Parallelized Matrix Factorization with GPU for Recommender System

Zhifan Liang
Department of Electrical and Computer Engineering
Washington State University, Pullman, WA, USA
zhifan.liang@wsu.edu

***Abstract*—Recommender system is an important part of a content-based online service. To give precise recommend to users, the system needs to calculate the estimated users' ratings from a sparse matrix of known user ratings. This sparse matrix factorization is key to the recommender system's accuracy and performance. Parallelize this algorithm to accelerate it by GPU can provide a noticeable performance improvement. This paper discussed two implementations of sparse matrix algorithms and compare them with other known approaches.**

I. Introduction

Matrix Factorization is an important part of the rrecommender system. The recommender system factorizes a matrix of users' rating, which is a sparse matrix, into two dense matrices represent the users' bias and items' bias. Using these two dense matrix, Recommender System can calculate the unknown element in the rating matrix as the estimated rating of users. With the estimated rating, the system can make an accurate recommendation to each individual user. With GPU accelerate the process, matrix factorization can be parallelized to obtain the accurate result within less time.

A. Research Problem

The recommender system needs to calculate the estimated rating of an item for each user and the known rating matrix is sparse. Sparse matrix factorization in a recommender system requires high accuracy while processing data as fast as possible.

B. Proposed Solution

In this paper, we propose two methods to calculate matrix factorization for the recommender system. One relies on the random number to guess the result and another one calculates with a Singular value decomposition (SVD) algorithm.

C. Contributions

In this paper, we implement two algorithms for matrix factorization for the recommender system. We compare those two algorithms' performance, along with other common implementation algorithms for recommender systems. With those evaluations, we propose the limitation and possible optimization for those algorithms.

D. Outline of this paper

This paper is organized as follows. Sections II contains the summary of previous work on matrix factorization for recommender systems. Section III provides a detailed description of the two approaches we implemented with design ideas and explanations. We evaluate the two algorithms in section III, compare them with their accuracy and performance in Section IV. Section V provides the evaluation. Section VI provides the conclusion we get and the future work. Acknowledgment and references are listed in the end.

II. Related Work

Kilitcioglu et al.[1] proposed optimization of Stochastic Gradient Descent (SGD). They implemented SGD into a recommender system with parallelization. There were three algorithms in their paper, first one is SGD Kernel, which helps to iterate the data. Their second algorithm is focused on calculating the loss, and the third algorithm calculates prediction in order to improve the accuracy. In their paper, they found they reached a 10 times speed improvement compared to the sequential approach.

He et al.[2] proposed an algorithm to parallelize the K-SVD sequential algorithm. They found current K-SVD algorithm does not fully utilize the GPU resource, resulting in the sequential dictionary update. In this case, they proposed an algorithm with a parallel updating dictionary. As a result, the algorithm reaches more than 300 times speedup compared to the original algorithm running in MATLAB.

For the group recommender system approaches, Hu et al.[3] explored an algorithm that implements SVD as the base of the recommender system. They found when recommending for a group of users, combine the least strategy and average strategy into the algorithm will result in higher accuracy.

Snopce et al. proposed a parallelized Jacobi method for calculating SVD. They calculate SVD with Jacobi rotation and parallelized the algorithm.

Gupta et al.[5] discussed the sparse matrix factorization and scalable parallel algorithms. They discussed the multi-frontal algorithm and its implementation for sparse matrix factorization. Then they proposed the parallel method of the algorithm. With the improved scalability of their algorithm, they found their algorithm significantly reduced data movement while processing.

III. Methodology

This section explains two approaches implemented for calculate the sparse matrix factorization.

A. *First Approach*

For the first approach, the algorithm random generate a set of the parameter of the item features. Then use the guessed random features as a known value, guess another set of user feature parameters. The algorithm calculates the error of the guess with the, then calculates the total error with the following equations. After a few iterations, the algorithm determines the random area with the total error. With this algorithm, gradually calculate every feature.

R is the rating matrix, a, b, c, x, y, z are features parameters of user and item. In this case we use three parameters each. G_n is the guess error of each iteration. S_t is total error, the sum of all G_n.

$$R_{ij} = a_i x_j + b_i y_j + c_i Z_j$$

$$G_n = a_n x_j + b_n y_j + c_n z_j - R_{nj}$$

$$S_t = \Sigma G_n$$

The challenge to this approach is to determine how to modify the random boundary of each iteration. We propose a standard to determine the boundary in TABLE I. Another challenge is in this algorithm, every iteration needs to wait for the previous iteration to complete before it can start. To address this issue, we let each thread handle all iterations for one user and parallel compute multiple users' data. With this design, we can avoid the data movement between threads and reduce the dependency between different threads.

TABLE I. Determine random area

St	S(t-1)	Area
>0	>0	0 : L(t)
>0	<0	H(t-1) : L(t)
<0	>0	H(t) : L(t-1)
<0	<0	H(t) : 1

H(t) represents higher boundary of iteration t.

