Parallel Kmeans

Ellouze Skander

April 2019

Introduction

Introdution

Given a dataset x_i, the algorithm aims to separate it into **K different Clusters** C_j. This is done by generating **K centroids** c_j and assigning each point of the dataset to the cluster of the nearest centroid. The algorithm aims then to minimize :

$$J = \sum_{j \in \{1,..,K\}} \sum_{x_i \in C_j} \|x_i - c_j\|^2 \textit{ is minimal}$$

This problem is an **NP-problem**, but the K-Means Algorithm gives usually a satisfying solution.

Serial KMeans

Serial Kmeans

The serial algorithm consists of four steps :

1. Generate random centroids.
2. Assign each point to the cluster of the nearest centroid : $x \in C_k$ such that $k = argmin_j(\|x - c_j\|^2)$
3. compute the new centroids : $c_j = \frac{1}{|C_j|} \sum_{x \in C_j} x$
4. These two last steps are repeated until convergence of J $(|J_{step+1} - J_{step}| \leq \epsilon)$

$$\Rightarrow T_S^* = O(N_{steps}KN)$$

Serial KMeans : Illustration

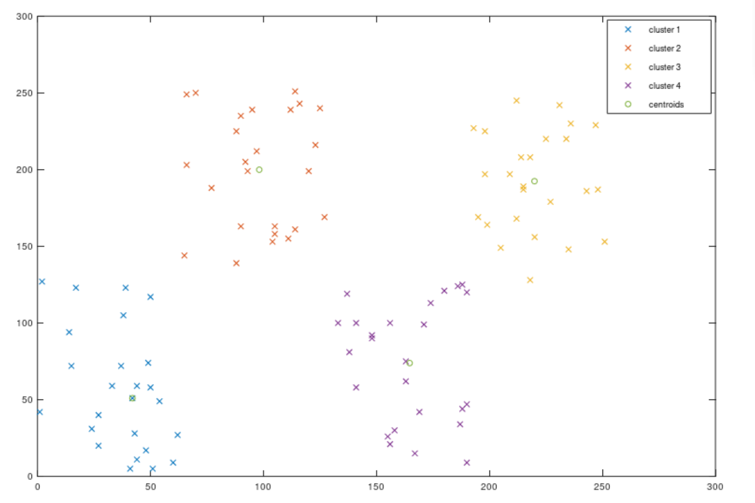

Fig. 1: Illustration of KMeans

Parallel Kmeans

Parallel Kmeans

1. Centroids Initialization : Each Process choose randomly K centroids.
2. Assigning each point to the cluster : Each Process must have the global centroids.
3. Computing the new Centroids : This is done by recursive doubling, see next slide.
4. Computing the global cost : Assuming each process has computed the local cost $J_{local} = \sum_{j\in\{1,..,K\}} \sum_{x_i \in C_j} \|x_i - c_j\|^2$ it's possible to sum all costs by recursive doubling : $J = \sum_{0\leq p\leq P-1} J_{local}$.

Computing New Centroids : Recursive Doubling

- Suppose we have 2 processes.
- Each of them has already computed its new local centroids $c_j^p = \frac{1}{m_p} \sum_{x_i \in C_j^p} x_i$.
- The global centroids should be $c_j = \frac{1}{m} \sum_{x_i \in C_j} x_i = \frac{1}{m^0 + m^1} \sum_{x_i \in C_j^0 \cup C_j^1} x_i = \frac{1}{m^0 + m^1} (\sum_{x_i \in C_j^0} x_i + \sum_{x_i \in C_j^1} x_i) \Rightarrow c_j = \frac{m_j^1 c_j^1 + m_j^0 c_j^0}{m_j^1 + m_j^0}$.
- Each process have to send (m_j^p, c_j^p).

Computing New Centroids : Recursive Doubling

We Suppose that $P = 2^D$.

Algorithm 1: Recursive doubling for new centroids

Result: New Global Centroids

for $d = 0 : D - 1$ **do**

send($(m_j, c_j)_{0<K}$,bitflip(p,d))

receive($(m', c')_{0<K}$,bitflip(p,d))

$\forall j,\ c_j = \frac{m_j * c_j + m' * c'}{m_j + m'}$

$\forall j,\ m_j = m_j + m'$

end

KMeans : Parallel Algorithm

Algorithm 2: KMeans Parallel Algorithm

Result: K centroid/clusters

Randomly choose K points as the initial centroids : c_j

while $|J_{step+1} - J_{step}| \leq \epsilon$ **do**

Assign each point x to the cluster corresponding to the nearest centroid : $\forall x \in D_p, x \in C_k$ such that $k = argmin_j(\|x - c_j\|^2)$

Compute the local new centroids and their weights : $m_j = |C_j|$ & $c_j = \frac{1}{m_j} \sum_{x \in C_j} x$

Compute the global new centroids with recursive doubling.

