Parallel Algorithm of Improved FunkSVD Based on GPU

Yue Xiaochen¹, Liu Qicheng²
¹School of Computer and Control Engineering, Yantai University, Yantai, China
²School of Computer and Control Engineering, Yantai University, Yantai, China

Corresponding author: Liu Qicheng (e-mail: ytliuqc@163.com).

This work was supported in part by the National Natural Science Foundation of China (62172351).

ABSTRACT Nowadays, the increasing amount of network data information and the development of big data technology have brought development opportunities and challenges to the recommendation system. The model-based collaborative filtering algorithm has become one of the mainstream algorithms in the recommendation system. For example, the Funk Singular Value Decomposition (FunkSVD) algorithm. However, in the face of big data calculations, data sparseness and iterative oscillations often affect the accuracy of the FunkSVD algorithm. Moreover, when the data volume is in units of GB or more, the FunkSVD algorithm runs slowly and is not effective. Therefore, we propose an improved FunkSVD algorithm (IFABG) based on RMSProp (Root Mean Square Prop) and GPU (Graphics Processing Unit) to solve this problem. Firstly, we use RMSProp algorithm to improve the traditional FunkSVD algorithm, alleviate data sparseness and iterative shock, and improve the prediction accuracy of the algorithm. Next, we implemented the parallelization of the improved FunkSVD algorithm in the GPU, which increased the calculation speed of the algorithm. Finally, we verify the IFABG algorithm under the Movielens dataset. The experimental results show that the IFABG algorithm is very suitable for processing sparse data, and it alleviates the iterative shock, and the prediction accuracy rate is about 30% higher than that of the traditional FunkSVD algorithm. The experimental results also show that the IFABG algorithm has a good acceleration effect. Under the same size data set, the IFABG algorithm is faster than the traditional FunkSVD, and the acceleration ratio can be as high as 19.27.

INDEX TERMS CUDA, FunkSVD, GPU, RMSProp

I. INTRODUCTION

With the explosive growth of Internet information, what people are now facing is not the lack of information but the proliferation of information, that is, information overload[1]. People often cannot filter out the most suitable information from the massive heterogeneous information in a short period. Collaborative filtering recommendation algorithm is widely used as an information screening technology[2][3]. However, with the increase of data volume and data complexity, collaborative filtering algorithms often face problems such as data sparseness, cold start, and scalability. In this case, the collaborative filtering algorithm based on the matrix factorization model can handle big data well[4]. Sui and Yin[5] fused Proportional-Integral-Derivative-incorporated Stochastic Gradient Descent-based Latent Factor Analysis (PSL) model with the Biasvd algorithm to solve the problem of slow algorithm convergence. Wang[6] proposed an improved SVD++ (Singular Value Decomposition++) algorithm for the low computational efficiency of the SVD++ algorithm, which uses backtracking search to accelerate SVD++. Walek[7] proposed a single-chip hybrid recommendation system called Predictory, which combines a recommendation model composed of SVD (Singular Value Decomposition), content-based system and fuzzy expert system.

Collaborative filtering algorithms based on matrix factorization models include SVD, FunkSVD (Funk Singular Value Decomposition), SVD++, and Biasvd. Among them, the FunkSVD algorithm decomposes the original scoring matrix to obtain two low-rank matrices, which can reduce the dimensionality of the data. Compared with other matrix factorization algorithms, FunkSVD algorithm is simpler and the algorithm complexity is lower, which is very suitable for
processing big data. Therefore, this article uses the FunkSVD algorithm for big data recommendation[8].

However, in the face of tens of millions of users and tens of millions of projects, the running speed of traditional collaborative filtering algorithms is relatively slow. Relying solely on the computing power of traditional stand-alone computers can no longer meet the needs of applications. There is an urgent need for new technology to provide more powerful calculation ability. The maturity general-purpose computing technology of GPU (Graphics Processing Unit) has just solved this demand[9]. GPU is a multi-core processor used in graphics and image processing. It is designed to parallelize computationally intensive tasks. It has very strong computing power and very good data throughput. In scientific research and practical applications, the processing and computing task modules that can be parallelized and less logic in the system are often transplanted to the GPU for execution, which can usually achieve a substantial performance improvement. Kang[10] runs the non-negative matrix factorization algorithm on the GPU platform, which accelerates the running speed of the algorithm and improves the effectiveness of the algorithm. Jin[11] proposed an efficient GPU algorithm to solve the matrix decomposition problem based on Stochastic Gradient Descent (SGD) method and improve the efficiency of the algorithm.

Although the traditional FunkSVD algorithm solves the problem that the SVD algorithm needs to fill in the missing values, as the number of data increases and the data sparsity increases, the accuracy of the algorithm will decrease. In addition, the FunkSVD algorithm uses the gradient descent method to update the parameter values, which may cause iterative oscillations, thereby reducing the accuracy. In response to this problem, the paper proposes the Improved FunkSVD Algorithm Based on GPU (IFABG). The algorithm not only has a high prediction accuracy when processing big data, but also a fast calculation speed. The highlights of this paper are made a summary as follows:

- We use GPU to implement the parallelization of the improved FunkSVD algorithm. The algorithm performs more complex operations such as input data and parameter settings in the CPU, and the simple operations of updating parameters and calculating the inner product between the parameters are mainly performed in the GPU. The operation in the GPU is parallel, so it can update the parameter value and calculate the inner product in parallel, thereby reducing the running time of the algorithm and improving the effectiveness of the algorithm in processing big data.

The rest of this article is organized as follows: Section 2 mainly introduces some related algorithms. Section 3 introduces the derivation process of the IFABG algorithm. Section 4 conducts experimental verification and analysis. Finally, Section 5 summarizes the work of the paper.

