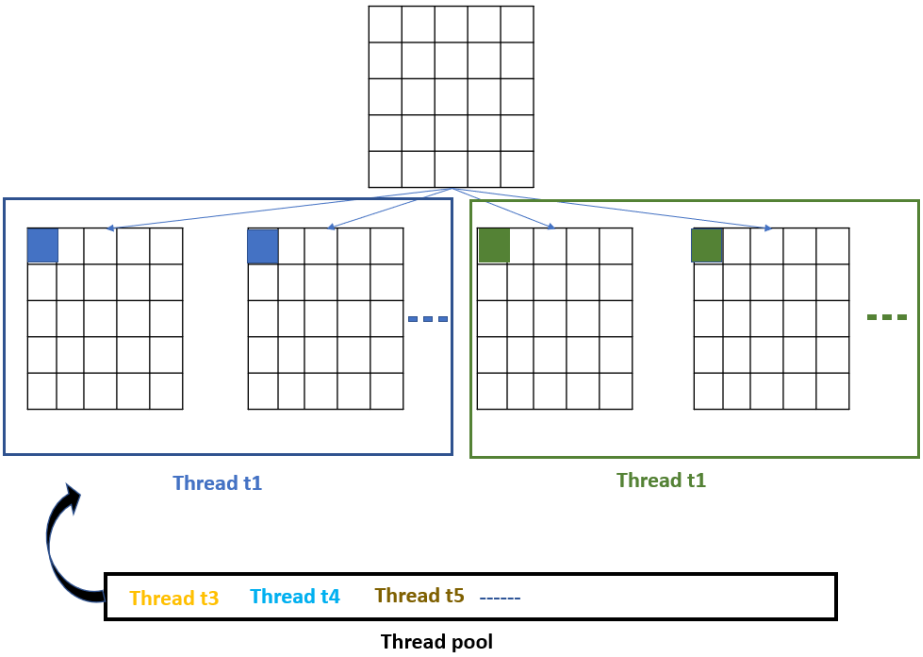

Figure 6 : Thread pool

4. Manuel d'utilisation de la solution :

L'algorithme backtracking parallèle est lancé par défaut. Ouvrir le fichier main.cpp pour choir l'algorithme à utiliser. Pour compiler et exécuté la solution, il faut exécuter étapes suivantes :

a) cd TetravexGame

b) g++ -O3 -o main game/*.cpp solver/*.cpp main.cpp

c) ./main

La solution s'affichera comme indiqué à droite de la figure 7. Le format de l'affichage est le suivant :

« A B : C » . A et B représente les coordonnées de la grille. Le C représente le numéro de la pièce qui s'y trouve. Par exemple la première pièce est déposée dans la quatrième case de la première ligne. La fenêtre indique également l'algorithme utilisé pour résoudre le jeu et le temps de résolution.

