Design and Evaluation of a Parallel Execution Framework for the CLEVER Clustering Algorithm

Chung Sheng CHEN [a,1], Nauful SHAIKH [a], Panitee CHAROENRATTANARUK [a], Christoph F. EICK [a], Nouhad RIZK [a] and Edgar GABRIEL [a]

[a] *Department of Computer Science, University of Houston, Houston, TX 77204-3010, USA*

Abstract. Data mining is used to extract valuable knowledge from vast pools of data. Due to the computational complexity of the algorithms applied and the problems of handling large data sets themselves, data mining applications often require days to perform their analysis when dealing with large data sets. This paper presents the design and evaluation of a parallel computation framework for CLEVER, a prototype-based clustering algorithm which has been successfully used for a wide range of application scenarios. The algorithm supports plug-in fitness functions and employs randomized hill climbing to maximize a given fitness function. We explore various parallelization strategies using OpenMP and CUDA, and evaluate the performance of the parallel algorithms for three different data sets. Our results indicate a nearly linear scalability of the parallel algorithm using multi-core processors, reducing the execution time and allowing to solve problems which were considered not feasible with the sequential version of CLEVER.

Keywords. clustering algorithm, data mining, GPU Computing, OpenMP

Introduction

The last decade, there has been an enormous increase in the data available for both business communities as well as in science. Mining these vast pools of data to extract valuable knowledge is however challenging, both due to the computational complexity of employed algorithms as well as due to the problems handling large data sets itself. The usage of parallel compute resources, such as given by multi-core processors, general purpose graphics processing units (GPGPUs) or clustered resources offers the potential to reduce the time required to run these algorithms and therefore the time to solution. Exploiting parallelism in clustering and classification algorithms however does require significant changes in existing codes and sometimes a re-design of the algorithm itself.

In the following, we would like to highlight the occurring computational problems using a clustering algorithm called CLEVER (CLustEring using representatiVEs and Randomized hill climbing) [4]. CLEVER is a prototype-based clustering algorithm which seeks for clusters maximizing a plug-in fitness function. It forms clusters by as-

[1] Corresponding Author

signing the objects in the dataset to the closest representative. CLEVER uses randomized hill climbing to seek for a good clustering, i.e. it samples p solution in the neighborhood of the current solution, and continues this process as long as better solutions are found.

CLEVER is more powerful than traditional prototype-based clustering algorithms in that it can be applied to a much broader class of problems. CLEVER has been successfully used for collocation data mining [4], spatial regression [1], clustering polygons [13], and mining related datasets [14].

However, the search process of CLEVER is significantly more complex than some other algorithms, such as K-means [10]. In particular, CLEVER needs to compute the fitness of a large number of clusterings and the current clustering has to be compared with p clusterings in the neighborhood of the current clustering. When forming clusters a large number of 1-nearest neighbor queries have to be evaluated to determine the closest representative for an object in the dataset and a large number of clusters has to be created. Furthermore, similarly to K-means, K-medoids [7], and EM [3], CLEVER is sensitive to initialization. Consequently, it is desirable to run CLEVER multiple times using different initial clusterings, and to return the best results found in the different runs.

Past research parallelized the following types of clustering algorithms using the following parallelization strategies: Olman etc. [11] parallelized a graph-based clustering algorithm and Foti [5] parallelized a model-based clustering algorithms using MPI. [6] uses OpenMP and MPI to speed up K-means. [8] and [9] use multi-cores to speed up K-means and K-medoids and Wu [15] parallelizes K-means on a GPU.

The topic of the paper is the design and evaluation of parallel computation frameworks to speed up CLEVER alleviating the performance challenges outlined in the above. The main contributions of the paper are the development of an OpenMP and CUDA version of CLEVER and its evaluation. The evaluation is based on a unique and challenging clustering benchmark consisting of 3 datasets and their associated fitness function.

1. Parallel CLEVER

Prototype-based clustering algorithms construct clusters by seeking an 'optimal' set of representatives—one for each cluster; clusters are then created by assigning objects in the dataset to the closest cluster representative. The representative set is a subset of the objects in the dataset. Popular prototype-based clustering algorithms are K-Medoids/PAM [7] and K-means [10]. CLEVER seeks to maximize a plug-in fitness function q—the quality of a clustering is the sum of rewards individual clusters receive.

