Parallel Pruning for K-Means Clustering on Shared Memory Architectures

Attila Gürsoy and İlker Cengiz

Computer Engineering Department, Bilkent University
Ankara, Turkey
{agursoy,icengiz}@cs.bilkent.edu.tr

Abstract. We have developed and evaluated two parallelization schemes for a tree-based k-means clustering method on shared memory machines. One scheme is to partition the pattern space across processors. We have determined that spatial decomposition of patterns outperforms random decomposition even though random decomposition has almost no load imbalance problem. The other scheme is the parallel traverse of the search tree. This approach solves the load imbalance problem and performs slightly better than the spatial decomposition, but the efficiency is reduced due to thread synchronizations. In both cases, parallel tree-based k-means clustering is significantly faster than the direct parallel k-means.

1 Introduction

Clustering is an important area which finds application in a variety of fields including data mining, pattern recognition, explorative data analysis, image processing, and more [1] [2]. K-means [3] is a partitional clustering method and it is one of the most commonly used clustering algorithms. In this paper, we focus on parallelization techniques for a faster version of k-means clustering algorithm, a tree-based k-means method [4].

The k-means (direct) algorithm treats input patterns as points in a d dimensional space and employs Euclidean-distance based similarity metric between patterns and cluster centers. The algorithm chooses an initial set of cluster centers and then each pattern is assigned to the cluster represented by the closest cluster center. After all patterns processed and new clusters are formed, cluster centers are updated to represent new clusters. The sequential execution time of k-means can be improved by reducing the number of distance calculations. The algorithm presented in [4] is one of the such approaches. The algorithm organizes patterns in a k-d tree. The root of the tree represents all patterns and children nodes represent patterns in subspaces. In each iteration, the k-d tree is traversed in a depth-first manner starting at the root node. At the root level, all cluster centroids are candidates to be the closest centroid to any pattern in the space represented by the root node. As we traverse the tree, a pruning method is applied to eliminate some of the candidates for the subspace represented by each node visited. That is, the candidate set that comes from the parent node might contain some clusters centroids that cannot be closest to any pattern in the subspace. When the candidate set is reduced to one cluster centroid, all the patterns in the subspace is assigned to that cluster. Otherwise, a leaf node eventually is reached and pairwise distance calculations are performed for all patterns in the leaf node and cluster centroids in the candidate set.

Parallelization of the direct k-means method is relatively easier. However, in the case of tree-based k-means, the traverse of the irregular tree structure complicates parallelization and poses load balancing and synchronization problems. In this paper, we discuss parallelization of the tree-based k-means method and propose two different schemes based on pattern decomposition and parallel search of the k-d tree used in the tree-based algorithm. The main motivation behind this study is to develop and evaluate alternative parallelization schemes. The implementation uses Pthreads which is a standard thread interface available on most multiprocessor systems Although this study is done for shared memory machines, the proposed pattern decomposition scheme can be used for distributed memory machines as well.

2 Parallel Pruning for Tree Based K-Means

The parallelization of the tree-based k-means is more challenging due to the irregular tree decomposition of space (since it depends on pattern distribution), and varying computations during the traversal. The computations done during the traversal can be coarsely divided into two groups: internal-node computations and leaf computations. In the internal nodes, the space covered by a node is compared against the current candidate set of centroids. Since some of the cluster centroids might have been pruned in the upper levels, the number of distance calculations (which is proportional to the size of the candidate set) can vary across internal nodes. At the leaves, similarly, distance calculations among a differing number of remaining patterns and number of candidates results in varying computation loads. With this load imbalancing in mind, a way of distributing computations done at the nodes need to be developed. One approach is to partition patterns among threads such that each thread is responsible for the pruning due to the space covered by its own patterns. A second one would be the parallel traversal of a single tree in a dynamic fashion.

Pattern Decomposition – parallel multiple tree pruning. Pattern decomposition scheme divides patterns into p disjoint partitions where p is the number of processors. Then, each partition is assigned to a different thread. Every thread forms a k-d tree for its own patterns and performs tree based algorithm on its own set of patterns. This way of parallelization is similar to the parallelization of the direct k-means scheme. However, the tree version has two major differences or problems: load imbalance and possibly less pruning compared to the sequential case. For example, consider two threads, one with a set of patterns that are concentrated in a small space, and another one with the same number of patterns but scattered around much larger space. There will be more pruning of candidate cluster centroids at the upper levels of the local tree (that belongs to a thread)) in the case of compact subspace because many cluster centroids possibly will be far from the compact space. In the case of sparse and larger subspace, the pruning might shift towards to the leaves which might result in more distance calculations. In the direct k-means, it does not matter which patterns form a partition. However, in the tree-based case, we have to choose patterns such that the problems mentioned above are addressed. We have tried two different partitioning schemes: (a) **random pattern decomposition** where each thread gets equal number of patterns chosen randomly from the space covered by all patterns, (b) **spatial decomposition** where each thread gets equal of number of patterns that belong to a compact space.

Parallel single tree pruning. The possible load imbalance problem of the static pattern decomposition can be solved by distributing distance calculations (not the patterns) to threads dynamically. In this scheme, we have a single tree representing all the patterns. The tree traverse for pruning is parallelized by maintaining a shared pool of work. A work is a tuple composed of a node (where the traverse to be started) and a set of candidate clusters. An idle thread simply picks a work from the queue and applies the pruning procedure starting from the node possibly going downwards in a depth-first manner. In order to create enough work for other threads, a thread puts more work (some of its sub-trees that need to traversed) into the shared queue if there is no work in the queue for other threads. The advantage of this scheme is dynamic load balancing . An idle thread will always find work to do as long as some parts of the tree still have not been traversed. However, this scheme needs to maintain a shared work queue which requires usage of locks for enqueue and dequeue operations. The frequency of such operations attempted by the threads has significant impact on the performance. Blocking kernel-level threads on a lock results in a switch from user space to kernel space in the operating system and is expensive.

