Parallel Implementation of the Slope One Algorithm for Collaborative Filtering

Efthalia Karydi and Konstantinos G. Margaritis
Parallel and Distributed Processing Laboratory
Department of Applied Informatics, University of Macedonia
156 Egnatia str., P.O. Box 1591, 54006 Thessaloniki, Greece
Email:karydithalia@gmail.com
kmarg@uom.gr

Abstract—Recommender systems are mechanisms that filter information and predict a user’s preference to an item. Parallel implementations of recommender systems improve scalability issues and can be applied to internet-based companies having considerable impact on their profits. This paper implements two parallel versions of the collaborative filtering algorithm Slope One, which has advantages such as its efficiency and the ability to update data dynamically. The first presented version is parallely implemented with the use of the OpenMP API and its performance is evaluated on a multi-core system. The second is an hybrid approach using both OpenMP and MPI and its performance is evaluated in an homogeneous and an heterogeneous cluster. Experiments proved that the multithreaded version is 9,5 times faster than the sequential algorithm.

Keywords—Collaborative Filtering; Recommender Systems; Slope One Algorithm; Parallel Programming; MPI; OpenMP;

I. INTRODUCTION

Collaborative filtering based recommender systems introduce the users’ opinion to the procedure of the recommendations generation. Collaborative filtering recommender systems have gained wide popularity. Thus, as the number of users and items of such systems increases, so inevitably does the amount of data. One of the most challenging factors in recommender systems, which is caused due to the data abundance, is the way to achieve high quality recommendations in the shortest time possible. Consequently, a great need is emerging. The achievement of quick data processing, in order to accomplish high quality recommender systems of an increased performance.

In this paper the Slope One algorithm [1] was chosen for parallelization due to the presence of advantages such as its speed and efficacy and the dynamically updatable data. Two parallel implementations are introduced, using MPI and OpenMP, and the experimental results are evaluated.

The rest of this paper is organized as follows: In section II related work is discussed. Section III presents an overview of the different versions of the Slope One algorithm. Section IV presents the parallel implementations of the Slope One algorithm. Finally, the experimental results are analyzed in section V.

II. RELATED WORK

Recently, the main versions of Slope One are being used together with other algorithms, such as data mining techniques, in order to accomplish faster and more effective recommender systems.

In [2] was presented a method that used the Slope One algorithm to produce the predictions of a user, and continued by using the Pearson Correlation metric to calculate the neighborhood of similar items and produce recommendations. The quality of the predictions was affected by the size of the set of similar items. This variation was more accurate than the traditional Collaborative Filtering algorithm.

Another method, first divides the set of items into subsets, taking into account the kind of items requested by the user. In this manner, the dimension of the set of items, the number of ratings and some times even the number of users, is being reduced. The co-clustering of the data is accomplished by using the K-Means algorithm, taking as parameters the demographic data that the user must have previously determined. After the dimensionality reduction, Slope One is being used to the smaller dataset to produce the predictions. This approach reduces the time needed for calculations and augments the predictions’ accuracy [3].

Between other approaches that use the Slope One algorithm is the one presented in [4], which combines the Slope One with Userrank. Userrank is based on Pagerank algorithm and attaches weights to each user, depending on how many related items he has rated. These weights are being used to calculate the differences between the items’ ratings. Another algorithm, that combines item-based with user-based collaborative filtering, was proposed in 2009. This algorithm uses Slope One to fill in the empty spaces of the array containing the ratings and on the new dense array applies user-based collaborative filtering techniques [5]. Recently another recommender system based on Slope One has been designed [6]. This approach selects Slope One for being more efficient in the item similarity calculation than other item-based algorithms.

The need to accomplish fast real-time recommender systems that are able to handle enormous data, has led the research trends to the study and implementation of parallel
regularization algorithm is presented in [8] and uses the concept of decomposition technique, which uses Posix Threads NPTL API on 32 cores and takes 3.2 µs to compute a prediction on Netflix dataset [9].

A distributed algorithm, based on co-clustering, implemented with MPI and OpenMP is presented in [10]. Netflix Prize dataset is used on a 1024-node Blue Gene/P architecture and achieves training time of only 6 seconds and 1.78 µs prediction time per rating. Other parallel co-clustering algorithms exist, as the one described in [11], which simultaneously creates user and item neighborhoods by dividing among the processors the rows and columns of the matrix averages, and the dataflow implementation of a parallel co-clustering algorithm, presented in [12], which uses the Netflix dataset and achieves prediction runtime of 9.7 µs per rating.

