SYSTÈME DE RECHERCHE D'INFORMATIONS POUR LES DOCUMENTS

CALCUL PARALLELE

Encadré par : Mr.Abbad Zakariae

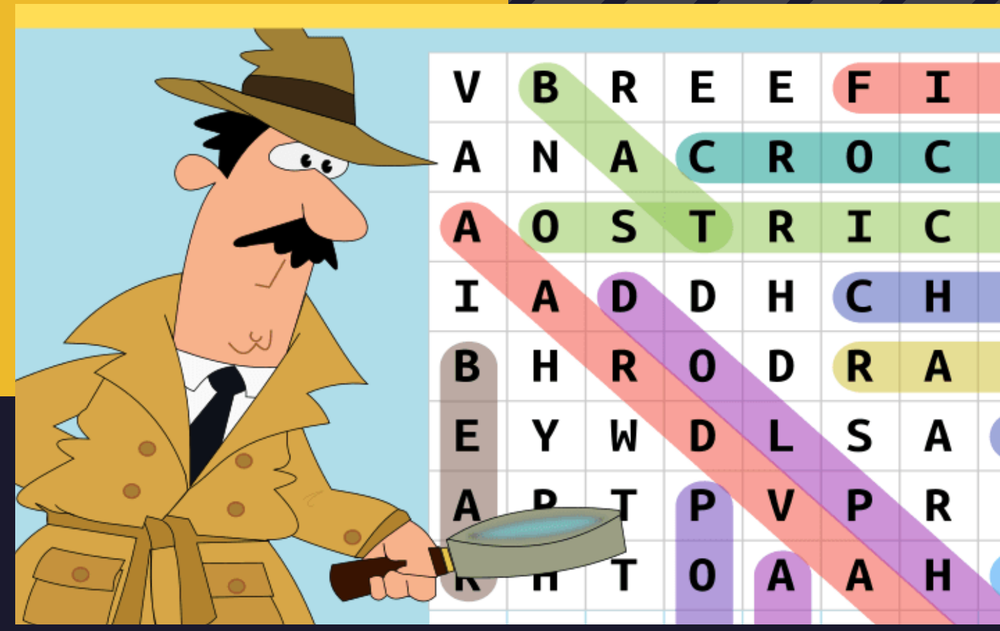
INTRODUCTION

La recherche de mots est utilisée partout, de la recherche locale des pages (Cntrl + F) à la recherche de mots sur les visionneurs des documents comme "reader" dans Windows. En fait, toute une branche appelée Information Retrieval a été développée pour ça. Ce projet a en fait été inspiré par la recherche d'informations. Il a beaucoup d'application dans le vrai monde .

READ MORE

PROBLEMATIQUE

- La technologie "**search word**" recherche des mots dans plusieurs fichiers texte en parallèle à l'aide de concepts **open MP**. Ceci réduit le temps d'exécution du code par rapport à la recherche séquentielle de mots.
- Les documents sont scannés mot par mot et un dictionnaire est maintenu afin que les mots puissent être recherchés.
- Puisque les documents peuvent contenir plusieurs milliers de termes, l'analyse de chaque terme peut prendre beaucoup de temps et donc on a une grande opportunite pour **la parallélisation**.
- Le but de ce projet est d'analyser quelle methode de recherche de mot est efficace pour gérer les fichiers multiples. Les resultats de ce projet nous donnerons la reponse .



Comprehension du prossesus

PREMIEREMENT

Comme son nom l'indique, la "recherche de mots / word search" consiste à rechercher des mots dans des documents en parallèle à l'aide de open MP.

Il ne s'agit pas simplement d'une simple recherche de mots, mais il classe également les documents en fonction de la pertinence de documents par rapport au mot recherché. Les documents sont scannés mot à mot et un dictionnaire est maintenu afin que les mots puissent être recherchés.

