Parallel k-means Clustering

SF2568 – Parallel Computations for Large-Scale Problems

Lukas Bjarre **Gabriel Carrizo**
lbjarre@kth.se **carrizo@kth.se**

Abstract

This report details the implementation and analysis of a parallelisation of the k-means clustering method. A serial version of Lloyd's algorithm for computing the k-means clustering was analysed and a parallelisation stategy of dividing data points over multiple processes was proposed. The parallel implementation was a single program multiple data scheme on distributed memory using the Message Passing Interface (MPI) standard. The implementation was tested on the super computer cluster Tegner located at the PDC Center for High Performance Computing at the Royal Institute of Technology in Stockholm. <something about the results>.

1 Introduction

1.1 Clustering

Clustering is described in [1] as the grouping of similar objects. Hartigan explains that two objects are similar if their separate observations could be of the same object. Using the eample of planets, Hartingan explains that two observed objects are planets if their observations are similar enough in significant features. Consider for example the features: shape and light intensity, versus color to discern planets from stars.

1.2 Parallel Computations

Today, clustering is an important tool for automatic grouping or preprocessing of data in data science. With the emergence of *big data* where the number of data points and dimensions of the data are very large these algorithms can be computationally costly. Traditionally these algorithms are written in a linear manner, where *instruction A* is followed by *instruction B*, even if they operate on disjoint data. Modern processors with multiple cores have the capability of executing multiple instructions, or processes, at the same time. This enables some problems to be computed more efficiently, however one prerequisite is the possibility of dividing the problem into disjoint subsets that can be processed individually. This class of problem falls into the parallel category whilst the opposite is classed as a sequential problem. A sequential problem is one in which each section of the code is dependent on the previous sections output in order to proceed with the computations. Many problems are partly parallel and sequential and can be subdivided into both categories. Parallel computations are more complex than sequential and are usually preserved for problems where the computation time is important or where the computation time can be reduced substantially.

Parallel algorithms can be divided into two sub classes: shared vs. distributed memory. Distributed memory implementations require communication between processes where data and results of computations are exchanged or gathered. Although these transactions take relatively long time to perform, distributed implementations are prefered as shared memory implementations suffer from problematic *race conditions*.

1.3 Aim

This project proposes a distributed parallel implementation of a clustering algorithm to reduce the computation time for more efficient processing of data. The project aims to reduce the computation time of a modern interpretation of a fairly simple clustering algorithm *K-means clustering* as a potential stepping stone to more complex algorithms.

2 Theory

2.1 k-means clustering

k-means clustering is a data clustering method which clusters input data from the data set $\mathcal{X}$ into k different classes. The classes are represented by the class means $\boldsymbol{\mu}_i$ and points are considered to be in a class S_i if the squared distance to the class mean is the minimum compared to the squared distance to the other class means. Formally:

$$S_i = \{\boldsymbol{x} \in \mathcal{X} : \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2 \leq \|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2, \forall 1 \leq j \leq k\}$$

A clustering method aims to find a selection of these classes $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ which divides the data points in some favorable way. k-means finds the placement of the class means by minimization of the summed squared distance of all class points to the class mean for all k classes:

$$\mathcal{S}_{k\text{-means}} = \arg\min_{\mathcal{S}} \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in S_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2$$

A common algorithm to find this is Lloyd's algorithm [2], which iteratively classifies points according to current class means and updates them with the average of all classified points until convergence. Algorithm 1 describes this procedure in pseudocode. This algorithm requires multiple loops for each iteration. The most costly part is the classification of all data points, which requires computation on all d dimensions of the data set for all k classes over all n points, giving each iteration a complexity of $O(nkd)$.

Algorithm 1: Lloyd's algorithm for finding the k-means clustering class means.

Input: Data points $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ with $\boldsymbol{x}_i \in \mathbb{R}^d \; \forall \boldsymbol{x}_i \in \mathcal{X}$, number of clusters k
Output: Class means $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k$

initialize $\bar{\boldsymbol{\mu}}_i$ on random points in $\mathcal{X}$ $\forall i = 1, \ldots, k$
while $\boldsymbol{\mu}_i \neq \bar{\boldsymbol{\mu}}_i \; \forall i = 1, \ldots, k$ **do**
  for $i = 1, \ldots, k$ **do**
    $\boldsymbol{\mu}_i \leftarrow \bar{\boldsymbol{\mu}}_i$
    $\bar{\boldsymbol{\mu}}_i \leftarrow 0$
  forall $\boldsymbol{x} \in \mathcal{X}$ **do**
    class $\leftarrow \arg\min_k \|\boldsymbol{x} - \boldsymbol{\mu}_k\|^2$
    $\text{count}_{\text{class}} \leftarrow \text{count}_{\text{class}} + 1$
    $\bar{\boldsymbol{\mu}}_{\text{class}} \leftarrow \bar{\boldsymbol{\mu}}_{\text{class}} + \boldsymbol{x}$
  for $i = 1, \ldots, k$ **do**
    $\bar{\boldsymbol{\mu}}_i \leftarrow \frac{\bar{\boldsymbol{\mu}}_i}{\text{count}_i}$
    $\text{count}_i \leftarrow 0$
return $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k\}$

2.2 Parallelisation strategy

Algorithm 1 shows a lot of possibility of parallelisation due to the large number of independent calculations. For every iteration all the points needs to be reclassified to the new means, for which every point is independent of the other points. Computing the new class means consist of adding up the classified points over all d dimensions as well as the total count, which are additions that can be done firstly on separate processors and finally reduced over all processors.

Overall we get a execution time for each iteration T_{iter} that with the number of processes P that roughly scales as

$$T_{\text{iter}} = t_{\text{calc}} \frac{nkd}{P} + t_{\text{comm}} k(d+1) \log P$$

Noteworthy for the analysis of this performance is that n will be much larger than k and d in most applications. k and d normally lie in the 10^1-10^2 range, while n can go up to 10^6-10^9

easily. The reduction of the calculation time will therefore affect the overall computation time more than the increasing communication time.

References

[1] John A Hartigan. Clustering algorithms. pages 1–14, 1975.

[2] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.