A

Project Report

On

“Implementation of K-Means Clustering Algorithm using Parallel Computation Cluster”

Submitted in partial fulfilment of the requirements

of the UG Fellowship (URFU)

by

Ms. Bhosale Divya R.
Mr. Musale Bhoopalsinh S.
Ms. Pawar Priyanka A.
Mr. Wattamwar Shubham R.

Under the Guidance of

Prof. Andrey Sozykin

Ural Federal University, Russia.
2017 – 2018

ACKNOWLEDGMENT

It is our proud privilege and duty to acknowledge the kind of help and guidance received from several people in preparation of this report. It would not have been possible to prepare this report in this form without their valuable help, cooperation and guidance. First and foremost, we wish to record our sincere gratitude to Management of VIMEET and to SUMMER UNIVERSITY (URFU), for their constant support and encouragement in preparation of this report and for making available library and laboratory facilities needed to prepare this report. Our sincere thanks to Prof. Andrey Sozykin, Head of Department of Computer Science, Ural Federal University Russia, for his valuable suggestions and guidance throughout the period of this report.

We express our sincere gratitude to our coordinator Mrs. Ekaterina Lyubimova, Department of International Programs, Ural Federal University Russia, for guiding us in investigations for this seminar and in carrying out experimental work. We hold her in esteem for guidance, encouragement and inspiration received from them. We sincere thanks to coordinators of URFU for their continuous guidance, visit to different structures and sparing valuable time for giving us good industrial exposure. The report on “Implementation of K-Means Clustering Algorithm using Parallel Computation Cluster” was very helpful to us. Their contributions and technical support in preparing this report are greatly acknowledged.

Ms. Bhosale Divya R. Ms. Pawar Priyanka A.

Mr. Musale Bhoopalsinh S. Mr. Wattamwar Shubham R.

ABSTRACT

Nowadays, all most personal computers have multi-core processors. We try to exploit computational power from the multi-core architecture. We need a new design on existing algorithms and software. In this project, we have implemented the parallelization of the well-known k-means clustering algorithm. In this project OpenMp is used which is API for multithreaded applications. Also we have used Putty which is client side application for connecting to the computational cluster. The experimental results demonstrate considerable speedup rate of the proposed parallel k-means clustering method run on multicore / multiprocessor machine, compared to the serial kmeans approach.

Keywords: Clustering Algorithms, High Performance Computing, K-means Algorithm, and Parallel Computing.

INDEX

List of Figures

INTRODUCTION

The term (or) techniques Data Mining has become very popular in recent years. Finding groups of objects such that the objects in a group will be analogous (or related) to one another and different from (or unrelated to) the objects in other groups. Cluster Analysis is very useful without proper analysis implementation of clustering algorithm will not provide good results. Cluster analysis is very useful to understand group related documents like web browsing, group genes and proteins that have similar functionality or group stocks with related price fluctuations and also reduces the size of very large data sets. Traditionally clustering techniques are usually divided in hierarchical and partitioning and density based clustering.

K Means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. Aims to partition n observations into k clusters where each observation belongs to the cluster with the nearest mean. Given a set of observations (x1, x2, …, xn), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k ($\leq$ n) sets S = {S1, S2, …, Sk} so as to minimize the within-cluster sum of squares.

The OpenMP API covers only user directed parallelization, wherein the programmer unambiguously specifies the actions to be taken by the compiler and runtime system in order to execute the program in parallel. OpenMP compliant implementations are not required to ensure for data dependencies, data conflicts, race conditions, or deadlocks, any of which may occur in conforming programs. The OpenMP API uses the fork and join model of parallel execution. Multiple threads of execution perform tasks defined implicitly or explicitly by OpenMP directives. The OpenMP API is intended to support programs that will execute correctly both as parallel programs and as sequential programs.

K MEANS: SERIAL VERSION

The popularity of k-means algorithm is due to its linear computational complexity, O(nkt), where n is the number of data points or objects, k is the number of desired clusters, t the number of iterations the algorithm takes for converging to a stable state. The real time result of serial version of k means clustering algorithm for 10 dimensions 10 million points and 10 clusters was 520.272 secs when implemented on computational cluster.