The algorithm (see Fig. 1 which gives the pseudo-code of the algorithm) starts with randomly selecting k' representatives from the dataset O—k' is a parameter of the algorithm. Next, CLEVER samples p solutions in the neighborhood of the current solution, and picks the solution s with the maximum value for $q(s)$ as the new current solution, if there is some improvement in fitness. Neighboring solutions of the current solution are created using three operators: 'Insert' - inserts a new representative into the current solution, 'Delete' - deletes a representative from the current solution and 'Replace' - replaces a representative with a non-representative. Each operator has a certain selection probability and representatives to be manipulated are chosen at random. The algorithm also allows for larger neighborhood sizes, in this case, solutions that are sampled are

generated by applying more than one randomly selected operators to the representative set of current solution. Moreover, to battle premature convergence, CLEVER re-samples $p' = p * q > p$ solutions before terminating.

Inputs:

Dataset O, k', neighborhood-size, p, q, β , object-distance-function d or distance matrix D, $imax$

Outputs:

Clustering X, fitness $q(X)$, rewards for clusters in X

Algorithm:

1. Create a current solution by randomly selecting k' representatives from O.
2. If $imax$ iterations have been done terminate with the current solution
3. Create p neighbors of the current solution randomly using the given neighborhood definition.
4. If the best neighbor improves the fitness q, it becomes the current solution. Go back to step 2.
5. If the fitness does not improve, the neighborhood of the current solution is re-sampled by generating $p * q$ more neighbors. If re-sampling does not lead to a better solution, terminate; otherwise, go back to step 2 replacing the current solution by the best solution found by re-sampling.

Figure 1. Pseudo-code of CLEVER

1.1. OpenMP parallelization

The primary goal in parallelizing CLEVER is to improve the execution time of a given, large problems on a particular platform. Due to the advent of multi-core processors, virtually all desktop computers today have two or more computational cores available. Thus, the first parallel version of CLEVER, called *PAR-CLEVER*, is targeting these multi-core processors. The parallelization is based on OpenMP [12] directives.

In order to identify the most time consuming portions of the code, a detailed analysis of the code has been performed using a profiling tool. The analysis revealed, that the creation of clusters along with the calculation of distances between objects in the dataset, generating solutions in the neighborhood of the current solution and fitness computations are the most time-consuming parts of the code. Since the according code section are organized in large loops, they map very well to the OpenMP parallel loop construct.

Moreover, the steps 3 and 4 in the pseudo-code find the best of p randomly selected neighboring solutions of the current solution. A loop which forms the clusters for each neighboring solution and computes its fitness can be parallelized since the computations inside this loop are totally independent of each other. Thus a task parallelism approach can be applied to parallize steps 3 and 4 of CLEVER.

One common bottleneck faced by prototype-based clustering methods is during cluster formation when objects have to be assigned to the nearest cluster representative. The current implementation computes object cluster assignments on the fly. One alternative solution is to use incremental updating for cluster formation. For example, if a cluster/cluster representative is deleted, only the objects that belong to the deleted clusters have to be assigned to other clusters, but the cluster assignments of all other objects re-

main the same. On the other hand, if a cluster representation is inserted, only the distance between an object and its "old" cluster representative has to be compared with its distance to the newly created representative–if the latter distance is smaller the object has to reassigned to the new cluster representative. Our experimental results show that if 20% of the representative set is changed, a speed up of 2 can be accomplished with incremental updating in the cluster formation step.

At this time, there are three different OpenMP versions of PAR-CLEVER:

1. Loop-level parallelism: this versions relies on parallelizing the for loops which find regions for a set of representatives, distance computation, neighboring solutions generation and fitness computation.
2. Loop-level parallelism and incremental updating: this version additionally uses the incremental updating approach to form the clusters described above.
3. Task-level parallelism: this version assigns different neighboring solutions to different threads and creates clusters and computes their reward in parallel.

