Operating System Project 2023/2024

Write a C program to implement a parallel version of the K-means clustering algorithm for bidimensional data.

Background

K-means is an algorithm in unsupervised machine learning designed for grouping similar data points together. Its primary objective is to identify natural clusters within a dataset without prior knowledge of the groupings. By iteratively refining cluster assignments, K-means reveals underlying patterns and structures in the data.

Key Concepts

- Cluster: The algorithm organizes data points into distinct clusters, grouping those that share similarities.

- Centroids: K-means revolves around the concept of centroids, which are representative points at the centre of each cluster.

- Iterative Refinement: K-means iteratively refines its clusters by adjusting the position of centroids based on the proximity of data points.

Advantages

- Simplicity: K-means is easy to understand and implement, making it a popular choice for clustering tasks.

- Scalability: The algorithm is computationally efficient, allowing it to handle large datasets.

Considerations

- Predefined Number of Clusters: users must specify the number of clusters (K) beforehand, which can be a limitation in some cases.

- Sensitivity to Initialization: the results can be sensitive to the initial placement of centroids, affecting the quality of clustering. For this reason, most implementations require executing the algorithm several times. The aim of this project is to parallelize the re-execution of the k-means algorithm with different random centroids.

Look at https://hckr.pl/k-means-visualization/ for a two-dimensional visualization of the k-means clustering algorithm.

Specifications

The entire application consists of two executables: a **master** program that spawns worker processes and a **worker** program that runs the K-means algorithm.

MASTER

Input

The master program takes 4 inputs as command line arguments.

1. The first argument is the number of clusters K.
2. The second argument is the maximum number of processes to be spawned N.
3. The third argument is an IPC key.
4. The fourth argument is the path to a text file containing a dataset of 2D points of the form "x, y", one per line.
   For example:
   1,2
   3,4
   5,6
   7,8
   is a valid dataset containing four points with x coordinates 1, 3, 5, 7 and y coordinates 2, 4, 6, 8.

Output

The master program writes to file the centroids of the best clustering (with the lowest variance) following the format reported below:

centroid_x0, centroid_y0
centroid_x1, centroid_y1
centroid_x2, centroid_y2
…
centroid_xK, centroid_yK

where K is the number of clusters.

Execution flow

The master process implements the following execution flow:

1. Read the input arguments
   - Read the number of clusters K
   - Read the maximum number of processes N
   - Read the IPC key
   - Read the path to the dataset file

2. Handle the dataset file
   - Read the dataset file line by line
   - For each point “x, y”, parse the x and y coordinates and store them in an IPC (System V) shared memory segment. The shared memory segment should be large enough to store all the points in the dataset. Use the given IPC key to create the shared memory segment.

3. Run the clustering

The master distributes the computational load of the clustering algorithm to the worker processes. The master must spawn N worker processes.
After each iteration of the clustering algorithm, the worker process sends the variance of the clusters and the found centroids to the master process. The master keeps track of the best clustering so far (lowest variance). The master process must terminate when the variance of the best clustering does not improve for more than 1000 iterations: the master received 1000 messages containing a variance higher that the current minimum.

In particular, the master process must implement the following features:

- The master process uses fork() to spawn worker processes, then uses exec (the designer must decide which type of exec to use) to replace the worker process with the worker executable
- The master process receives the variance and centroids from the worker processes using an IPC message queue. The message queue must be created using the IPC key.
- If the variance of the best clustering does not improve for 1000 iterations, the master process must terminate all the executing worker processes by sending them a SIGINT signal. The master process must always gather the termination status of the worker processes (using wait or waitpid).
- The master dumps to file the centroids according to the guidelines provided above.
- If the master receives a SIGINT signal, it frees all the allocated resources and terminates.

WORKER

Input

The worker program takes 3 inputs as command line arguments.
The first argument is an IPC key.
The second argument is number K of centroids to be generated randomly.
The third argument is the number of points in the dataset (number of lines in the input file)

Output

The worker process uses an IPC message queue to send the variance and the centroids of the clusters to the master process.

Execution flow

The worker process implements the following execution flow:

1. Read the input arguments
   - Read the IPC key
   - Read the number of clusters K
   - Read the number of points in the dataset

2. Handle the IPCs
   - Read the dataset from the shared memory segment: attach it to the shared memory segment using the IPC key. The memory segment must be opened in read-only mode.
   - Connect to the master's message queue using the IPC key
   - Add a custom signal handler for SIGINT: if the worker receives a SIGINT signal, it must detach from the shared memory segment and exit with code 0.

3. Run one iteration of the clustering algorithm (until convergence)

   - Run the clustering algorithm:

     I. Initialization: randomly initialize K cluster centroids. Each centroid is a data point of the input dataset (make sure not to have repeated centroids pointing to the same data point; furthermore, remember to randomize the seed to avoid having each worker performing the same identical task).

     II. Assign Data Points to Clusters: For each data point, calculate its Euclidean distance (https://en.wikipedia.org/wiki/Euclidean_distance) to each centroid.

