Hybrid Implementation of Alternative Least Square Algorithm

Pragati Chopade, Manjunath Hegde
Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore, India
pragati.chopade06@gmail.com
serchegde@serc.ssl.iisc.ernet.in

***Abstract*—This article mainly discusses about the parallelizing the Alternative Least Squares Approach. Parallelization is done using CPU and GPU i,e the hybrid version. This improves the speedups as the number of processors increase. This technique is used in Collaborative Filtering approach in recommender systems. Datasets used are Movielens datasets with 100K and 1M non-zero entries. The significant improvements in speedup in Hybrid version over the MPI version is shown further.**

I. Introduction:

Recommender systems is a family of methods that enable filtering through large observation and information space in order to provide suggestions. A common goal of recommender systems is helping customers sort through offered products to find the ones they will enjoy. They do so through personalized recommendations. We will use two matrices one for users and one for movie. A movie profile could include attributes regarding its genre, the participating actors, its box office popularity, etc. User profiles might include demographic information or answers to a suitable questionnaire. The resulting profiles allow programs to associate users with matching products.

Collaborative Filtering(CF) is a subset of algorithms that exploit other users and items along with their ratings(selection, purchase information could be also used) and target user history to recommend an item that target user does not have ratings for. Fundamental assumption behind this approach is that other users preference over the items could be used recommending an item to the user who did not see the item or purchase before.

We have users u for items i matrix as in the following:
Qui=r if user u rate item i
0 if user u did not rate item i

We first estimate factor movie matrix(Y) using factor user matrix(X) and estimate user matrix using movie matrix. After enough number of iterations, we are aiming to reach a convergence point where either the matrices X and Y are no longer changing or the change is quite small. We have neither user full data nor full items data, this is also why we are trying to build the recommendation engine in the first place. Therefore, we may want to penalize the movies that do not have ratings in the update rule. By doing so, we will depend on only the movies that have ratings from the users and do not make any assumption around the movies that are not rated in the recommendation. Let's call this weight matrix wui as such:

Wui=0 if Qui $=0$
1 otherwise

Then, cost functions that we are trying to minimize is in the following:

$\mathrm{J}(X\mathrm{u}) = (\mathrm{QuXu}Y)\mathrm{Wu}(\mathrm{QuXu}Y)^T + \mathrm{Xu}(\mathrm{Xu})^T$
$\mathrm{J}(Y\mathrm{i}) = (\mathrm{Qi}X\mathrm{Yi})\mathrm{Wi}(\mathrm{Qi}X\mathrm{Yi})^T + \mathrm{Yi}(\mathrm{Yi})^T$

Solutions for factor vectors are given as follows:

$X\mathrm{u} = (Y\mathrm{Wu}(\mathrm{Y})^T + I)^{-1} Y\mathrm{WuQu}$
$Y\mathrm{i} = ((\mathrm{X}^T)\mathrm{Wi}X + I)^{-1}\mathrm{X}^T\mathrm{WiQi}$

With these final factor matrices, user will be recommending the movies which he might be interested in. If we want to have the both content and collaborative filtering's good worlds, then we could adopt a hybrid approach i.e Cascade (Content + Collaborative Filtering) :After content based filtering, we also apply collaborative filtering to refine the recommendations even more.

II. Datasets used:

1. MovieLens Dataset 1: 100k ratings from 1000 users for 1700 movies
2. Movielens dataset 2: 100 million ratings from 6000 users for 4000 movies

The above datasets are sparse matrices, so we are using an iterative method here.

III. Related papers and limitations:

A. 1.*Large scale Parallel Collaborative Filtering for the Netflix Prize(Yunhong Zhou,Dennis Wilkinson,Robert Schreiber and Rong Pan:*2008*)*

In this paper, they have described Alternating-Least-Squares with Weighted--Regularization(ALS-WR), a parallel algorithm that we designed for the Netflix Prize, a large-scale collaborative filtering challenge. They use parallel Matlab on a Linux cluster as the experimental platform. They show empirically that the performance of ALS-WR monotonically increases

with both the number of features and the number of ALS iterations.