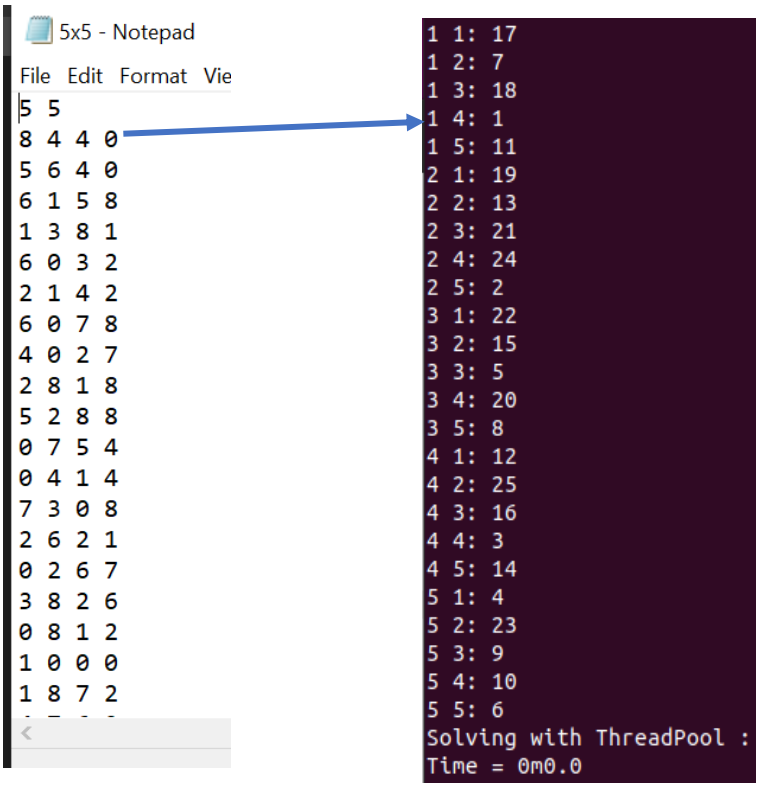

Figure 6 : Solution

5. Résultats obtenus et interprétations :

La machine utilisée pour évaluer les algorithmes possède les caractéristiques suivantes :

i. HP ElitBook 745 G3, System Ubuntu 20.10

ii. **Processeur :** AMD PRO A10-8700B R6, 10 Compute Cores 4C+6G, 1800 Mhz, 2 Core(s), 4 Logical Processor(s)

a) Comparaison entre les algorithmes :

Le Tableau A récapitule le temps de résolution des différentes tailles de jeu par les trois algorithmes. On peut remarquer que les algorithmes parallélisés sont plus rapides que le backtracking récursive normale. Le backtracking parallèle et le thread pool ont à peu près les mêmes performances. Huit threads ont été utilisé pour le thread pool et N threads pour le backtraking parallèle. Les performances sont mesurées en milliseconde. Cependant, l'algorithme de Backtracking récursive prends énormément de temps pour le 8x8.

Taille du jeu	Backtracking récursive	Backtracking parallèle	ThreadPool
5x5	0.2 ms	0.1 ms	0 ms
6x6	0.64 ms	0.11 ms	0.11 ms
7x7	12.69 ms	2.608 ms	2.623 ms
8x8	Inf	370.201 ms	394.671 ms

Tableau A: Performances des algorithmes

La Figure 7 mais plus en évidences l'écart des performances entre les algorithmes.