3 Evaluation of the Multiple Tree and Single Tree Pruning

We have conducted some preliminary performance experiments on a Sun system where 8 processors are available. We used two different data sets with 2-dimensional patterns: dataset1 with 100000 patterns and dataset2 with one million patterns. Dataset1 is the same as in used in [4] which contains randomly generated globular 100 clusters. For spatial decomposition, we partitioned the patterns into strips containing equal number of patterns. The execution time and load balancing data are shown in Table 1 for varying number of threads. The spatial decomposition has better execution times than the random pattern decomposition. The reason for this is explained by the amount of pruning done and the computational load imbalance. We collected statistics about the distance calculations done at the leaves of the tree. The more distance calculation means the less pruning done at the interior nodes. For spatial decomposition, the number of distance calculations increases slightly as the number of threads increases. That is, the amount of work done in the parallel version increases with the number of threads, but slightly, which results in reduced efficiency. This effect is more significant in the random decomposition case. The number of distance calculations increases more than 3 times for 8 threads and this causes the parallel execution time of the random decomposition to be significantly worse than the spatial decomposition. On the other hand, random decomposition has better balanced computational load. The spatial decomposition has upto 20% load imbalance whereas random decomposition has no more than 6% load imbalance. As a result, the one dimensional spatial decomposition scheme suffers from load imbalance, random decomposition suffers from increased work due to less pruning, but overall, spatial decomposition is superior to random decomposition.

Table 1. Load balance and execution time results for pattern decomposition

Num. of Threads	Time (seconds)		Num. of distance calc. (x10^6)		Load imbalance	
	spatial	random	spatial	random	spatial	random
1	18.03	21.34	14.93	14.93	0	0
2	11.92	14.43	14.89	22.64	11	2
3	9.10	14.36	15.73	28.5	12	2
4	7.91	11.52	16.86	34.62	13	6
5	7.15	13.37	17.32	39.02	18	1
6	6.12	16.66	17.40	44.80	20	6
7	6.08	14.90	18.16	50.52	18	2
8	6.40	15.85	19.84	54.25	16	4

Table 2 compares performance of the single tree pruning, spatial-decomposition, and direct k-means (all parallel) for dataset1 and dataset2. First, the execution time of the tree-based parallel k-means is significantly superior than the direct k-means. When we compare single tree pruning and spatial decomposition techniques, we observed that single tree pruning performs better because it does not have a load imbalance problem. However, the efficiency is not as good as the one of the direct k-means. One of the reasons is the locks to access to the shared queue. Although we tried to reduce the use of locks, performance problems after five processors are noticeable. Another reason might be due to cache when the tree is traversed in a dynamic and irregular way. This is particularly noticeable in the execution time of spatial and random decompositions for one thread case (Table 1). Both spatial and random decompositions (one thread) perform exactly the same sequence of computations. However, the random one is 20% slower (which was tested on several different Sun machines). Since the number of threads is one, this difference is not due to parallelism. In the spatial decomposition, the tree is build such that the sequence of memory locations allocated for nodes mimics the order of nodes visited during the depth-first search which (most probably) results in better use of cpu caches. However, in the random case, that order is not valid. A similar case is true for the single tree version (visiting nodes in an arbitrary order). However, both spatial and single-tree cases are significantly superior than the parallel direct k-means.

Table 2. Execution time for single tree, spatial decomp., and direct

Num. of Threads	Dataset 1 (100000 patterns)			Dataset 2 (1000000 patterns)		
	single-tree	spatial	direct	single-tree	spatial	direct
1	17.99	18.03	145.6	167.55	167.03	1987.91
2	11.48	11.92	73.31	98.98	107.85	996.91
3	8.60	9.10	52.54	71.90	80.97	693.69
4	6.51	7.91	39.36	57.29	67.02	524.83
5	5.73	7.15	31.78	48.62	54.41	419.18
6	5.73	6.12	26.58	42.49	50.61	351.92
7	5.60	6.08	22.58	41.31	49.08	302.97
8	5.87	6.40	20.05	39.66	51.81	269.93

4 Conclusion and Future Work

In this work, we developed and evaluated two parallelization schemes for a tree-based k-means clustering on shared memory architectures using Pthreads. One of the schemes is based on pattern decomposition (multiple trees). We determined that spatial decomposition of patterns outperforms random pattern decomposition even though random decomposition has almost no load imbalance problem. The spatial decomposition, on the other hand, can be improved further by forming partitions in a more clever way and can be used also for running the algorithm on distributed memory machines. The other approach is the parallel traverse of the single tree. This approach solves the load imbalance problem, but the overhead of thread synchronizations need to be handled, for example, by employing programming environments with more efficient thread support.

References

1. Jain, A. K., Murty, M. N., Flynn, P. J.: Data Clustering: A Review. ACM Computing Surveys, Vol. 31, No. 3, (1999) 264-323 321
2. Judd, D., McKinley, P. K., Jain, A. K.: Large-Scale Parallel Data Clustering. In Proc. of the 13th Int. Conf. on Pattern Recognition, (1996) 321
3. McQueen, J.: Some Methods for Classification and Analysis of Multivariate Observations. Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, (1997) 173–188 321
4. Alsabti, K., Ranka, S., Singh, V.: An Efficient K-Means Clustering Algorithm. IPPS/SPDP 1st Workshop on High Performance Data Mining, (1998) 321, 323