Note, that none of the three version contains nested OpenMP parallelization.

1.2. CUDA parallelization

General Purpose Graphics Processing Units (GPGPUs) offer end-users the ability to utilize graphics hardware for general purpose computing. The most popular programming paradigm to use these graphics units is currently based on the CUDA [2] programming language, an extension to the C language developed by NVIDIA. Furthermore, compute cores on a GPGPU can only access data items that have been transferred into the main memory of the GPU.

Currently, the CUDA version of CLEVER is focusing on two computational intensive tasks: assigning nearest representatives, and calculating the fitness of clusterings. The main challenge when assigning objects to clusters is a bank conflict when all threads try to access the representative information. To resolve this, data items are loaded into the shared memory segments of each Streaming Processor of the GPGPU first. Threads are synchronized before they execute the actual computation, namely determining which representative is closest to an object in the dataset.

1.3. Parallel Hill Climbing

In the previous sections, we discussed several approaches to speed up a single run of a clustering algorithm which employs hill climbing. As mentioned earlier hill climbing approaches are attractive—and frequently the last (and only) "resort"—to solve challenging NP-hard problems, but they are very sensitive to the initialization. Consequently, if hill climbing approaches are used in practice usually *hill climbing with restart* is used: running hill climbing multiple times for different initial solutions, and the best result that was found in the different runs is returned as a final solution.

There are two options to implement randomized hill climbing with restart for PAR-CLEVER: Option 1: CLEVER is run independently r times with different initial seeds Option 2: The different runs of CLEVER exchange information that is used as feedback to search more intelligently.

Option 1 can be trivially parallelized as the different runs are completely independent. As far as Option 2 is concerned, we envision a version of CLEVER which initially

Table 1. 100vals data set characteristics

data size	3,359 points
attributes	<x , y, class label>
Distance Function	Euclidean Distance
Fitness Function	$i(r) = Max(0, (P(Majorityclass(r)) - threshold))^{\nu}$ $Fitness = \sum i(r) \times \|r\|^{\beta}$ where $P(Majorityclass(r))$ is the percentage of example in r belonging to the majority class in the region(cluster) r and $\|r\|$ is the number of objects in region r

conducts a large number of searches for just a few iterations, and then based on feedback from the fitness function, continues the "more promising" searches, and aborts the searches which accomplished only low fitness. This leads to the interesting question if the exchange of feedback in Option 2 leads to better solution quality, if compared with Option 1. If the paper is accepted, we plan to discuss an implementation of Option 2 version of CLEVER and we will also conduct a small experiment which evaluates, if such an approach leads to better solution quality.

2. Evaluation

In the following, we describe the benchmarks used to evaluate the parallel versions and give a subset of the results obtained on a shared memory architecture.

2.1. The Benchmark Datasets

10Ovals. This is an artificial dataset which contains objects that belong to 10 ovals which have a class label associated with it. The task of this test case is to find pure clusters, i.e. all or a large number of objects in a cluster belong to the same class. The main purpose of this dataset it to validate the correctness of different versions of CLEVER; As the "ground truth" is known, good solutions can be easily distinguished from bad solutions. Table 1 gives the main characteristics of this data set, and the left part of Fig. 2 visualizes the dataset.

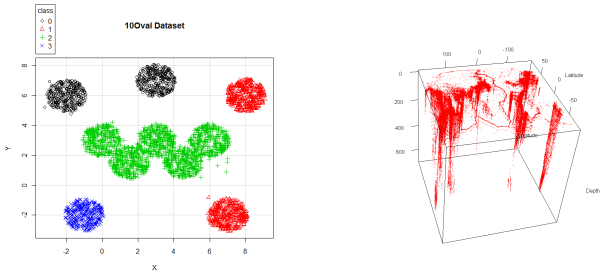

Figure 2. 10oval data set (left) and 3D Earthquakes data set (right).

