Propose Parallelization of K-Medoid Clustering Algorithm

Khaled A. Alenezi¹,a, Mohammad A. Alahmad²,b, Walid Aljoby³,c,
¹Department of Computer Science, International Islamic University of Malaysia, 53100 Jalan Gombak, ²Public Authority for Applied Education and Training, College of Basic Education, Computer Science Department, Kuwait City, Kuwait, ³Sana'a University, Yemen, aalenezi.khaled@yahoo.com, bmalahmads@yahoo.com, cwalid_aljoby@yahoo.com

ABSTRACT

This paper presents an method for paralleling K-Medoid clustering algorithm. The K-Medoid algorithm will be alienated into responsibilities, which will be charted into multiprocessor system. The control configuration for the way of stating the responsibilities in parallel form and the communication model that pleased the tool for contact between these responsibilities is existing. Data parallel model is constructed by rotting the responsibilities between the processors. The operation and trying of the parallel model have showed using SESE academic simulator under Fedora 11 version at Linux OS environment.

Keyword: Data parallel model, K-Medoid, Parallel model, SIMD.

1. Introduction

The central keys in innovation of computer architecture diversity are processing units, data paths, and memory units. Furthermore, the growing in clock rate leads into design of the supercomputer machines. This design presents many procedures of parallism that are not observable to programmer such as pipelining, dynamic scheduling, speculative execution and non-blocking branches (George et al., 2003). The obtainability of parallel environments motivates the programmers to increase the performance of the complex algorithms at any part of computer system. Data mining and machine learning algorithms are complex algorithms that require parallel processing to enhance the speed of execution. Most public results of data mining are data clustering and data classification. Clustering is the process of grouping the data into classes or clusters, so that objects within the cluster have high similarity in comparison to one another but are very dissimilar to the objects in the other clusters (Han et al., 2006). This paper is organized as follows. In Section 2, the K-medoids algorithm is reviewed. In Section 3, k-medoids algorithm analysis is presented, followed by introducing the proposed parallel model in Section 4. In Section 5, the performance of the proposed model is evaluated and results are discussed. Section 6 concludes the paper.

2. K-Medoids Algorithm Overview

K-Medoids is a traditional partitioning technique of clustering that groups the dataset (D) of N objects into K clusters based on similarity. It is more robust and better than K-means in detecting the noise and outliers. The object is considered as Medoid of a cluster, where the similarity between object and all objects in the same cluster is high and low with the objects in the other clusters. In the K-Medoids method, we can choose any actual objects to represent the clusters, each one as representative object for each cluster. The remaining dataset (objects) are clustered with the representative object based on the similarity between them. The initial representative objects (Medoids) are chosen randomly. The iterative process continues as long as the quality of the resulting clustering is improved. This quality of clustering is estimated using a cost function that measures similarity between an object and the representative object. There are four cases to decide
whether a non-representative object, $O_{random}$, is a good replacement for a current representative object, $o_j$, as following:

**Case 1:** $p$ currently belongs to representative object, $o_j$. If $o_j$ is replaced by $O_{random}$ as a representative object and $p$ is closest to one of the other representative objects, $o_i$ ($I \neq J$), then $p$ is reassigned to $o_i$.

**Case 2:** $p$ currently belongs to representative object, $o_j$. If $o_j$ is replaced by $O_{random}$ as a representative object and $p$ is closest to $O_{random}$, then $p$ is reassigned to $O_{random}$.

**Case 3:** $p$ currently belongs to representative object, $o_i$, $I \neq j$. If $o_j$ is replaced by $O_{random}$ as a representative object and $p$ is still closest to $o_i$, then the assignment does not change.

**Case 4:** $p$ currently belongs to representative object, $o_i$, $I \neq j$. If $o_j$ is replaced by $O_{random}$ as a representative object and $p$ is closest to $O_{random}$, then $p$ is reassigned to $O_{random}$.

