INF5171 – Programmation concurrente et parallèle

TP2 – Automne 2021

K-Moyennes

Nom, prénom, matricule des membres de l'équipe	Hauchecorne, Jules, HAUJ21049307
Note finale sur 20	0

Problématique

Le but de cet atelier est de paralléliser l'algorithme de k-moyennes (*k-means*) à l'aide de TBB.

L'algorithme de k-moyennes accepte en entrée une liste d'échantillons et un nombre de groupe à créer (aussi appellé *cluster*). Il permet de classifier les échantillons qui sont semblables. À la figure suivante, on a généré un jeu de données avec des points dans le plan (x, y). On voit clairement deux groupes et l'algorithme est en mesure de les retrouver seulement à partir de la liste de points.

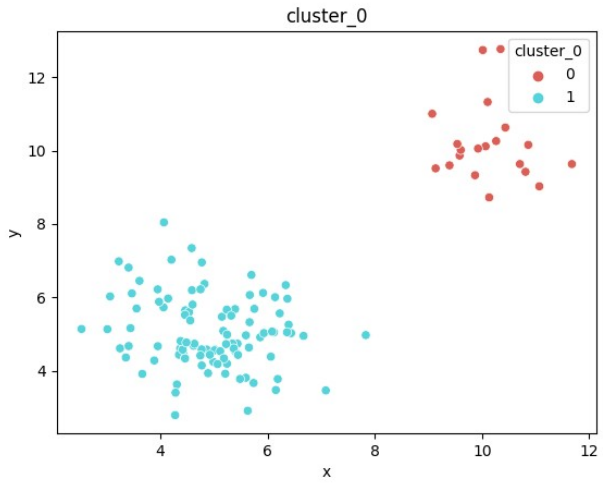


Figure 1: Groupes d'exemple

Par exemple, ce jeu de données pourrait représenter la taille et le poids de deux espèces d'animaux, les des chats et des chiens. Une fois les groupes établis, on peut se servir de cette information pour prédire si un animal est un chat ou un chien à partir de sa taille et de son poids.

Un autre exemple concerne l'analyse d'image. Chaque pixel RGB peut être considéré comme un échantillon à 3 dimensions. L'algorithme retrouve les couleurs semblables et on peut utiliser cette

méthode pour réduire le nombre de couleurs dans une image. Bref, l'algorithme est utilisé dans plusieurs domaines.

La première partie de l'algorithme consiste à retrouver le centre des groupes. On spécifie le nombre de groupes à former (soit la valeur de k). L'algorithme fonctionne de la manière suivante :

1. On choisit des échantillons au hasard comme centre des groupes pour initialiser l'algorithme.

2. Pour chaque échantillon, on calcule la distance entre de l'échantillon et le centre de chaque groupe. L'échantillon est assigné au groupe le plus près et on enregistre cette distance pour l'étape suivante. Si le centre le plus près est différent, alors l'échantillon change de groupe. On incrémente le nombre d'échantillons qui changent de groupe pour l'étape 4.

3. On calcule le nouveau centre d'un groupe basé sur les échantillons membres du groupe. Ceci implique de faire la moyenne des coordonnées des échantillons.

4. Tant qu'il y a au moins un échantillon qui change de groupe, on répète les étapes 2 et 3.

La deuxième étape est la classification. Elle consiste à évaluer la distance entre l'échantillon et chaque centre de groupe. L'échantillon est associé au groupe le plus près.

Cet algorithme a plusieurs particularités. Premièrement, comme le point de départ est aléatoire, il se peut que la solution ne soit pas identique d'une fois à l'autre et nécessite plus ou moins d'itérations pour converger. L'algorithme retrouve uniquement un minimum local, ce qui veut dire qu'il existe peut-être de meilleures solutions. Aux fins du calcule de l'accélération, il faut que l'exécution série et parallèle de l'algorithme soit identique. Pour cela, on réutilise les mêmes points de départ pour chaque version. Ils termineront avec le même nombre d'itérations.

À propos de *kmeans*

Les sources du laboratoire sont compilées avec `CMake`. Voir le fichier `README` pour plus de détails sur la compilation, les options pour le débogage et les dépendances de compilation pour Fedora et Ubuntu. Il est aussi possible de faire le TP sur Windows avec MinGW en suivant les mêmes instructions que les exemples du cours.

La compilation produit le binaire exécutable `kmeans`. Le programme génère des points aléatoires 2D en fonction d'un centre et d'une dispersion (variance). Chaque groupe est défini par une ligne dans un fichier texte. Le format comprend 5 valeurs : n_points centre_x centre_y stdev_x stdev_y. Voici un exemple dans lequel on génère deux nuages de points avec 20 et 100 points respectivement:

```
20 10 10 1 1

100 5 5 1 1
```

La figure d'introduction a été produire à partir de ce fichier.

La classe principale est KMeans. Elle comprend 3 méthodes virtuelles :

- fit : prend un ensemble de points et trouve le centre des groupes.
- classify : prend un ensemble de points et associe le groupe le plus proche.
- compare : comparaison de deux assignations de groupes, utilisé pour évaluer la précision.