Earthquake Dataset. The earthquake dataset describes characteristics of 333,561 earthquakes. Due to its size, most commercial, sequential implementations of clustering algorithms will not be able to cluster this dataset. The particular task of this test case is to find regions where both deep and shallow earthquakes are collocated; that is, for which the variance of earthquake-depth for a region is above a given threshold. The fitness function used in this test case, assigns a reward which is proportional to the difference between the observed variance and the threshold. Table 2 gives the main characteristics of this data set, and the right part of Fig. 2 visualizes this dataset.

Table 2. 3D Earthquakes dataset characteristics

data size	330,561 points
attributes	< latitude, Longitude, Depth >
Distance Function	Euclidean Distance
Fitness Function	Variance High (high variance of Earthquake Depth) $i(r) = ((variance(r)/variance(wholeDataset)) - threshold)^{\nu}$ $Fitness = \sum i(r) \times \|r\|^{\beta}$

Yahoo's User Daily Activities Dataset. This dataset contains 3 billion bcookies (browser cookie) which characterize the behavior of anonymous users for a period of 30 days. The goal of analyzing this dataset is to find groups of users who exhibit a similar Internet behavior, which could, for example, be exploited for personalized advertisement. As far as this paper is concerned, we plan to use subsets of the dataset to compare the CUDA versions of CLEVER with PAR-CLEVER, and to derive upper bounds for both approaches with respect to dataset sizes that realistically can be clustered.

Table 3. Characteristics of the Yahoo data set

data size	3,009,071,396 records
attributes	Bcookie id Days Avg. No. of categories for page-view Avg. No. of page-view, no. of search queries per day, no. of search queries per category
Distance Function	Eight dimensional (exclude the Bcookie id) Euclidean distance function
Fitness Function	$i(r) = max(0, a - Avg\_Dist(r))^{\nu}$ where *a* is a distance threshold Avg_Dist(r) is the average distance of objects in the region r to the representative of r $Fitness = \sum i(r) \times \|r\|^{\beta}$

2.2. OpenMP Experimental Results

The three OpenMP versions described in Section 1 have been evaluated on a node of the shark cluster at the University of Houston with two quad-core AMD Opteron processors running at 2.2 GHz and with 8 GB of main memory. The node runs OpenSuSE 11.0 and we used gcc 4.3 as the compiler. Experiments have been executed for 1, 2, 4 and 8 threads. Each test has been executed three times and we report the average execution time

in seconds. Since all experiments run with the same parameters, the clustering results only differ in execution time.

Table 4. Execution time in seconds for the OpenMP versions of PAR-CLEVER for Different Data Sets

Threads	1	2	4	8
100val Dataset				
Parameters: p=100, q=27, k'=10, ν=1.1, threshold=0.6, β=1.6				
Loop-level	233	139	70	41
Loop-level + Incremental Updating	215	126	66	41
Task-level	231	138	69	35
Earthquake Dataset				
Parameters: p=15, q=12, k'=50, ν=2, threshold=1.2, β=1.4				
Loop-level	43,194	21,741	11,946	5,913
Loop-level + Incremental Updating	13,737	7,066	3,851	2,202
Task-level	43,243	21,607	11,366	5,608
Yahoo Reduced Dataset				
p=5, q=12, k'=80, ν=1.2, threshold=0, β=1.000001				
Loop-level	25,554	13,130	6,687	3,691
Loop-level + Incremental Updating	11,652	6,061	3,240	1,825
Task-level	25,696	14,911	9,756	4,872

The results obtained with the three OpenMP versions are shown in Table 4. The results indicate first of all the necessity for parallel computing, since executing some of the data sets took for the sequential version tens of thousands of seconds for some test cases. All three OpenMP code versions tested scale very well. In fact, the speed up obtained with for 2, 4 and 8 threads is very close to theoretical maximum, a linear speedup. Second, the optimization introduced in Section 1 which performs incremental updates is clearly outperforming the original algorithm. For large problems, the performance difference between the original algorithm and the version performing incremental updates is more than a factor of two. Finally, while the performance obtained when using loop-level parallelism and using task-level parallelism are often close, the version relying on loop-level parallelism seems to be typically faster for the large data sets (earthquake, Yahoo), while the task-parallel version performed better for the 100vals data set—we also use a larger sampling rate for this dataset.