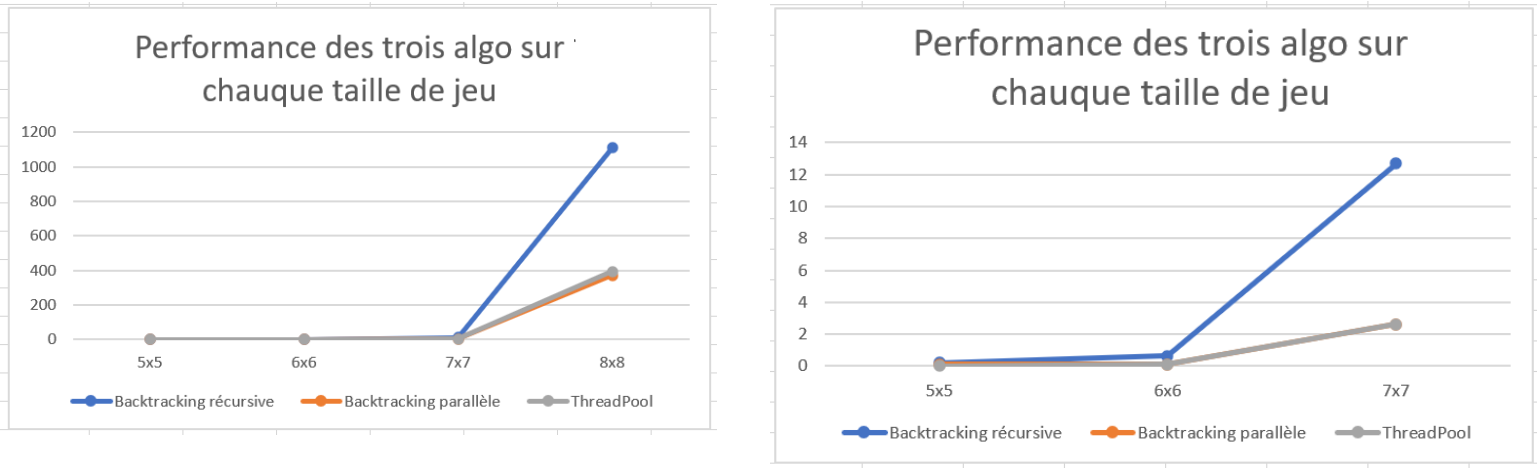

Figure 7 : Performances pour (5x5, 6x6, 7x7 et 8x8) à gauche et pour (5x5, 6x6 et 7x7) à droite

b) Performances du thread pool en fonction du nombre de threads.

Le Tableau B et la Figure 8 montrent que les performances sont en général meilleures pour un nombre de thread qui tourne au alentours du nombre de processeurs disponible sur la machine qui est 4 dans ce cas. L'axe des abscisses représente le nombre de thread et l'axe des ordonnés le temps en milliseconde.

Nombre de threads	**6x6**	**7x7**	**8x8**
2	0.4 ms	7.582 ms	Inf
4	0.11 ms	4.831 ms	214.31 ms
8	0.11 ms	2.326 ms	380.15 ms
16	0.11 ms	2.956 ms	368.328 ms

Tableau B: Performances du thread pool en fonction du nombre de thread

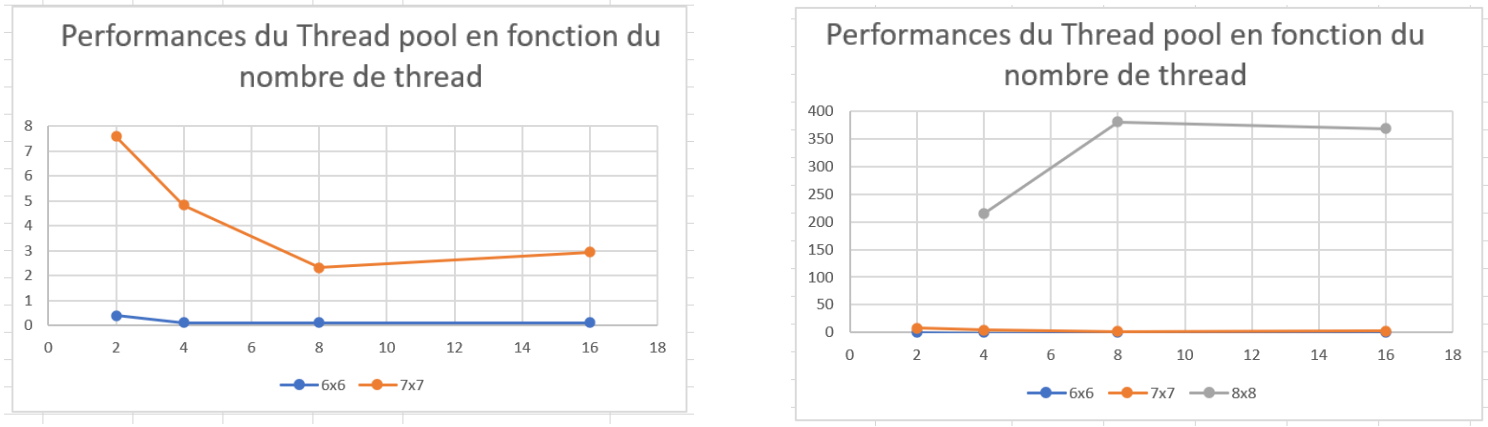

Figure 8: Performances pour (6x6 et 7x7) à gauche et pour (6x6, 7x7 et 8x8) à droite