Vous devez surcharger ces 3 méthodes et les paralléliser dans la classe KMeansParallel.

Voici les classes et les structures de données auxiliaires.

- Point : juste un point 2D, avec une fonction de distance entre deux points.
- KMeansClusterDefinition : représente l'information requise pour définir un nuage de points.
- DataSet : méthodes pour générer des points à l'aide des définitions. Les points peuvent être générés de manière aléatoire ou sur la circonférence d'un cercle. Nous allons utiliser les nuages aléatoires.
- start_point_random() : choisi des points aléatoirement pour initialiser l'algorithme
- make_cluster_mapping() : l'algorithme peut associer les points au bon groupe, mais le numéro du groupe peut être différent des numéros de groupe de la référence d'une exécution à l'autre. Par exemple, à la figure 1, il se peut que les chats obtiennent le groupe 0 ou 1. Cette méthode associe les numéros de groupe réels à ceux retrouvés par l'algorithme.
- On utilise le type uint8_t (1 octet) pour représenter le numéro de groupe. L'implémentation supporte donc au maximum 256 groupes.

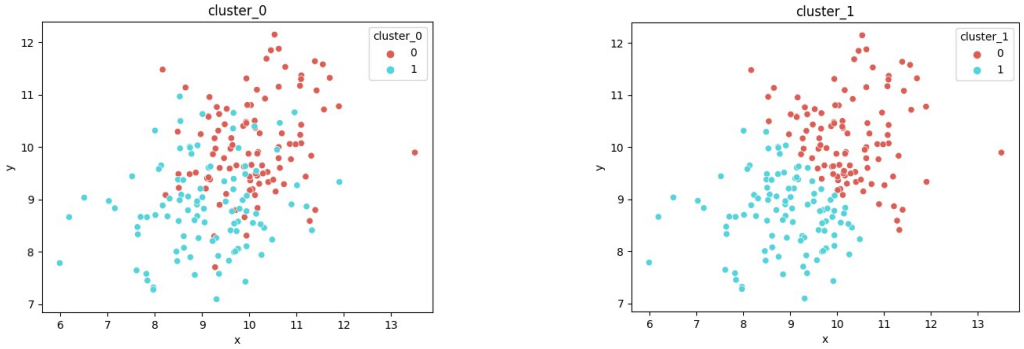
Des fichiers de jeu de données sont disponibles dans le répertoire « data ».

Voici un exemple pour l'exécution du programme principal :

```
./bin/kmeans data/overlap2.txt

# produit les fichiers kmeans_fit.csv et kmeans_classify.csv

python ../show.py kmeans_fit.csv

# produit deux graphiques avec les groupes de référence et les groupes établis
```

Les résultats sont écrits dans le répertoire courant. Vous pouvez visualiser le résultat avec le script Python

show.py. Le fichier requirements.txt avec les dépendances se trouve dans le répertoire des sources. Le nuage de point de départ et le groupe assigné par l'algorithme sont générés.



On peut voir à gauche le nuage de point initial. Dans ce cas-ci, les échantillons des deux groupes sont superposés. À cause de cela, l'algorithme des k-moyennes divise en deux groupes, mais certains points sont assignés au mauvais groupe. Pour cet exemple, la précision baisse à environ 75%.

N'utilisez PAS l'exécutable kmeans pour les bancs d'essai. Le problème est qu'il y a beaucoup d'entrées-sorties (écriture de fichiers de points) et pour cette raison, ce n'est pas nécessairement représentatif de l'accélération des parties sur lesquelles vous allez travailler. Utilisez plutôt l'exécutable kmeans_benchmark, dans lequel vous n'écrirez pas les données sur disque, vous allez simplement calculer la précision pour plusieurs ensembles de données.

Décompression et compilation

Voici les étapes à suivre pour décompresser et compiler l'archive du code source de départ. La procédure est la même pour tous les laboratoires du cours INF5171.

```
# Décompressez l'archive

$ unzip inf5171-tp2-21.8.1-Source.zip

$ cd inf5171-tp2-21.8.1-Source/

# Configurez les sources

$ mkdir build

$ cd build

$ cmake .. -DCMAKE_BUILD_TYPE=Debug
```

```
# Compilez les sources

$ make

# Exécutez les tests

$ make test
```

Informations techniques

- Répondez directement dans le questionnaire ODT avec Libre Office.
- Remettez le questionnaire en format ODT.
- La correction se fera directement dans le document électronique.
- Le questionnaire rempli et le code doivent être remis par Moodle dans une archive ZIP.
- Une seule personne de l'équipe doit effectuer la remise sur Moodle.
- Pour la remise, inscrivez les matricules dans le fichier CmakeLists.txt à la racine et produisez l'archive ZIP avec la commande `make dist`. Le ou les matricules doivent apparaître dans le nom du fichier ZIP séparé par un tiret.
- Vérifiez que l'archive que vous remettez contient tous les fichiers nécessaires.
- Pour obtenir les mesures de performance, compilez avec l'optimisation -DCMAKE_BUILD_TYPE=Release.
- Recommencez une expérience si vous jugez que les données sont anormales.
- Utilisez un ordinateur avec au moins 2 coeurs pour obtenir vos résultats.

