الجـــمهوريـــة الجزائريـــة الديمـــقراطيـــــة الشـعبيـة
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵜⴰⵣⵣⴰⵢⵔⵉⵜ ⵜⴰⵎⴰⵙⵙⴰⵡⵉⵜ ⵜⴰⵖⵓⵔⴰⵜ
République Algérienne Démocratique et Populaire
وزارة التعليـــم العـــالي والبـــحث العلمـــي
ⴰⵖⵡⴰⵙ ⵏ ⵜⵓⵙⵎⵔⵙⵉⵜ ⵜⵉⵏⵣⵡⴰⵏ ⴷ ⵜⵓⵙⵙⵏⵉⵡⵉⵏ
Ministère de l’Enseignement Supérieur et de la Recherche Scientifique

ECOLE NATIONALE SUPÉRIEURE D’INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
ⴰⵖⵓⵔⴰⵙ ⴰⵖⵡⵍⴰⵍ ⵜⵉⵏⵣⵡⴰⵏ ⵏ ⵜⵓⵙⵙⵏⵉⵡⵉⵏ
École nationale Supérieure d’Informatique

**Module :** High Performance Computing ( HPC )

**2CS SIQ 1**

# Thème :
# Parallélisation de l’algorithme k-means avec CUDA

**Réalisé par :**

- LARDJANE Ikram
- NOUALI Asmaa

**Année universitaire 2024/2025**

# 1. Introduction :

Dans le domaine du traitement de données massives, la performance computationnelle est devenue un enjeu crucial. Les algorithmes traditionnels de clustering, tels que K-means, sont particulièrement gourmands en ressources de calcul lorsqu'ils sont appliqués à des jeux de données de grande envergure. Cette problématique nous a conduits à explorer des techniques de parallélisation avancées pour optimiser les performances algorithmiques.
L'objectif principal de ce TP est de démontrer comment les technologies modernes de calcul parallèle, notamment CUDA peuvent significativement améliorer l'efficacité de l'algorithme K-means. En exploitant la puissance de calcul massivement parallèle des GPU, nous cherchons à réduire le temps de traitement et à augmenter la scalabilité de cet algorithme de clustering.
Notre étude comparative portera sur trois implémentations différentes :

- Une implémentation séquentielle
- Une implémentation parallèle utilisant Pthreads
- Une implémentation parallèle optimisée avec CUDA

# 2. Présentation de l'algorithme K-means :

L'algorithme K-means est une méthode de **clustering non supervisé** largement utilisée dans le domaine de l'apprentissage automatique pour partitionner un ensemble de données en **k groupes** ou clusters homogènes, où chaque groupe contient des éléments ayant des caractéristiques similaires. L'objectif principal de l'algorithme est de minimiser la variance intra-cluster, c'est-à-dire que les points au sein d'un même cluster doivent être aussi proches que possible. C'est une approche particulièrement appréciée dans des domaines comme le marketing, où l'on cherche à segmenter les clients selon des comportements ou des caractéristiques communes pour mieux cibler les actions commerciales.

## pseudo-algorithme de k-means :

1. Initialisation des centroïdes
    - Sélectionner aléatoirement k points de données dans l'ensemble de données comme centroïdes initiaux.
2. Répéter jusqu'à convergence :
    - Pour chaque point de données dans l'ensemble de données :
      i. Calculer la distance entre le point de données et chaque centroïde.
      ii. Assigner le point de données au cluster ayant le centroïde le plus proche.
3. Mise à jour des centroïdes:
    - Pour chaque cluster :
      i. Calculer le nouveau centroïde en prenant la moyenne de tous les points de données qui lui ont été assignés.
4. Critères de convergence :

- Vérifier si les centroïdes ont cessé de se déplacer (c'est-à-dire que les changements dans les positions des centroïdes sont en dessous d'un certain seuil).
- Si les centroïdes ont convergé, terminer l'algorithme.
- Sinon, répéter les étapes 2 et 3.