Compute the local cost : $J_{local} = \sum_{j \in \{1,..,K\}} \sum_{x_i \in C_j} \|x_i - c_j\|^2$

Compute the global cost with recursive doubling : $J = \sum_{0 \leq p \leq P-1} J_{local}$

end

In practice, we use the built-in function **MPI_Allreduce** to compute the global cost.

Parallel KMeans

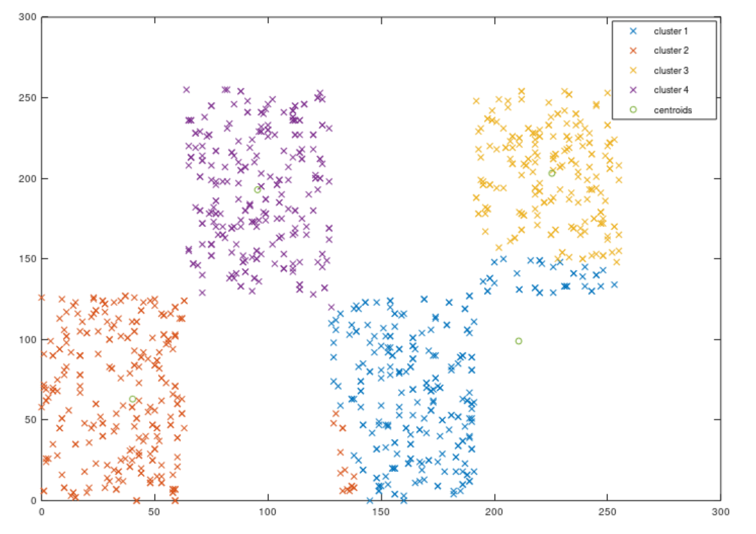

Fig. 2: Parallel KMeans using $N = 1000$ and K=4

Performance Study

Speedup

Computation Time (For a single step):

- Assigning points to clusters : $t = 8 * \frac{N}{P} K t_a$
- Computing the local centroids : $t = 10 * \frac{N}{P} K t_a$
- Computing the local cost : $t = 7 * \frac{N}{P} t_a$
- Computing the global centroids(Recursive doubling): $t = 12 * log(P) K t_a$
- Computing the global cost(Recursive doubling): $t = log(P) t_a$

Speedup

- Communication Time (For a single step):
  - Compute the global centroids recursive doubling : $t = log(P)(t_{st} + 2Kt_{data})$
  - Compute the global cost recursive doubling : $t = log(P)(t_{st} + t_{data})$

$$\Rightarrow T_P = N_{steps}((18\frac{N}{P}K+7*\frac{N}{P}+(1+12K)log(P))t_a+2log(P)t_{st}+(2K+1)log(P)t_{data}$$

$$\Rightarrow T_P = O(N_{steps}(\frac{N}{P}K + Klog(P)))\ if\ 1 \ll N, P$$

Speedup

$$S_P = \frac{(18K+7)Nt_a}{(18\frac{N}{P}K + 7 * \frac{N}{P} + (1+12K)log(P))t_a + 2log(P)t_{st} + (2K+1)log(P)t_{data}}$$

$$= P\frac{1}{1 + \frac{(1+12K)Plog(P)}{(18K+7)N} + \frac{2Plog(P)}{(18K+7)N}\frac{t_{st}}{t_a} + \frac{(18K+7)Plog(P)}{(18K+7)N}\frac{t_{data}}{t_a}}$$

$$\Rightarrow S_p = \frac{P}{1 + A\ P\ log(P)}$$

- If P is fixed, increasing N will increase the Speedup.
- If N is fixed, the speedup will degrade if P gets extremely large (communication overhead).

Experimental Results

Exp1: Large Inputs

- We run our algorithm for : $N = 10^8$, K=4 and $P \in [1..48]$
- We use **optimization O2** when compiling.
- The next figure shows the **Speedup**.
- The curve accelerates quickly at the beginning.
- The Speedup seems to stagnate after P = 12.

Exp1: Large Inputs

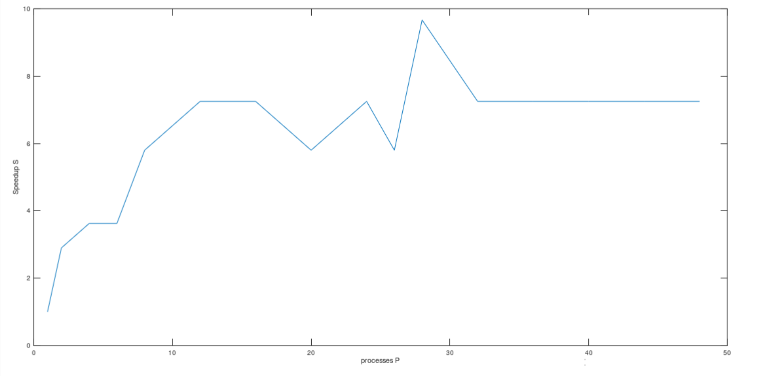

Fig. 3: $S_P = f(P)$ using $N = 10^8$ and K=4

Exp2: Communication Overhead

- We run our algorithm for : $N = 1000$, K=2 and $P \in [1..48]$
- The next figure shows the Speedup.
- The first part is similar to exp1.
- For P>30, the Speedup drops from 2.30 to 1.18 (Confirms the Theory).