The computing process needs improvements to efficiently apply the method to applications with huge number of data objects such as genome data analysis and geographical information systems. K means is a method of data analysis. The objective is to partition N data objects into K clusters (K<N) such that the objects in the same cluster are as similar as possible and as dissimilar as possible in different clusters.

The algorithm is composed of the following steps:

1. Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.

2. Assign each object to the group that has the closest centroid.

3. When all objects have been assigned, recalculate the positions of the K centroids.

4. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

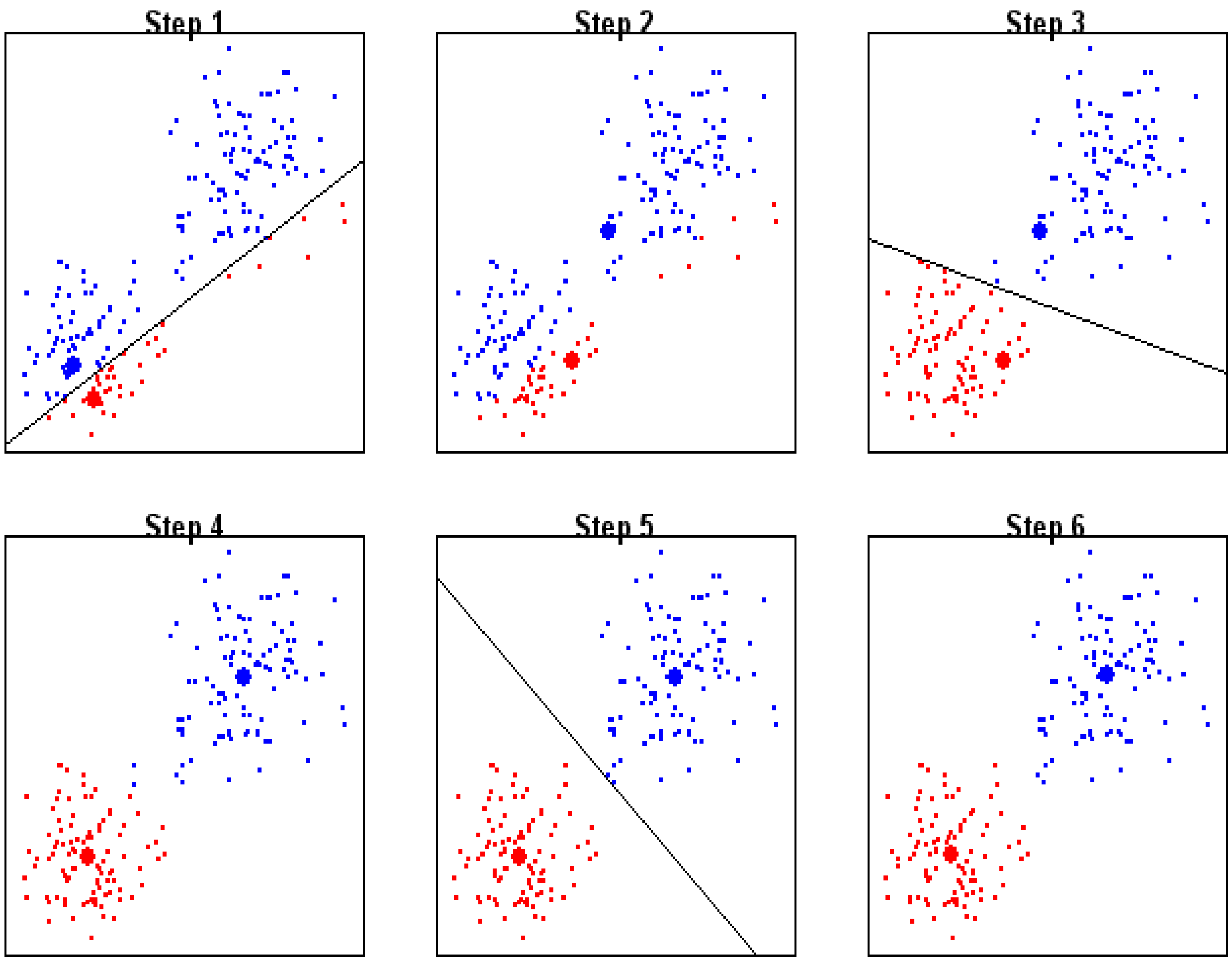


Fig. K-means Algorithm Stepwise Execution

OpenMp

OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most platforms, instruction set architectures and operating systems, including Solaris, AIX, HP-UX, Linux, macOS, and Windows. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior. De-facto standard of parallel programming technology for shared memory systems.

Automatic parallelization

Components of OpenMP:
•Compiler directives