Table 1: The K-Medoids algorithm partitioning based on Medoids

<table>
<thead>
<tr>
<th>Input: k, the number of clusters; Dataset (D) containing n objects.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output:</strong> A set of k clusters. Method: 1- arbitrarily choose N objects in Dataset as the initial representative objects 2- repeat 3- assign each remaining object to the cluster with the nearest representative object; 4- randomly select a non-representative object, $O_{random}$; 5- compute the total cost, $S$, of swapping representative object, $o_i$, with $O_{random}$; 6- if $S&lt;0$ then swap $o_i$ with $O_{random}$ to form the new set of k representative objects; 7- until no change;</td>
</tr>
</tbody>
</table>

Table 1 shows the steps of the K-Medoids clustering algorithm.

Table 2: K-Medoids algorithm symbols and definitions

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Dataset to be clustered</td>
</tr>
<tr>
<td>N</td>
<td>Number of objects in D</td>
</tr>
<tr>
<td>$o_i$</td>
<td>Object i in D</td>
</tr>
<tr>
<td>K</td>
<td>Number of clusters</td>
</tr>
<tr>
<td>$O_{random}$</td>
<td>Random object</td>
</tr>
<tr>
<td>S</td>
<td>Cost of its dissimilarity</td>
</tr>
<tr>
<td>P</td>
<td>Number of Processors</td>
</tr>
<tr>
<td>$P_{id}$</td>
<td>Processor ID</td>
</tr>
<tr>
<td>PKM</td>
<td>Paralleling K-Medoids</td>
</tr>
</tbody>
</table>
Table. 2 shows the symbols and definitions related to the serial and parallel K-Medoid clustering algorithm.

3. **K-Medoids Algorithm Analysis**

The algorithm requires dataset (D) and clusters (K) as input. After choosing \( n \) objects as representative objects (medoids), the algorithm repeatedly tries to make better selection for objects around it. The algorithm tries to combine a pair of each object with medoid objects, and then calculates the cost for each combination and cluster the objects according the cost quality.

From the real implementation the time complexity for the algorithm depends on the dataset (# of objects and # of attributes) and number of clusters. Whenever the number of clusters (\( K \)), attributes (\( f \)), and number of objects are duplicated, the time complexity for generating the K-clusters output is also duplicated.

To gain high speed up of paralleling K-medoids algorithm, it should be necessarily to analyse the control structure of the algorithm, the degree of concurrency, interaction graph between tasks, and the mapping of the tasks among the processors.

4. **Parallelizing model**

The steps for designing and implementation of PKM can be classified into four parts: identifying the portion of the work that can be performed concurrently, mapping the concurrent pieces of work into multiple processors running in parallel, managing access to data shared by multiple processors, and the synchronizing the processors.

For identifying the portion of the work that can be performed concurrently, the K-medoids algorithm can be divided into seven tasks as shown in Table. 3. The sequential flow for execution tasks is shown in Fig. 2.

![Fig. 2: Sequential flow for execution serial K-medoid algorithm](image)

These tasks are generated statically, because they are known before the algorithm starts execution. The tasks from two to six can be parallelized and performed together.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Iteration determinant</td>
</tr>
<tr>
<td>T2</td>
<td>Distance calculation</td>
</tr>
<tr>
<td>T3</td>
<td>Objects clustering</td>
</tr>
<tr>
<td>T4</td>
<td>Cost calculation</td>
</tr>
<tr>
<td>T5</td>
<td>Cost comparison</td>
</tr>
</tbody>
</table>
The degree of granularity depends on the number of processor and dataset size. The average degree of concurrency for the main tasks, two, three, five, and six are represented by disconnected and empty edge-set task dependency graph, because each subtask inside the multitask processed independently. Fig. 3 shows the task dependency graph for these tasks.

![Task dependency graph for main tasks 2, 3, 5, and 6.](image)

From the above figure the average degree of concurrency is $P/1=p$, which represents full concurrency execution. Also, the speed up approximately equal to the number of processors which given by the following formula:

$$\frac{p + \left(\frac{N}{p}\right) \times p}{\left(\frac{N}{p}\right) + p}$$

(2)

The first and last tasks perform very small working and cannot be parallelized at all.