Assign the data point to the cluster whose centroid is the closest. Note that it might happen that no points are assigned to some of the centroids; in this case, you need to reinitialize the affected centroids with a random point of the dataset.

III. Update Cluster Centroids: Recalculate the centroid of each cluster by taking the mean of all the data points assigned to that cluster. Repeat Steps II and III until convergence. Convergence can be determined by checking if the centroids did not change from the previous update. The final centroids represent the centers of the clusters.

- Send the variance and the centroids of the clusters to the master process using the message queue. The variance can be computed in several ways, one way is:

$$\text{variance} = \frac{\sum_{i=0}^{nPoints} \text{euclideanDistance}(p_i, c_i)^2}{nPoints}$$

where p_i is a 2D point of the dataset and c_i is the centroid closest to p_i and nPoints is the number of datapoint in the dataset.
However, it does not matter how it is calculated as long as the algorithm moves towards the right direction.

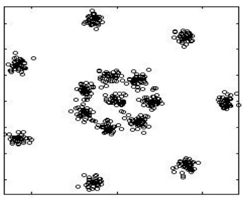
4. Repeat Step 3 until receiving a SIGINT signal either from the master process or from the user (Ctrl+C).

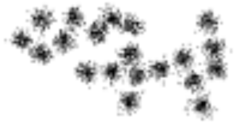
Testing

How do you know if the implementation is “correct”?
I prepared 4 datasets to test your application.

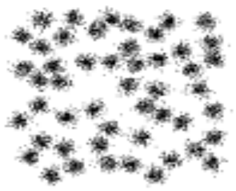
- toy.csv : N=9, K=3.
  The centroids should be:
  1005, 1005
  505, 505
  5, 5

- r15.csv : N=600, K=15





- a1.csv : N=3000, K=20



- a3.csv: N=7500, K=50

Use the “plot.sh” to check if the final centroids correctly single out the clusters of data points in the dataset.
Example of running the script: “bash plot.sh dataset.csv centroids.csv”.
(you will need to have ‘gnuplot’ installed on your machine: run “**sudo apt-get install gnuplot”** to install it)

Additional Remarks

- Every system call must be checked for errors
- The project must contain a Makefile to compile the master and the worker executables (there are several examples of Makefile in the exercises solved throughout the course)
- The project must be available on GitHub https://github.com and must be downloadable using “git clone <URL-to-your-project>”.

Note sull’esame di laboratorio

Consegna

- Ogni studente che indenta superare la parte di laboratorio deve consegnare l’elaborato entro le 23:59 del giorno precedente dell’orale. Il progetto può essere fatto in gruppi. In ogni caso, ogni studente deve consegnare il progetto e partecipare all’orale individualmente.
- Il progetto si intende consegnato inviando a samuele.germiniani@univr.it il link al repository GitHub del vostro progetto.
- Questo progetto è valido solo fino all’ultima sessione dell’anno accademico 2023-2024 (Febbraio 2025).

Metodo di valutazione

Lo scopo dell’orale è determinare se lo studente è in grado di padroneggiare le conoscenze acquisite durante il corso.

- Se il progetto non è su GitHub, non ha un Makefile o non compila ed esegue sulle macchine del laboratorio Delta (Linux - Ubuntu), allora il voto per la parte di laboratorio scende automaticamente a 0; lo studente può riprovare l'esame durante l’appello successivo. Alternativamente, lo studente può acquisire fino a 33 punti che faranno media con il voto di teoria. L’esame di laboratorio si intende superato solo acquisendo almeno 16 punti. Se non si è in grado di recarsi in aula Delta per “provare” il proprio progetto, l’università mette a disposizione delle aule virtuali (https://virtualab.univr.it/guacamole/#/) con la stessa configurazione dei laboratori dell’università.
- Max 13 punti per il progetto.

    - Tutte le specifiche sono state implementate? (5 punti); il progetto esegue correttamente sui 4 dataset? (4 punti); lo studente sa cosa ha implementato o il progetto è stato fatto totalmente co AI e/o dai compagni di gruppo? (4 punti).
- Max 10 punti per domanda sui concetti di laboratorio.
    - Alcuni esempi di domande possono essere: Quali funzioni si usano per creare e utilizzare i semafori? Qual è lo scopo di system call X? Spiegare i parametri e valori di ritorno di system call X. In generare non verrà chiesto niente di più (niente di meno) di quanto riportato nelle dispense sul sito del corso.
        - 10 punti per risposta corretta e completa (5 punti per risposta parzialmente corretta o parzialmente completa)
- Max 10 punti per modifica in tempo reale del progetto.
    - In questa parte, lo studente avrà a diposizione tutto il materiale del corso: codice, dispense e il system reference manual (man). Alcuni esempi possono essere: modifica il codice in modo tale da usare le PIPE invece che le code di messaggi; aggiungi l'utilizzo di semafori per proteggere una risorsa contesa; invia segnale X a processo Y.
    - Lo studente avrà 20 minuti di tempo per completare il task.
    - 10 punti se la vostra implementazione compila ed esegue correttamente (5 punti se l'idea è corretta ma il vostro codice non compila e/o non esegue correttamente)