•Functions

•Environment variables.

Parallel programs are hard to develop, test, and debug.

Automatic parallelization:
•Possible only for simple cases.

•Compiler is not always able to find code fragments suitable for parallelization.

•Compiler is often not able to prove that parallelization is safe.

Advices to compiler:
•Developer uses compiler directive to indicate which code fragment should be parallelized.

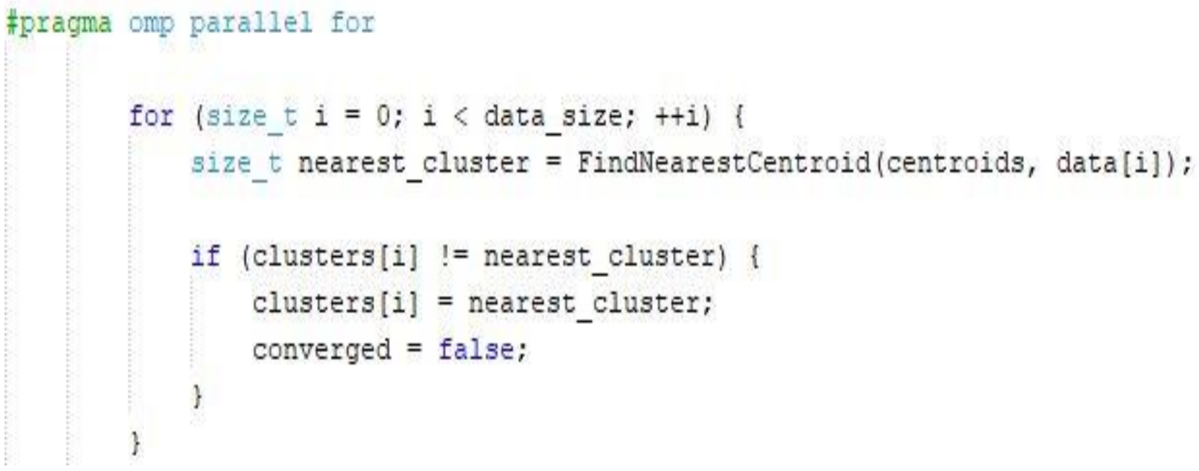
•Compiler parallelizes the specified code fragment.

Pragmas Directive :-

The **‘#pragma’** directive is the method specified by the C standard for providing additional information to the compiler, beyond what is conveyed in the language itself. The forms of this directive (commonly known as pragmas) specified by C standard are prefixed with STDC. A C compiler is free to attach any meaning it likes to other pragmas. All GNU-defined, supported pragmas have been given a GCC prefix.

C99 introduced the _Pragma operator. This feature addresses a major problem with ‘#pragma’: being a directive, it cannot be produced as the result of macro expansion. _Pragma is an operator, much like size of or defined, and can be embedded in a macro.