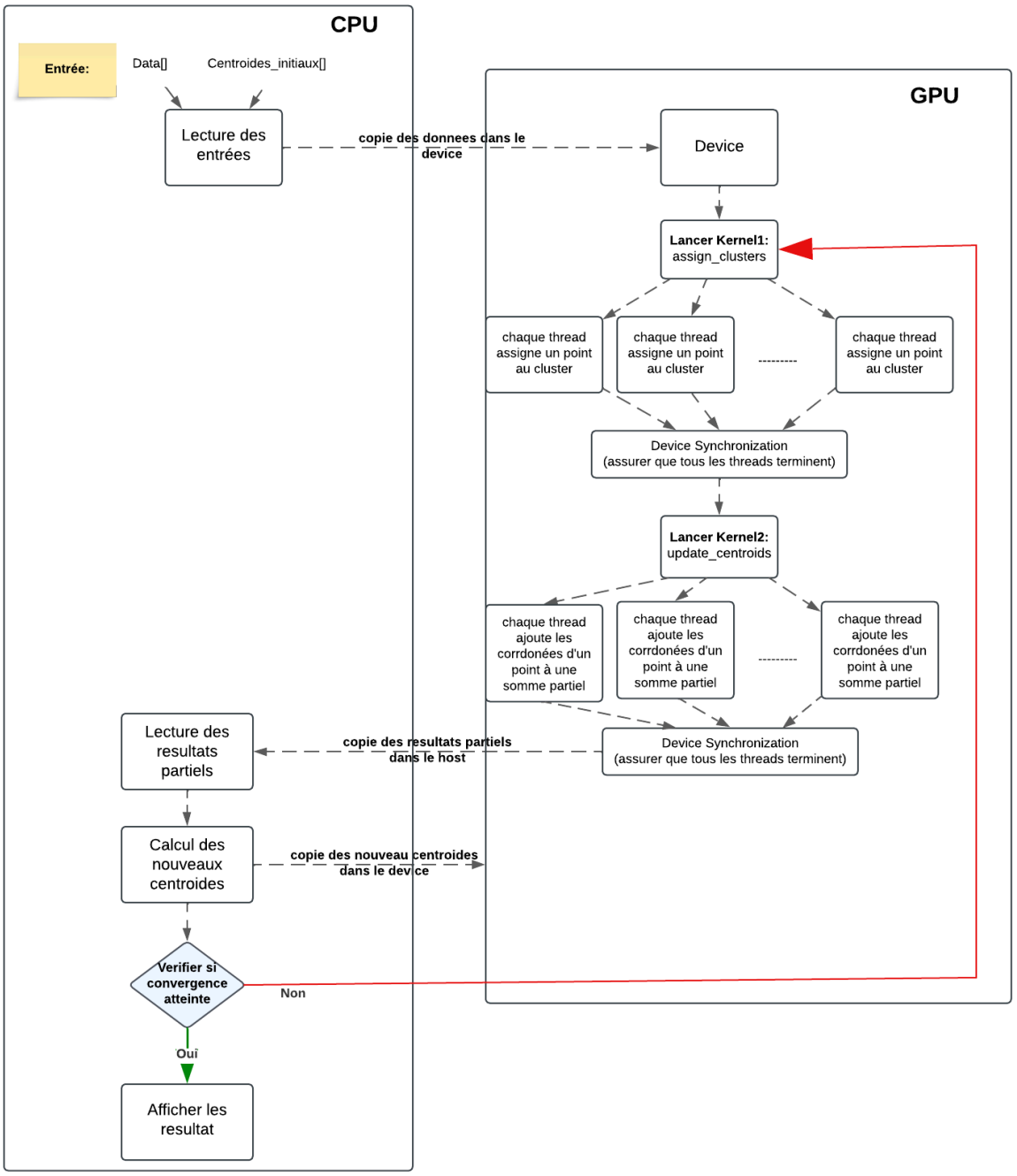
Fin de l'algorithme

# 3. Stratégie de parallélisation avec CUDA:

La stratégie de parallélisation de l'algorithme K-means avec **CUDA** repose sur deux principaux kernels :

1. **Parallélisation de l'assignation des clusters** :
   Chaque thread est responsable de l'assignation d'un point de données à un cluster en calculant la distance entre le point et les centroids. Cette opération est effectuée en parallèle pour tous les points du dataset.
2. **Parallélisation de la mise à jour des centroids** :
   Les threads calculent la somme des coordonnées des points dans chaque cluster. Grâce à l'utilisation des opérations atomiques, plusieurs threads peuvent mettre à jour les centroïdes de manière sécurisée et sans conflit.