Exp2: Communication Overhead

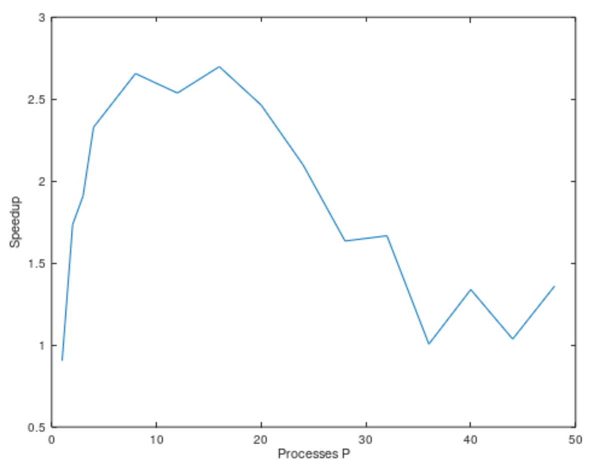

Fig. 4: $S_P = f(P)$ using $N = 10^8$ and K=4

Reducing the communication overhead

Solution 1 : Merging the two communication operations

- Merging the computing cost and the computing new centroids operations.
- The algorithm will compute the cost of the previous iteration.
- This will add an iteration to the algorithm.
- The Setup Time of Communication is divided by 2.

Solution 1 : Results

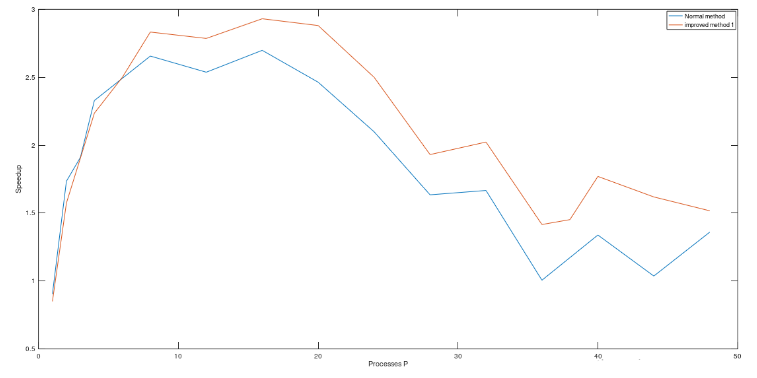

Fig. 5: Speedup Curves for normal method & improved method 1

Solution 2 : Gather results every two iterations

- limit the communication to one iteration out of two.
- one iteration out of two the program will only calculate the local centroids.
- This will also add some iterations to the algorithm.
- The communication time is divided by 2.

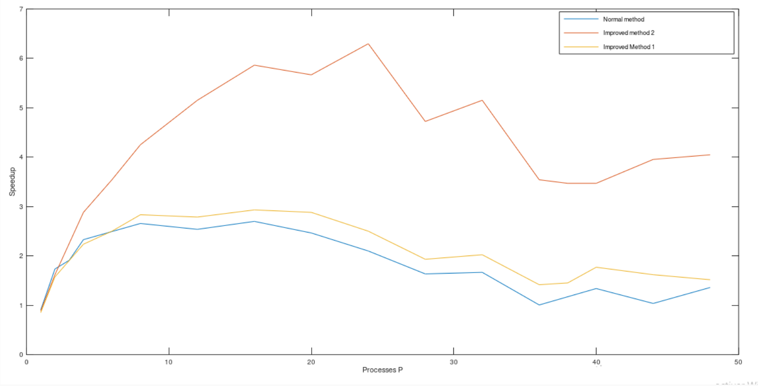

Fig. 6: Speedup Curves for normal method, improved method 1 improved method 2

Conclusion

- All the curves have the same shape.
- Modifications made it possible to increase the speedup and to dampen the fall
- The results are only **valid for a small input N** (N≤100P). Otherwise, the results are the opposite.

Conclusion

Conclusion

- The KMeans Algorithm gives a satisfying solution to the Clustering Problem.
- The KMeans Algorithms is easily parallelizable.
- In the case of communication overhead, reducing the number of communication operations (at the expense of computation time) may accelerate the Algorithm.