Its syntax is _Pragma (string-literal), where string-literal can be either a normal or wide-character string literal. It is destringized, by replacing all ‘\\’ with a single ‘\’ and all ‘\"’ with a ‘"’. The result is then processed as if it had appeared as the right hand side of a ‘#pragma’ directive. For example,

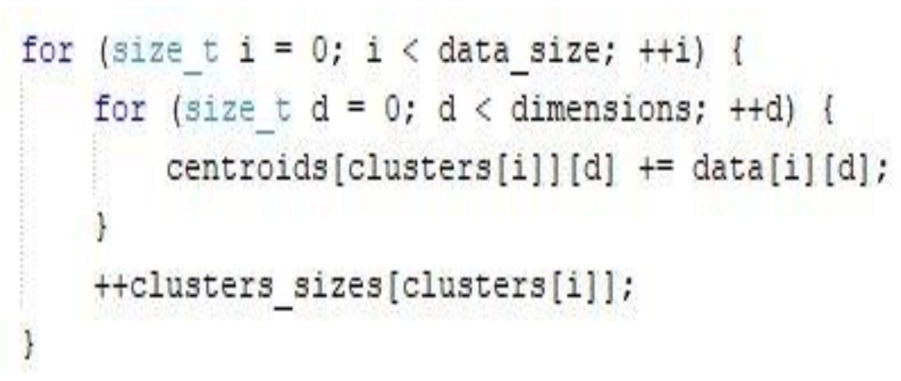


```
#pragma omp parallel for

    for (size_t i = 0; i < data_size; ++i) {
        size_t nearest_cluster = FindNearestCentroid(centroids, data[i]);

        if (clusters[i] != nearest_cluster) {
            clusters[i] = nearest_cluster;
            converged = false;
        }
    }
```

Fig. Loop containing Pragma directive

In computer memory or storage, a race condition may occur if commands to read and write a large amount of data are received at almost the same instant, and the machine attempts to overwrite some or all of the old data while that old data is still being read. The result may be one or more of the following: a computer crash, an "illegal operation," notification and shutdown of the program, errors reading the old data or errors writing the new data. A race condition can also occur if instructions are processed in the incorrect order.

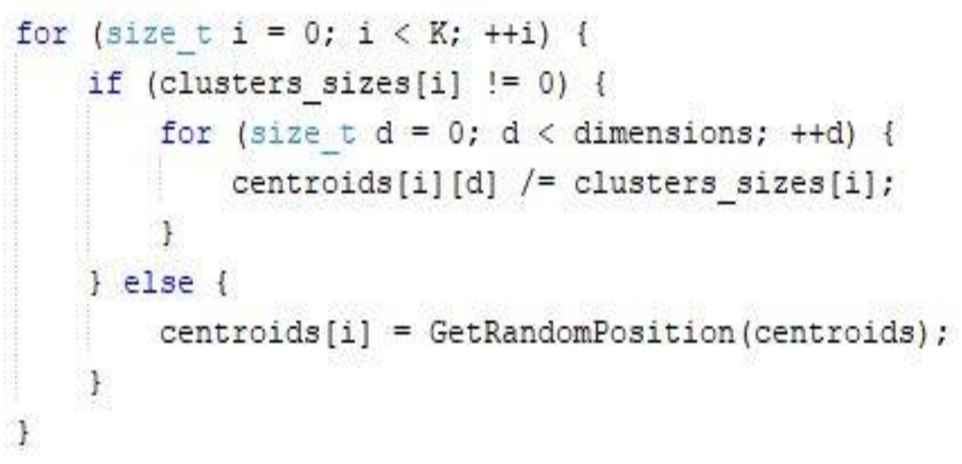
For the below shown fragment of code there is no inclusion of pragma directive as it leads to Data Race Condition. Hence the execution time is infinity.



```
for (size_t i = 0; i < data_size; ++i) {
    for (size_t d = 0; d < dimensions; ++d) {
        centroids[clusters[i]][d] += data[i][d];
    }
    ++clusters_sizes[clusters[i]];
}
```

Fig. Loop Excluding the pragma directive

The below fragment of code doesn't involve pragma as the loop iterates for k times. K is the number of clusters and hence a loop with limited computation.