Most recent attempts in the field of parallel collaborative filtering algorithms embrace the use of frameworks. To attain better scalability, frameworks such as Hadoop [13] and GraphLab [14] are extensively used. In [15] is implemented a user-based collaborative filtering algorithm on Hadoop, using Netflix dataset on 9 dual-core processors. Item-based collaborative filtering algorithm is implemented on Hadoop in [16]. This approach separates the three most excessive computations into four Map-Reduce phases, which are executed in parallel on a three node Hadoop cluster. An open source collaborative filtering library is implemented in [17], using the GraphLab parallel machine learning framework, and two approaches of SGD on Hadoop are presented in [18] and [19].

[20] implements the Weighted Slope One algorithm using Hadoop. This approach clusters users and assigns weights to each cluster. Then, the ratings are predicted using Weighted Slope One. Lately, the research community has drawn further attention to the Slope One algorithm and many approaches have been published [21], [22], [23], [24].

III. BACKGROUND AND NOTATION

In this section, given an overview of the Slope One algorithm. Slope One defines in a pairwise mode, how much better is one item preferred than another by calculating the difference between the items’ ratings. One of the main characteristics of the algorithm is that only the ratings of users who have evaluated some common items with the user for whom the prediction is being produced and this user’s predictions are introduced in the predictions calculation.

Given a set \( \chi \), consisting of all the evaluations in the training set, and two items \( i \) and \( j \) with ratings \( u_i \) and \( u_j \) respectively, in a user’s \( u \) evaluation \( (u \in S_{j,i}(\chi)) \), the average deviation of \( u_i \) regarding \( u_j \) is given by

\[
dev_{j,i} = \frac{\sum_{i \in S_{j,i}(\chi)} u_i - u_j}{\text{card}(S_{j,i}(\chi))}. \tag{1}
\]

The average deviation of the items is used for the prediction of the rating that the user \( u \) would give to item \( j \),

\[
pred(u, j) = \bar{u} + \frac{1}{\text{card}(R_j)} \sum_{i \in R_j} \text{dev}_{j,i}, \tag{2}
\]

where \( R_j = \{i | j \in S(u), i \neq j, \text{card}(S_{j,i}(\chi)) > 0 \} \) is the set of all relevant items and \( \text{card}(S_{j,i}(\chi)) \) is the number of all the evaluations in the set \( S \) that contain ratings for both items \( i \) and \( j \).

Two additional versions of the Slope One algorithm exist. The Weighted Slope One, in which the number of observed ratings for each item, \( c_{j,i} = \text{card}(S_{j,i}(\chi)) \), is taken into account and the predictions are calculated according to

\[
pred(u, j) = \frac{\sum_{i \in S(u)-\{j\}} (\text{dev}_{j,i} + u_i)c_{j,i}}{\sum_{i \in S(u)-\{j\}} c_{j,i}}. \tag{3}
\]

In the Weighted Slope One version, if a pair of items has been rated by more users than another, this fact affect the predictions.

Another version is the Bi-Polar Slope One, which predicts only if an item will be liked by a user or not. Thus, a two value scale is used, instead of a multivalued, which was used to predict the exact rating a user would give to an item.

IV. PARALLEL SLOPE ONE

Through the constant increase of data, a need for better processing speed acquisition is emerging. To accomplish better speed, two parallel versions of Slope One are implemented and are described in this section. For both implementations C language is being used.

A. Multithreaded Implementation

The first implemented version can be applied to shared memory systems. It is a multithreaded implementation with OpenMP. In order for the predictions to be produced, the values of four arrays have to be calculated. The values of the array that contains the ratings that each user has inserted to the system, the values of the arrays which contain the differences and the frequencies of the items’ appearance in pairs, and finally the values of the deviation matrix. Since these calculations are the most time-consuming part of the code, they are computed in parallel, using the Data Parallel Model. According to this model data is being shared to the threads, and the computations are shared between all threads. The Task Parallel Model has been deliberately avoided, because the calculations needed to the formation of some of the above matrices involve the use of the rest of the matrices. Thus, the use of Task Parallel Model would delay the overall performance. After these calculations, the program calls two
functions. One function computes the predictions and the other, the weighted predictions. A parallel region is defined in each of these functions, and each thread produces the predictions for the items that have not been rated.

As can be seen in the pseudocode presented below, there is a main procedure that initializes the OpenMp routines and creates the threads. Two parallel regions are defined. The first one is used for the computation of the values of the ratings array. In the second are inserted various for directives to compute in parallel the intensive loops needed for the differences, frequencies and deviation matrices. For step 3, a different parallel region is defined. To each thread are assigned different users and each thread computes all the missing ratings that correspond to its users by calling the predictions’ functions. In the predictions’ functions other parallel regions are defined, which are enabled only when nested parallelism is supported.