The PKM consists of several routines. For parallelization these routines, the tasks are distributed uniformly between processors. From the data and control flow between PKM routines, and also from the task dependency graph between tasks inside each routine, it is noted that there are independent tasks that can be performed together. Those tasks mapped independently for each thread, but some tasks need data that is produced by other tasks. These tasks require waiting for other to finish its execution. Using the data parallel model, the mapping of tasks for each processor is assigned in sequence. Namely, the first processor processes the first record in the data set, then jump to process record with index $(0+\text{pid})$, the second processor process the second record in the data set, then jump to process record with index $(1+\text{pid})$ and so forth for other processors. For example, to calculate the distance between 8 objects and $O_{\text{rep}}$ object over 3-processor, processor one will calculate the distance for object one, then jump to calculate the distance with object 4. Processor two will calculate the distance for object two, and then jump to calculate the distance for object 5.

This means that each processor will jump by the number of processors with accumulative pattern until reach the last object in the dataset. Fig. 4 shows the mapping of PKM main tasks over $p$-processors and its control flow.
PKM is implemented by the programming under shared address space platform. The shared data requires locking until finish its execution. Also, we require a barrier for threads early finish its work to wait for other threads to start execution the next tasks equally. The synchronization is represented as in Fig. 4 by inserting a barrier between two consecutive tasks.

### 5. Evaluation and Results Discussion

The parallelization of K-Medoids algorithm has evaluated using SESC academic simulator. SESC is an event-driven superscalar simulator that simulates microprocessor architecture such as single processors, chip multiprocessors and processors-in-memory. SESC can model a full out-of-order pipeline with branch prediction, caches, buses, and every other component of a modern processor necessary for accurate simulation [9].

In this section, we will discusses the effects of concurrency, attributes (object features), clusters, and number of object on system performance. In the first part, we discuss the effects on system performance when clustering the same number of objects with different number of attributes. In the second part, we discuss the effects on system performance when increasing the number of objects for the same number of attributes. Finally, we discuss the effects on system performance and the degree of concurrency when running dataset on the N-processors (p).

**First Part:** This part discusses the effect of clustering a dataset with variation in the number of attributes on system performance. Clustering datasets with size (256- objects x 4- attributes) and (256- objects X 8- attributes) into two, four, and eight clusters are used as sample dataset.
Fig. 5: Clustering dataset with size (256 objects X 4 Features) over multiple processors.

Fig. 5 depicts that speed increases when grouping the dataset into two and four clusters over two to eight processors, but starts to decrease when clustering the same data set over sixteen processors. On the other hand, the speed of clustering the same dataset into eight clusters is improved significantly with increasing the number of processors as in Fig. 6. It can be concluded that whenever the features of objects or clusters increase, the performance improvement with increasing number of processors increases. Also, we conclude that PKM is very robust and scalable whenever the number of attributes or clusters increases.

Fig. 6: Clustering dataset with size (256 objects X 8 Features) over multiple processors.

**Part Two:** This part discusses the effect of clustering a dataset with variation in the number of objects on system performance. Clustering dataset with size (512–objects x 8-attributes) into eight clusters with different number of processor is used as sample dataset. Fig. 7 shows that number of processors is duplicated for which the running time consequently decreases approximately by half.

We also compare the system performance when clustering datasets with (256 objects – 512 objects – 1024 objects) with 8-attributes into 8-clusters over 4-processors.

It can be concluded that PKM works well when decompose the dataset into small number of large tasks. Namely, when the subsets of dataset have a large number of attributes mapped into processor for clustering into a large number of clusters. This means that PKM works well when the degree of granularity is coarse-grained.
6. Conclusion

The concluding form this paper that presentation of PKM was better-quality when the subsections of dataset have a huge amount of features that are charted over multiprocessors for grouping into a big number of groups. Also, the results presented that rapidity nearly is folded when repetition the number of processors for clustering any examined dataset.

7. References:


Han J, Kamber M. Data mining: concepts and techniques: Morgan Kaufmann; 2006.