B. 2.Alternating Least Squares for Low-Rank Matrix Reconstruction(Dave Zachariah,Martin Sundin,Magnus Jansson and Saikat Chatterjee:2012)

For construction of low rank matrices from undersampled measurements, they develop an iterative algorithm based on least squares estimation. While the algorithm can be used for any low-rank matrix, it is also capable of exploiting a-priori knowledge of matrix structure. In particular, they consider linearly structured matrices, such as Hankel and Toeplitz, as well as positive semidefinite matrices. The performance of the algorithm, referred to as alternating least-squares (ALS), is evaluated by simulations and compared to the Cramer-Rao bounds.

IV. PARALLEL ALGORITHM FOR ALTERNATING LEAST SQUARE ALS USING MPI:

1) The users items matrix Q is stored in binary format matrix W. It is obtained by placing 1 where the rating is present and 0 where the rating is absent.

2)The users-items matrix is stored in CSR format being a very sparse matrix. This matrix is stored at every processor.

3)The Q matrix is factored to get X and Y. These two decompositions are randomly initiated at every processor where X is users matrix and Y is items matrix.

4) Each processor is assigned some fixed no of users ie. rows of X matrix and some fixed no of items i.e columns of Y matrix. Then each processor will compute user rating for its assigned users using the initial Y(items matrix) matrix using the following equation,

$X\text{u} = (Y\text{Wu}(\text{Y})^T + I)^{-1} Y\text{WuQu}$

and it will also simultaneously compute item rating for its assigned items using initial X(users matrix)matrix using following equation,

$Y\text{i} = ((\text{X}^T)\text{Wi}X + I)^{-1}\text{X}^T\text{WiQi}$

These values are obtained by giving the above equations to the gaussian solver function implemented in our code. Also in each iteration, the cost functions for xu and yi which we are trying to minimize are obtained using following equations,

$\text{J}(X\text{u}) = (\text{QuXu}Y)\text{Wu}(\text{QuXu}Y)^T + \text{Xu}(\text{Xu})^T$
$\text{J}(Y\text{i}) = (\text{Qi}X\text{Yi})\text{Wi}(\text{Qi}X\text{Yi})^T + \text{Yi}(\text{Yi})^T$

These values are updated at all processors using MPI ALL GATHER.

5) In the next iteration, the X and Y matrices used will be the updated once. And the iterations will stop when the minimum values of both cost functions will be obtained.

6) Thus at the end we will get updated X and Y matrices at every processor and these will be sent to the root processor which will then compute X*Y and this will constitute updated matrix Q. Similarly this Q will be converted into binary matrix W . And the error will be computed using entries of W and W.

V. HYBRID(GPU+CPU) ALGORITHM FOR ALTERNATING LEAST SQUARES:

The algorithm involves many matrix and vector operations. This motivates to use GPUs to achieve fine grain parallelism. The construction of the CSR format is same as mpi. All the three vectors namely row, column and value are stored in each process. This is required as the construction of a vector for a item or user needs CSR vectors.

Gauss Jordan Iteration method: The time consuming step in the updating user and item matrix is the calculation of inverse $X^T\text{X}=\text{A}\ and\ \text{Y}Y^T = B$.

The usual matrix inverse calculation using cofactors and determinant may not work if the matrix is singular. Also,the time taken for calculating matrix inverse is very high $O(\text{n}^4)$. On the other hand Gauss Jordan iterative method takes less $O(\text{n}^3)$ time and it can be parallelizable.

Method of simultaneous updates of user and item matrix was not considered as it requires more memory. The simultaneous update requires multiple temporary matrices to be kept in memory. It also results in idle waiting time as the computation of A and B inverse takes more time. To reduce this idle waiting time, calculation of inverse for X^TX and $\text{Y}Y^T$ are done one after the other. While inverse of $Y\text{Y}^T$ is being calculated,the rows of user matrix are updated with calculated X^TX inverse. Once the inverse of the matrix $Y\text{Y}^T$ is computed,the columns of item matrix are updated.