3

1 Analyse préliminaire [4 pts]

Identifiez les structures appropriées pour la parallélisation des 3 méthodes virtuelles de la classe KMeansParallel.

TBB
tbb::parallel_for tbb::parallel_reduce tbb::parallel_for_each tbb::parallel_scan

Tableau 1: Éléments suggérés pour l'implémentation

1.1 Indiquez les fonctions que vous allez utiliser et expliquez vos choix. /4pt

Pour la fonction fit() la première partie qui trouve le centre le plus proche, il semble approprié d'employer tbb::parallel_for considérant qu'il s'agit d'un tâche indépendante par élément pour la boucle externe.
Pour la deuxième partie qui met à jour les centres, j'ai supposé qu'une reduction serait possible mais je n'ai pas réussi à l'implémenter.
Puis pour la dernière boucle de la fonction fit() j'ai à nouveau pu employer parallel_for pour optimiser la vitesse, techniquement reduce aurait pu fonctionner j'imagine en séparant la sommation des valeurs x et y des points en deux boucles mais comme parallel_for fonctionnait et ne nuisait pas à la précision des résultats je suis resté pour l'option la plus rapide.

Pour la fonction classify() encore une fois la meilleure option niveau accéleration tout en gardant des résultats valides semblait être parallel_for. Techniquement ça allait un peu à l'encontre de mon interprétation de comment la méthode fonctionnait. Puisque les itérations de la boucle interne n'étaient pas indépendant. Mais comme les résultats de test étaient bons avec parallel_for je m'y suis tenu.

Pour la dernière méthode compare() j'ai pu appliquer parallel_reduce pour conserver des résultats valides, puisqu'il s'agissait d'une somme.

2 Implémentation [8 pts]

Réalisez l'implémentation. Assurez-vous que le test fonctionne avec la commande « make test ».

2.1	Implémentation conforme de la fonction `KmeansParallel::fit()`.	/4pts
2.2	Implémentation conforme de la fonction `KmeansParallel::classify()`.	/2pts
2.3	Implémentation conforme de la fonction `KmeansParallel::compare()`.	/2pts

3 Analyse de la performance [8 pts]

Dans cette section, vous devez mesurer l'accélération de votre algorithme parallèle par rapport à la version série.

Créez un banc de mesure de performance dans le fichier `kmeans_benchmark.cpp`. Votre banc d'essai doit mesurer le temps d'exécution série et en parallèle. Voici les spécifications :

- Utilisez le fichier « data/large10.txt » comme source de données.
- Calculez seulement la précision de l'algorithme. N'enregistrez PAS la liste de points.
- Obtenez le temps d'exécution pour10 ensembles de données différents. Ne comptez PAS le temps de génération des ensembles de données d'entrainement et d'évaluation. Calculez la moyenne du temps d'exécution et de l'accélération.
- Attention de bien réutiliser le même ensemble de points de départ, sinon vos résultats ne seront pas représentatifs.
- Utilisez le maximum de CPU de votre ordinateur pour la version parallèle. Il n'est pas requis de faire un graphique d'accélération.

Assurez-vous de compiler avec l'option -DCMAKE_BUILD_TYPE=Release et que rien d'autre ne soit en exécution sur votre ordinateur au moment de rouler le banc d'essai.

Écrivez les résultats dans le fichier benchmark.csv avec, une ligne par ensemble de données, le temps d'exécution série, le temps d'exécution parallèle, l'accélération et l'efficience.

3.1	Réalisation du code de mesure de performance.	/4pt
3.2	Considérant que ma moyenne de temps d'éxécution pour 10 ensembles en version série est de 3.85. sec. et que je travaille avec 8 fils d'éxécutions, cela donne, sans surcout et idéalement, 0.48 sec. Comme j'ai 1.29 sec. en moyenne pour la version parallèle, je peux estimer avoir au moins 0.33 pour la proportion parallèle, donc 33%.	/2pts

3.3 # /2pts

Comme on peut voir dans cette échantillon de benchmark.csv l'accélération semble être meilleure pour un plus grand nombre d'itération, comme on pouvait s'y attendre.

Temps série, Temps parallèle, Accélération, Efficience

5.9 1.97 3 0.38

1.82 0.61 2.99 0.38

1.39 0.47 2.96 0.37

1.41 0.48 2.94 0.37

7.19 2.36 3.05 0.39

7.6 2.53 3.01 0.38

8.82 2.92 3.03 0.38

0.76 0.28 2.72 0.34

2.32 0.78 2.98 0.38

1.24 0.42 2.96 0.37

Moyenne temps série : 3.85

Moyenne temps parallèle : 1.29

Moyenne accélération : 2.97

4 Correction

Section 1 [4 pts]	
Section 2 [8 pts]	
Section 3 [8 pts]	
Note finale sur 20	