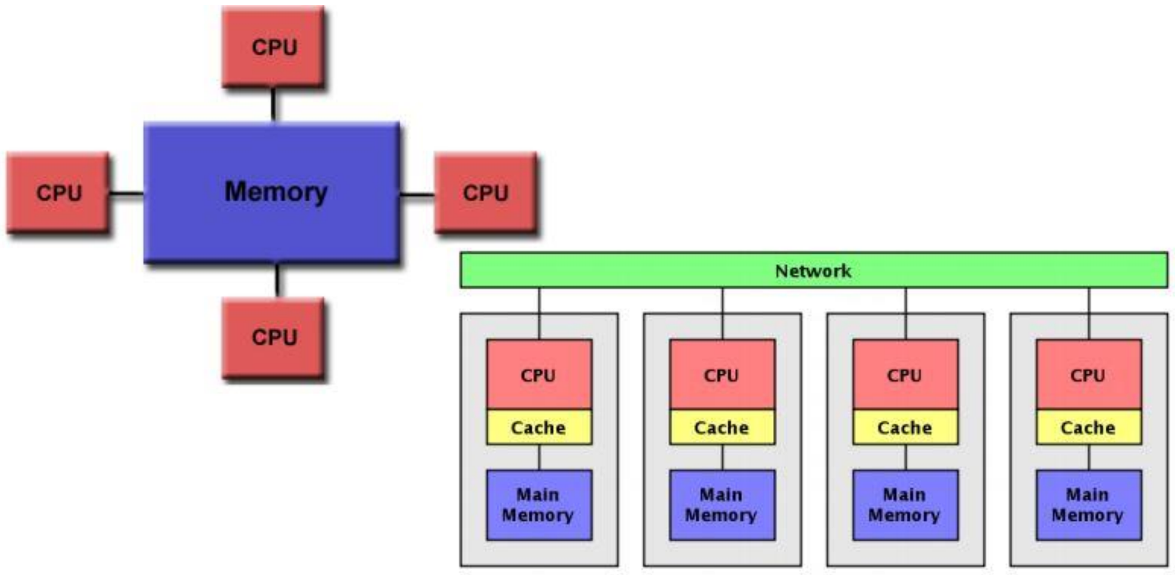


```
for (size_t i = 0; i < K; ++i) {
    if (clusters_sizes[i] != 0) {
        for (size_t d = 0; d < dimensions; ++d) {
            centroids[i][d] /= clusters_sizes[i];
        }
    } else {
        centroids[i] = GetRandomPosition(centroids);
    }
}
```

Fig. Loop excluding pragma directive

PARALLEL COMPUTING

Parallel Computing is the simultaneous use of multiple compute resources to solve a computationally large problem. Parallel computing is a type of computation in which many calculations or the execution of processes are carried out simultaneously. Large problems can often be divided into smaller ones, which can then be solved at the same time. There are several different forms of parallel computing: bit-level, instruction-level, data, and task parallelism. Parallelism has been employed for many years, mainly in high-performance computing, but interest in it has grown lately due to the physical constraints preventing frequency scaling. As power consumption (and consequently heat generation) by computers has become a concern in recent years, parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors.



Parallel computing is closely related to concurrent computing—they are frequently used together, and often conflated, though the two are distinct: it is possible to have parallelism without concurrency (such as bit-level parallelism), and concurrency without parallelism (such as multitasking by time-sharing on a single-core CPU). In parallel computing, a computational task is typically broken down in several, often many, very similar subtasks that can be processed independently and whose results are combined afterwards, upon completion. In contrast, in concurrent computing, the various processes often do not address related tasks; when they do, as is typical in distributed computing, the separate tasks may have a varied nature and often require some inter-process communication during execution.

Parallelism permits all levels of current computing systems, from single CPU machines, to large server farms. We have used putty which is a client side application which allows to access centralised resource cluster from authentication. Cluster consists of 2 CPUs and 12 cores i.e. 6 cores for each CPU. Hence 12 threads can be implemented maximum to avoid process starvation.

RESULTS

Here it is observed that the time required to compute the clusters is decreasing as the number of cores is increased.