The updating the item and user matrix requires matrix vector multiplication. The rows and columns of user and item matrix respectively are divided across processes. Each process launches kernel and loads data that is allocated to it. The number of threads and block size are decided based on the size of the matrix and vector. In our implementation, the block size is multiple of 32 or close to that in order to increase the speedup.

GPU is also used to compute $Y\text{Y}^T$ and X^TX. The concepts of warp is considered here also to achieve better result. Each process will handle few rows of user matrix and columns of item matrix.

Computation involves regularization of values $\text{Xu}(\text{Xu})^T$ where X is a user matrix and is a learning factor)in order to avoid the overfitting the data. This task is done in CPU while GPU is updating the user or item matrix. Load balancing between CPU and GPU is achieved to certain extent by doing this.

Allgather is used to communicate the user and item matrices. Since each process calculates only part of the matrix,after every iteration allgather is used to get the latest values of user and item matrices. This adds the MPIBarrier and hence some synchronisation time at the end of each iteration. The stopping criteria for the algorithm is the cost function value. When the change in the user and item matrix between two iterations is less than the threshold then we stop.

VI. Experiments and Observations:

Project has MPI and hybrid modules. Experiments are conducted on different datasets with different number of processors. The serial, MPI and Cuda implementations are compared for each dataset.

MovieLens data set has the movie ratings from different users. Experiments are conducted with datasets containing different number of users and movies. First dataset chosen has ratings from 1000 users for 1700 movies. It has total of 100k ratings. Second data set has ratings for 4000 movies given by 6000 people. Each line in the file has a user id, movie id and a value between 1-5 which is the rating from the user. Columns of the dataset is sorted according to userid and movie id.

The execution time for each dataset with serial, mpi and hybrid implementation are given in the below section.

The high parallelism using GPUs has reduce the total execution time of the algorithm. The hybrid approach works better than mpi as expected. The speedups obtained by parallel algorithms scales with number of processes. The one more observation regarding matrix factorisation is that the rank of the matrix to be considered affects accuracy and the run time of the algorithm. Since the rank of the ratings matrix unknown, dimensionality of user and item matrix will be unknown. So try and error method is adapted to find the approximate rank of the matrix. This algorithm takes more iterations to converge when the initialisation of matrix is not close to actual solution.

VII. Results:

Execution time for Sequential (number of process = 1), MPI and hybrid method are given in the below tables. Execution time is the average of the time taken by all the processes. Number of iterations are kept same in all the implementations. Speedup is calculated as the time taken by one process to time taken by n processes. In case of hybrid speedup is with respect to time taken by 1 CPU + 1 GPU to n processes and m GPUs used.

Jumpshot is used to analyse the MPI implementation of the project. The collective communication (mpi allgather) and mpi barrier were the communication overhead. The speedup of the program was affected by mpi barrier as it is required after each iteration, In case of hybrid implementation, the nvvc profiler is used to analyse execution of the program. The computation to memcpy ration was more than 90 percent. The percentage of time when memcpy is being performed in parallel with compute was low. This is because of the nature of the computation requires all the data in the beginning.

A. *Results for MPI:*

VIII. Execution Times:

For a Movielens dataset(100K)	
Number of Processors	Execution Time
1	16.1981
2	9.8605
4	5.5415
8	3.0611

For a Movielens dataset(1M)	
Number of Processors	Execution Time
1	73.2076
2	42.5024
4	24.2326
8	13.1413

IX. Speed-ups:

For a Movielens dataset(100K)	
Number of Processors	Speedup
1	1
2	1.64
4	2.92
8	5.29

For a Movielens dataset(1M)	
Number of Processors	Speedup
1	1
2	1.7225
4	3.0235
8	6.01