# 4. Schéma de la solution parallèle :




# 5. Analyse des résultats :

Dans cette partie, nous comparons les temps d'exécutions de l'algorithme K-means appliqué sur des datasets contenant respectivement 1 million, 2 millions et 5 millions de points, en utilisant trois approches différentes : l'implémentation séquentielle, l'implémentation avec Pthreads, et l'implémentation avec CUDA.

Les tests effectués divisent les données en 2 clusters, en utilisant comme centroïdes initiaux les points (31, 32) et (33, 34).

## a. Implémentation séquentielle:

| **Nombre de points dans le dataset** | 1000000 | 2000000 | 5000000 |
|---|---|---|---|
| **Temps d'exécution (s)** | 6.146 | 11,25 | 40,21 |

Étant donné que l'algorithme passe par plusieurs itérations et que chaque opération est effectuée séquentiellement, l'exécution sur un prend un temps assez long et augmente en augmentant la taille des données .

## b. Implémentation avec Pthreads:

Résultat pour un dataset contenant 1000000 points :

| **Nombre de threads** | 2 | 4 | 8 | 16 | 24 |
|---|---|---|---|---|---|
| **Temps d'exécution (s)** | 1.42 | 1.06 | 0.7 | 0.941 | 1.08 |

Résultat pour un dataset contenant 2000000 points:

| **Nombre de threads** | 2 | 4 | 8 | 16 | 24 |
|---|---|---|---|---|---|
| **Temps d'exécution (s)** | 2,46 | 1,69 | 1,22 | 1,29 | 1,35 |

Résultat pour un dataset contenant 5000000 points:

| **Nombre de threads** | 2 | 4 | 8 | 16 | 24 |
|---|---|---|---|---|---|
| **Temps d'exécution (s)** | 8,01 | 6,36 | 4.06 | 4,4 | 4,7 |

La solution parallèle utilisant Pthreads montre une réduction significative du temps d'exécution par rapport à l'implémentation séquentielle, avec un temps d'exécution qui décroît en augmentant le nombre de threads de 2 à 8. Cependant, au-delà d'un certain nombre de threads, le temps d'exécution commence à réaugmenter. Cela s'explique par l'augmentation des coûts de synchronisation entre les threads.
En augmentant la taille des données, le temps d'exécution augmente mais reste toujours beaucoup plus petit que le temps séquentiel.

### c. Implémentation avec CUDA:

Résultat pour un nombre de threads par bloc = 512 threads :

| **Nombre de points dans le dataset** | 1.000.000 | 2.000.000 | 5.000.000 |
|---|---|---|---|
| **Temps d'exécution(s)** | 0,197 | 0.32 | 0.774 |

Les résultats obtenus avec l'implémentation CUDA montrent des performances nettement supérieures par rapport à l'approche utilisant Pthreads. Grâce à l'exécution massivement parallèle sur le GPU, le temps d'exécution reste très faible, même avec des datasets volumineux.Cela illustre la capacité du GPU à gérer efficacement des calculs intensifs,même lorsqu'il s'agit de traiter des millions de points.

## Conclusion :

L'étude montre que la parallélisation améliore considérablement les performances de l'algorithme K-means. L'approche séquentielle est inefficace pour de grandes quantités de données. L'implémentation avec Pthreads offre des gains significatifs, mais ces derniers sont limités par les coûts de synchronisation au-delà d'un certain nombre de threads. En revanche, l'implémentation CUDA, grâce à la puissance des GPU, se distingue par des temps d'exécution très courts et une excellente scalabilité, en faisant la solution la plus efficace pour le traitement de données massives.