Étant donné que les documents peuvent contenir plusieurs milliers de termes parcourir chaque terme peut prendre beaucoup de temps et il y a donc beaucoup de possibilités pour
parallélisation.

DEUXIEMEMENT

Ce projet se focalise sur le principe des systèmes multithreads. Il utilise plusieurs threads pour lire
plusieurs fichiers et effectuer l'action.

L'action ici étant de rechercher le mot dans les fichiers, et cela en très peu de temps par rapport au système séquentiel.

Les paramètres sont similaires à celle de la méthode séquentielle, mais les processeurs sont utilisés pour faire le processus séquentiel sur plusieurs fichiers en même temps.

ANALYSER L'EXÉCUTION SÉQUENTIELLE VS PARALLÈLE AVEC LE SYSTÈME DE FICHIERS

01 L'execution sequentielle sur fichier texte

Prèsque 1000 fichiers texte sont chargés de données. Ensuite, un mot particulier (algo) est entré en entrée et le mot est vérifié dans chacune des données du fichier l'une après l'autre.

RÉSULTATS : (CODE SÉQUENTIEL)

Temps d'ecution : 0.3s

```
80    // entrer la commande "g++-11 -fopenmp finalwordsearch.cpp -o my1 && ./my1"

TERMINAL   DEBUG CONSOLE   COMMENTS   PROBLEMS   OUTPUT

Le mote existe  1 fois dans le fichier 991
Le mote existe  1 fois dans le fichier 992
Le mote existe  1 fois dans le fichier 993
Le mote existe  1 fois dans le fichier 994
Le mote existe  1 fois dans le fichier 995
Le mote existe  1 fois dans le fichier 996
Le mote existe  1 fois dans le fichier 997
Le mote existe  1 fois dans le fichier 998
Le mote existe  1 fois dans le fichier 999
Temps d'execution :0.322156
mohamedabdallaoui@Mohameds-MacBook-Pro test011_2 % g++-11 -fopenmp finalwordsearch.cpp -o my1 &&
Entrer le mot à rechercher : algo
Le mote existe  1 fois dans le fichier 127

Ln 62, Col 29   Spac
```

ANALYSER L'EXÉCUTION SÉQUENTIELLE VS PARALLÈLE AVEC LE SYSTÈME DE FICHIERS

02 L'execution parallele sur fichier texte

- Prèsque de 1000 fichiers texte sont chargés de données. Ensuite, un mot particulier (algo) est entré en entrée et le mot est en cours de vérification dans chacun des fichiers.
- #pragma omp permet d'executer la recherche du mot en parallel .
- Le concept utilisé est le fork join . Chaque sous thead execute la rechaerche des fichiers sur tous les 1000 fichiers , ce qui permet d'afficher le nombre d'itterations du mot dans chaque fichier.

RÉSULTATS : (CODE PARALLEL)

Temps d'ecution : 0.03s

```
75      cout << "Temps d'execution :" + to_string(end - start);
76      return 0;
77  }
```

TERMINAL DEBUG CONSOLE COMMENTS PROBLEMS OUTPUT

```
Le mote existe  1 fois dans le fichier 370
Le mote existe  1 fois dans le fichier 371
Le mote existe  1 fois dans le fichier 372
Le mote existe  1 fois dans le fichier 373
Le mote existe  1 fois dans le fichier 374
Le mote existe  1 fois dans le fichier 375
Le mote existe  1 fois dans le fichier 376
Le mote existe  1 fois dans le fichier 377
Le mote existe  1 fois dans le fichier 378
Temps d'execution :0.038282
mohamedabdallaoui@Mohameds-MBP test011_2 %
```

Conclusion :

- On constate un tres grande difference :
- **Code sequentiel : 0.3s**
- **Code parallel : 0.03s**
- On conclue donc que l'execution parallele est plus beaucoup efficace que l'execution sequentielle.

Travail Réalisé Par :
Kamel Tarek
Abdallaoui Mohamed
Belmessid Anas