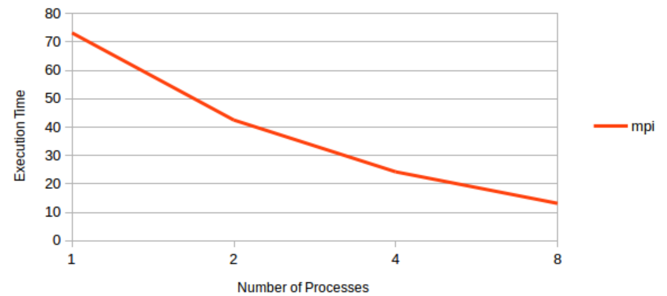

Fig. 1. Execution Times for MPI on dataset(1M)

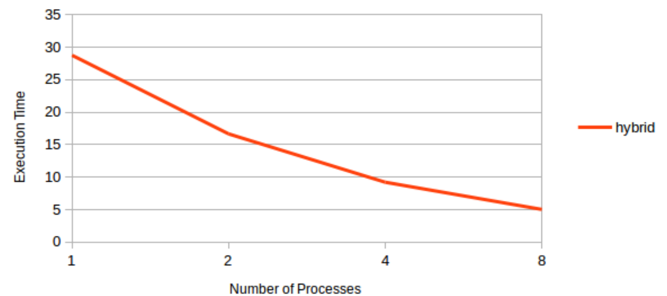

Fig. 2. Execution Times for HYBRID on dataset(1M)

A. *Results for Hybrid(CPU + GPU):*

X. EXECUTION TIMES:

For a Movielens dataset(100K)	
Number of Processors	Execution Times
1	9.6894
2	6.4391
4	3.2124
8	1.9653
For a Movielens dataset(1M)	
Number of Processors	Execution Times
1	28.4356
2	16.6721
4	9.4853
8	5.0263

XI. SPEED-UPS:

For a Movielens dataset(100K)	
Number of Processors	Speedup
1	1
2	1.50
4	3.01
8	4.93
For a Movielens dataset(1M)	
Number of Processors	Speedup
1	1
2	1.7273
4	3.1245
8	5.7223

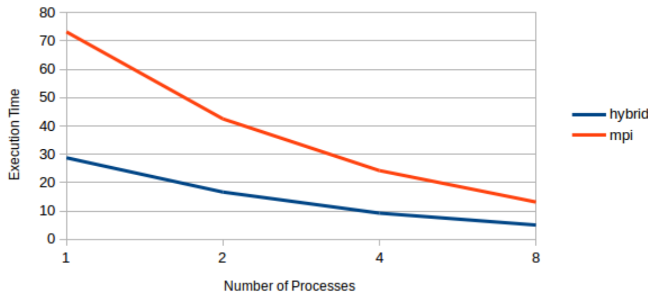

Fig. 3. Comaparison of Execution Times for both MPI and Hybrid on dataset(1M)

XII. Conclusions:

One of the primary observation is the scalability of the algorithm. The execution time reduces as and when the number of processes or gpu cores are increased. So this algorithm can be implemented in large distributed systems like hadoop. The speedup of the algorithm is also constrained by the fact that Gauss Jordan iterative method computes row reduction in a column wise manner i.e each element of the row is computed in parallel, but not across columns.

XIII. Future Work:

As discussed earlier, the finding rank of the matrix is done in try and error method. Methods like SVD decomposition can be used on the dataset to find the rank of the rating matrix. Better matrix initialisation techniques can be implemented so that the number of iterations required for convergence is less.

XIV. References:

1. Large-scale Parallel Collaborative Filtering for the Netflix Prize (Yunhong Zhou,Dennis Wilkinson,Robert Schreiber and Rong Pan:2008)

2. Alternating Least-Squares for Low-Rank Matrix Reconstruction (Dave Zachariah,Martin Sundin,Magnus Jansson and Saikat Chatterjee:2012)

3. Data sets from Movie lens (http://grouplens.org/datasets/movielens/)