**Pseudocode of multithreaded implementation.**

Main procedure (Input: The txt MovieLens file. Output: The predictions)

1. Initialize OpenMP routines;
2. All threads compute ratings, differences, frequencies and deviation matrices;
3. For i=0 to users * x items, if (ratings[i]==0) then call predictions and weighted predictions function;

(Weighted) Predictions Function

(Weighted) Predictions()

1. Calculate (weighted) prediction of a given user’s rating for a given item;
2. Return (weighted) prediction;

**B. Hybrid Implementation**

This implementation combines both Shared and Distributed Model, using OpenMP and MPI. The previous implementation is maintained as it was, and MPI is added to it. Multiple OpenMP threads are used under each MPI process. The Master-Workers Model with dynamic processes is being used.

One of the cluster’s nodes is the master node, which is engaged in data partitioning and distribution to the other nodes, the workers. The functions that this implementation has to complete are the coordination of the processes, the computations, the collection of their results and the predictions’ calculation. The master node is responsible for the coordination of the processes, the collection of the computations’ results and the formation of the predictions. All the computations needed to be used for the predictions, are made to the workers.

Both Data and Task parallelism are employed, since the master node distributes different data to the workers and different labors of the program are fulfilled in master and worker nodes. Although communication between nodes is needed, load balance is achieved by the use of dynamic processes. Master node sends fixed size data to the workers. Each time a worker has finished the computations, receives new data, until all data have finished. Then, all workers send their results to the master node, which is responsible for their appropriate placing. Master node also completes the deviations’ and predictions’ computation. Despite the fact that the computations that are realized at the master node affect the total execution time, they reduce the communication cost among nodes.

**Pseudocode of hybrid implementation.**

Main procedure ()

1. Initialize MPI and OpenMP routines;
2. If (process==master) then call master(); else call worker();
3. Exit MPI operations;

Master (Input: The txt MovieLens file. Output: The predictions)

1. All threads set the ratings matrix;
2. Send data blocks to the workers;
3. While (active workers>0), Receive signal that a worker is available to receive new data; Send next or last data or dietag to the available worker;
4. Receive results;
5. All threads arrange results properly;
6. For i=0 to users * items, if (ratings[i]==0) then call predictions and weighted predictions function;

Worker (Receive chunks of ratings matrix, send results for differences, frequencies and deviation matrices)

1. Receive data;
2. If(tag==dietag) then go to step 6; else go to step 3;
3. All threads calculate differences, frequencies and deviation matrices;
4. Send signal to master that is available to receive new data;
5. Receive new or last data;go to step 2;
6. Send the results to master;

(Weighted) Predictions Function

(Weighted) Predictions()


1. Calculate (weighted) prediction of a given user’s rating for a given item;
2. Return (weighted) prediction;
}

Describing more precisely, the calculation of the values of the ratings matrix is taking place only in the master node. Parts of this matrix are shared to the worker nodes, which process the received data and calculate the appropriate parts of the differences, frequencies and deviation matrices. Each worker node that completes this task sends a signal message to the master node to declare its availability to receive new data. Then, master node sends the next part of data to the available worker or a dietag when no more data exist. When a worker node receives the dietag sends the results to the master node. Master node receives the results and is responsible for their appropriate handling and the predictions’ calculation, which is done similarly to the multithreaded implementation.

V. RESULTS AND ANALYSIS

A. Experimental Methodology

The MovieLens dataset, available from GroupLens Research [25], was used for the performance and scalability evaluation of the implementations discussed above. MovieLens 100k was used for performance evaluation, and MovieLens 1M was divided into sub-datasets, augmenting the number of users in each one of them, and was used for scalability analysis.

The results of both implementations were compared to those of the sequential algorithm. To achieve this, the OMP_NUM_THREADS environment variable was used to the multithreaded implementation.

The experiments for the multithreaded implementation were performed on a system consisted of two CPU’s, AMD opteron(tm) Processor 6128 HE, with eight cores each, 800MHz clock speed and 16GB RAM, under Ubuntu Linux 10.04 operating system. The OpenMpt version 3.0 was used and time was measured by its omp_get_wtime() function.