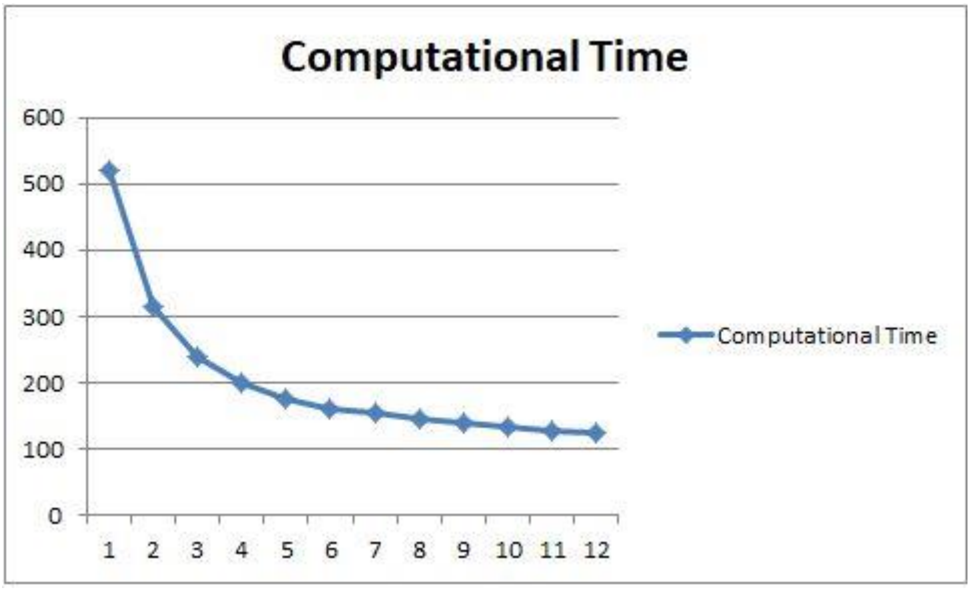


Fig. Graph of Computational Time

The increase in number of cores leads to increase in the speedup of the process. The graph shows that as the number of cores increases the speedup of the program also increases.

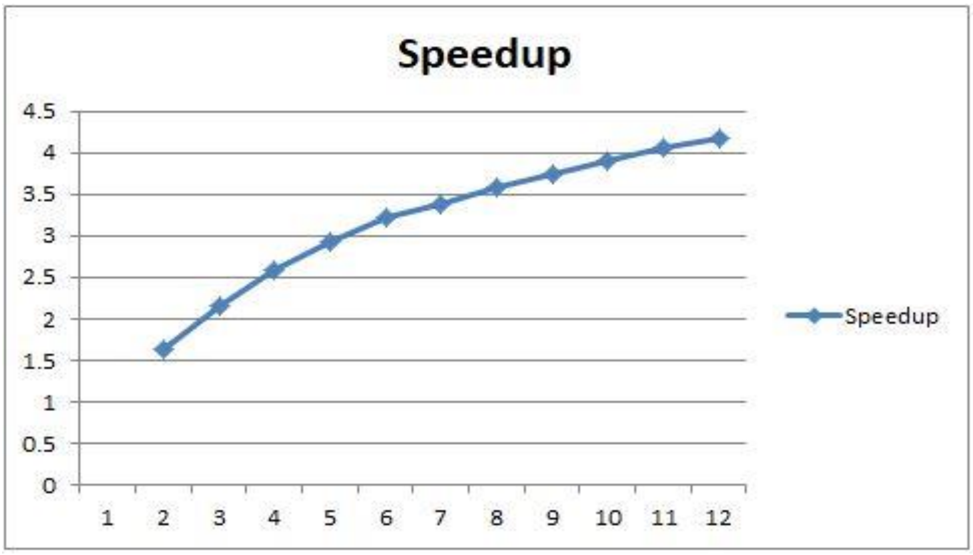


Fig. Graph of Speed Up

CONCLUSION

As compared to the serial version of the k means clustering technique the parallel computing technique gives improved and better results.

OpenMp is used to parallelise the k means clustering technique.

The computational time of the program is decreased as the number of cores is increased.

Speedup of the program increases due to implementation of parallel computational cluster.