L(t-1) represents lower boundary of iteration t-1.

B. *Second Approach*

For the second approach, the algorithm calculates the SVD factorization result of the sparse rating matrix. The SVD equation is shown below:

$$M = U * \sum * V^T$$

The matrix $\sum$ contains the importance weight of the other two matrices, so we can use it as a standard to form the output matrices. The algorithm selects the biggest elements of the matrix $\sum$, the total number of selections depends on the input needed features number. With the selected elements in the matrix $\sum$, the algorithm extracts the needed row and column from matrix U and matrix V^T to form the output matrix contains user feature parameters and item feature parameters.

In this approach, we set an input value K as the number of features. K defines the number of elements we need in the matrix $\sum$. In this case, we can input various K to find the relationships between accuracy and feature numbers.

IV. Results

The approach two is function as expected. We have tested a set of number of features. The eclipse time is shown in TABLE II. (Time based on a sparse matrix with the row number: 4039, column number: 4039, number of none zero: 88234)

TABLE II. Eclipse Time

No. of Features	Time (ms)
20	9822
40	9817
60	9840
80	9904
100	9907

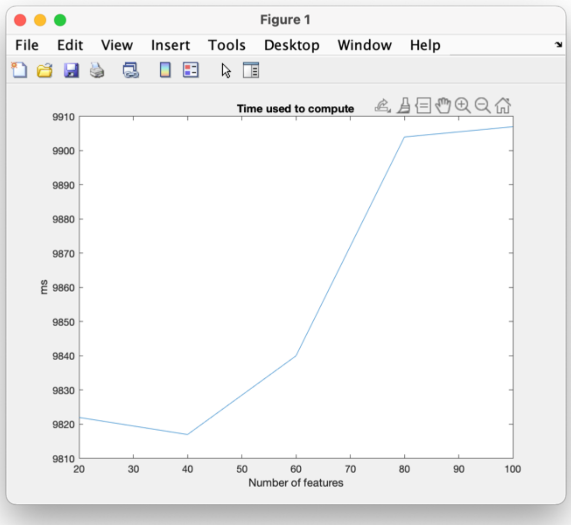

Fig. 1. Eclipse Time with the increasing number of features

As we can found, while increasing the features number can help improve the accuracy, it takes longer time. It is important to find the balance between accuracy and process time. The trend is shown in Fig. 1, we can find the time significant increased from 40 to 80. It is worth noting the time actually decreased from 20 to 40.

V. Evaluation

The first approach has low accuracy while calculating the matrix factorization. This approach cannot guarantee the accuracy increase as the iteration continues.

The second approach uses part of the SVD result to form the output matrices. This method reduced the accuracy compared to the iteration approaches such as Kilitcioglu et al. [1] proposed algorithm. But with dynamic features number, we can control the accuracy with this algorithm. For the intended vague implementation of recommender system such as privacy control, this approach can offer dynamic accuracy balance the privacy and recommender accuracy.

VI. Conclusion and Future Work

Matrix factorization for recommender systems is widely used in real life. Matrix factorization algorithm can be improved in multiple methods such as adding constraints when iterating, adjust parameters to keep the iteration improvement effective. In most cases, parallel algorithms will obtain advantages compare to sequential algorithms, accelerate the parallel algorithm will be an important part of future recommender system algorithm.

The next step should be to continue optimizing the parallel algorithm for GPU calculation. As GPU technology quickly evolving, it is essential to adapt algorithms with new technology as well as optimize it to fit increasing demand of recommender system.

References

[1] D. Kilitcioglu, N. Greenquist, M. Zahran and A. Bari, "GPU Accelerated Matrix Factorization for Recommender Systems," 2021 IEEE 6th International Conference on Big Data Analytics (ICBDA), 2021, pp. 272-281, doi: 10.1109/ICBDA51983.2021.9403110.

[2] L. He, T. Miskell, R. Liu, H. Yu, H. Xu, and Y. Luo, "Scalable 2d k-svd parallel algorithm for dictionary learning on gpus," *Proceedings of the ACM International Conference on Computing Frontiers*, 2016.

[3] X. Hu, X. Meng, and L. Wang, "SVD-based group recommendation approaches," *Proceedings of the 2nd Challenge on Context-Aware Movie Recommendation - CAMRa '11*, 2011.

[4] H. Snopce, A. Aliu and A. Luma, "The block Jacobi Method for The SVD-A Parallel Approach," 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), 2020, pp. 1-5, doi: 10.1109/ICECCE49384.2020.9179448.

[5] A. Gupta, G. Karypis and V. Kumar, "Highly scalable parallel algorithms for sparse matrix factorization," in IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 5, pp. 502-520, May 1997, doi: 10.1109/71.598277.