From the Yahoo data set we created a smaller subset containing 2,910,613 objects with a size of 253MBs. PAR-CLEVER was allowed to run 10 iterations using 1, 2, 4 and 8 threads respectively.

As far as the final version of the paper is concerned, we expect to have additional data from a set of new nodes that are currently being installed. These nodes have each 48 compute cores available with 64 GB of main memory, and will allow to scale our results up to the according number of threads. Furthermore, two new nodes each containing four NVidia Fermi GPU are currently being installed. The additional hardware will be used to compare the CUDA versions with the OpenMP versions of CLEVER for the two larger datasets.

3. Conclusion

CLEVER is a powerful prototype-based clustering algorithm which supports plug-in fitness functions and employs randomized hill climbing to maximize a given fitness functions. To make it feasible to run CLEVER for large datasets with complex fitness function, we explore various parallelization strategies using OpenMP and CUDA. Our results indicate a nearly linear scalability of the parallel algorithm using multi-core processors can be accomplished. Moreover, some of the presented strategies can be reused for a larger class of problems in which randomized hill climbing is used to solve other challenging optimization problems.

References

[1] O.U. Celepcikay and C. F. Eick. REG^2: A regional regression framework for geo-referenced datasets. In *Proc. 17th ACM SIGSPATIAL International Conference on Advances in GIS (ACM-GIS)*, Seattle, Washington, November 2009.

[2] Wen-mei W. Hwu David B. Kirk. *Programming Massively Parallel Processors – A Hands-on Approach*. Morgan Kaufmann Publishers, 2010.

[3] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39:1–38, 1977.

[4] C. F. Eick, R. Parmar, W. Ding, T. Stepinki, and J.-P. Nicot. Finding regional co-location patterns for sets of continuous variables in spatial datases. In *Proc. 16th ACM SIGSPATIAL Int. Conf. Advances in GIS*, 2008.

[5] D. Foti, D. Lipari, C. Pizzuti, and D. Talia. Scalable parallel clustering for data mining on multicomputers. In *Lecture Notes in Computer Science*, pages 390–398. Springer Verlag, 2000.

[6] Edgar Gabriel, Vishwanath Venkatesan, and Shishir Shah. Towards High Performance Cell Segmentation in Multispectral Fine Needle Aspiration Cytology of Thyroid Lesions. *Computational Methods and Programs in Biomedicine*, 98(3):231–240, 2010.

[7] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Willey & Sons, New York, 1990.

[8] Kittisak Kerdprasop and Nittaya Kerdprasop. Parallelization of k-means clustering on multi-core processors. In *Proceedings of the 10th WSEAS international conference on Applied computer science*, ACS'10, pages 472–477, Stevens Point, Wisconsin, USA, 2010. World Scientific and Engineering Academy and Society (WSEAS).

[9] Johann M. Kraus and Hans A. Kestler. A highly efficient multi-core algorithm for clustering extremely large datasets. *BMC bioinformatics*, 11(1):169+, April 2010.

[10] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, Berkeley, 1967. University of California Press.

[11] V. Olman, Fenglou Mao, Hongwei Wu, and Ying Xu. Parallel clustering algorithm for large data sets with applications in bioinformatics. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 6(2):344 –352, 2009.

[12] OpenMP Application Review Board. *OpenMP Application Program Interface, Draft 3.0*, October 2007.

[13] V. Rinsurongkawong, C-S. Chen, C. F. Eick, and M. Twa. Analyzing change in spatial data by utilizing polygon models. In *Proc. International Conference on Computing for Geospatial Research & Application*, Washington DC, June 2010.

[14] V. Rinsurongkawong and C.F. Eick. Correspondence clustering: An approach to cluster multiple related datasets. In *Proc. Asia-Pacific Conference on Knowledge Discovery and Data Mining (PAKDD)*, Hyderabad, India, June 2010.

[15] Ren Wu, Bin Zhang, and Meichun Hsu. Clustering billions of data points using gpus. In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, UCHPC-MAW '09, pages 1–6, New York, NY, USA, 2009. ACM.