For the hybrid implementation two clusters were used. An homogeneous, consisted of 12 dual-core processors with hyperthreading and Intel(R) Core(TM) i3 CPU’s at 2.93 Ghz and 4 GB memory, all running Ubuntu Linux 10.04 and connected using Realtec Semiconductor Co. Ltd RTL8111/8168B PCI Express Gigabit Ethernet Controllers, and an heterogeneous, consisted of the AMD opteron(tm) Processor 6128 HE, with eight cores described above, an Intel(R) Core(TM)2 Quad CPU Q6600 at 2.40 Ghz and 8 GB RAM, an Intel(R) Core(TM)2 Quad CPU Q9300 at 2.50 Ghz and 8 GB RAM and three Intel(R) Core(TM) i3 at 2.93GHz and 3GB RAM each. All running Ubuntu Linux 10.04. The time was measured using the MPI function MPI_Wtime().

B. Experimental Results

1) Performance Analysis of Multithreaded Implementation: As can be seen in figure 1 the total execution time of this implementation is reducing as the number of used threads is increasing. With the use of 16 threads, the total execution time is reduced by 9 times over the sequential time. The total execution time measured, refers to the computation time and to both predictions’ and weighted predictions’ time and their storage to text files. Thus, the total time needed for the predictions’ production is less than the total execution time measured in this implementation, because only one of the predictions’ functions would be necessary in a recommender system. Using 16 threads, the preprocess time in all datasets is reduced in a range from 13.9 to 15.33 times and predictions are generated faster.

2) Performance Analysis of Hybrid Implementation: In this application is also observed reduction of the total execution time over the sequential execution time. Using the MovieLens 100k dataset on 10 nodes, the parallel implementation is 5.65 times faster than the sequential, and the preprocess time is 15.49 times less. The speed up of this implementation can be seen in figure 2 Varying the data size that is sent each time to the workers, better execution time can be achieved.

The communication time among the system’s nodes,
which is measured at the master node, includes idle time, during which the master remains inactive while waiting for results. This is caused because of the blocking function of MPIRecv. Net communication time has been estimated using the ping-pong method. Figure 3 shows how the communication time including master node’s idle time is reduced by adding more nodes to the system, although net communication time is, as expected, augmenting.

In the heterogeneous cluster, the results of the hybrid implementation vary, depending on the selection of the master node. This occurs due to the calculations that must be completed by the master node. As master node has been chosen the Intel(R) Core(TM)2 Quad CPU Q9300 at 2.50 Ghz, because after testing for the proper master selection, better results were achieved with this one. The use of more powerful worker nodes than the master node has leaded to the minimization of the idle time contained in the communication time. This implementation delivered the results faster, in 14.06 seconds both predictions and weighted predictions were calculated for the MovieLens 100k dataset.

3) Comparison of all implementations.: Although the multithreaded implementation produced both predictions and weighted predictions faster than the other implementations, the hybrid implementation performed on the homogeneous cluster, is the fastest one. Regarding the number of predictions and weighted predictions produced per second, the multithreaded implementation is the most effective. Only 2.2 μs are needed per rating and 1.31 μs per rating using weights.

In all implementations was noticed that the number of predictions and weighted predictions per second tends to stabilize to a certain number, as the density of the different datasets remains the same. In table I can be seen the density of the different datasets and the number of predictions per second on each dataset for the multithreaded implementation. In table II can be seen some numerical results of all the implementations.

VI. CONCLUSION

In this paper two parallel implementations of the Slope One algorithm have been described and their performance has been evaluated. Improvement in the execution time has been achieved, up to 9.5 times over the sequential execution time, fact that proves that further optimization of the presented approaches will not be in vain.

The hybrid implementation on the homogeneous cluster was proved slowest than the multithreaded implementation, due to the presence of idle time on the master node. To minimize the idle communication cost, an heterogeneous system was used and assigned worker nodes more powerful than the master node, which proved to be successful. A complementary strategy could be the overlap of communication and computation, that is sending data and calculating of results.

Since both implementations reduce significantly the offline computation time, profits are expected from their use. Companies will be able to produce faster the recommendations, and multiple queries can be addressed to each node.
used. Therefore, more users can be served simultaneously.

In future, further optimization techniques are intended to be performed in order to improve memory usage, using for example sparse matrix storage techniques, execution time, especially in the case of hybrid heterogeneous configurations, and scalability, by testing the parallel implementation against larger and different data sets. Furthermore, a complete statistical analysis of the behavior of the above parallel algorithms while using different datasets is also intended to be undertaken.

REFERENCES

[1] Lemire, D., Maclachlan, A.: Slope One Predictors for Online Rating-Based Collaborative Filtering. SIAM Data Mining (SDM ’05), Newport Beach, California (2005) 471–476


179