II. RELATED WORKS

A. TRADITIONAL FUNKSVD ALGORITHM

The traditional FunkSVD algorithm is proposed to solve the problem of the low computational efficiency of the SVD algorithm and the need to fill in missing values. The so-called completion of missing values can also be understood as sparse data. Data sparseness means that most of the data in the data set is missing or zero. Because it is impossible for each user in the recommendation data set to comment (rating) on all items, only some users' ratings for certain items are recorded in the data set. This phenomenon can lead to the problem of missing values in the data set[12][13]. Although the FunkSVD algorithm is simple, it is indeed very extensive in practical applications and pushes the model-based recommendation algorithm to a new level.

The basic idea of the FunkSVD algorithm can be described as given a scoring matrix \( R = [r_{nm}]_{n \times m} = [r_1, r_2, \ldots, r_l] \), find an approximate matrix \( A \) to approximate \( M \) to predict the missing score. That is, decompose \( R \) into a user matrix \( X \) and an item matrix \( Y \), satisfying \( A = XY \approx R \). A column vector in the matrix \( R \) can be interpreted as the weighted sum of all column vectors in the user matrix \( X \), and the weight vector is an element in the corresponding column vector in the item matrix \( Y \). From the mathematical point of view, the FunkSVD algorithm is to reduce the dimensionality of a two-dimensional matrix and convert it into two low-rank matrices. This is a very effective method of decomposing multi-dimensional data. In terms of calculation, calculating two one-dimensional matrices is simpler than calculating a two-dimensional matrix, which greatly reduces the complexity of the calculation.

In fact, the FunkSVD algorithm transforms solving two optimal low-rank matrices into optimization problems. We need to define a loss function[14] such as (1), which is used to control the deviation of the model so that the gap between
the predicted score and the actual score is as small as possible.
\[ e^2 = \sum_{(m,n) \in T} (r_{mn} - x_m^T y_n)^2 \]  

(1)

Where \( T \) is the data set, \( x_m \) is the user feature matrix-vector, and \( y_n \) is the item feature matrix-vector.

However, when the FunkSVD algorithm is applied to the recommendation system, a regularization term is usually added to the loss function to control the variance of the model and prevent over-coupling, that is, to obtain a simple hidden feature vector[14]:

\[ E = \sum_{(m,n) \in T} (r_{mn} - x_m^T y_n)^2 + \lambda(\| x_m \|^2 + \| y_n \|^2) \]  

(2)

Where \( \lambda \) is the regularization coefficient.

Then, FunkSVD algorithm is transformed into the optimization problem of solving (3).

\[ \min_{x_m, y_n} E \]  

s.t. \( 1 \leq m \leq l, 1 \leq n \leq r \)  

(3)

The traditional FunkSVD algorithm uses SGD[15] to update parameter values and its update rule is:

\[ t_{n+1} = t_n + \alpha(-\frac{\partial}{\partial t} f(t)) \]  

(4)

B. RMSProp Algorithm

Deep learning is a new research direction in the field of machine learning, and it plays an important role in search technology, data mining, machine translation and other fields. The optimization algorithm in deep learning plays an important role in solving many practical problems[16]. For optimization algorithms, how to choose a suitable learning rate is very important[17]. The traditional gradient descent method uses the same learning rate for all parameters, so such a learning rate cannot adapt to the characteristics of all data, resulting in iterative oscillations and preventing convergence to the optimal solution. The so-called iterative oscillation means that each update may not proceed in the normal direction, and the parameter update has high variance, which causes the loss function to fluctuate sharply[18][19].

Adaptive Gradient Algorithm (AdaGrad) is an adaptive learning rate[20] algorithm. It uses a different learning rate for each update of each parameter. The learning rate is the proportion of the current gradient in the sum of all historical gradients. The AdaGrad algorithm does not require manual adjustment of each learning rate, and is suitable for processing sparse data. Specifically, the low-frequency parameters are updated with a larger learning rate, and the high-frequency parameters are updated with a smaller learning rate[21]. This feature is very suitable for processing sparse data. However, the AdaGrad algorithm still relies on manually setting a global learning rate. The learning rate is set too large, and the adjustment of the gradient is too large, causing the problem of a sharp drop in the learning rate. The mid-to-late gradient is close to 0, which makes the training end early[22]. In order to solve the problem of the sharp drop in the learning rate of the AdaGrad algorithm, Geoffrey Hinton proposed an adaptive learning RMSProp algorithm[23]. The only difference between the RMSProp algorithm and the AdaGrad algorithm is that the method of solving the cumulative square gradient is different. The RMSProp algorithm adds an attenuation coefficient to control how much historical information is obtained so that the gradient of the parameter moves in a more gentle direction.

When the RMSProp algorithm updates the parameters, the gradient moves in the direction of the exponentially weighted average of the square of the parameters. In this way, the learning rate of each element will no longer decrease (or remain unchanged) during the update process. The larger the derivative of a certain dimension, the larger the exponentially weighted average; the smaller the derivative of a certain dimension, the smaller the exponentially weighted average. This ensures that the derivatives of all dimensions are at the same order of magnitude, which avoids the drop-in learning rate. Therefore, it is very suitable for dealing with non-stationary targets, which can avoid the iterative oscillation problem of the gradient descent method and accelerate the training speed[24].

The gradient calculation of RMSProp algorithm is that given hyperparameter \( 0 \leq \gamma < 1 \), RMSProp algorithm calculates the average exponential decay at time step \( t \geq 0 \):

\[ E[g_t^2] = \gamma E[g_{t-1}^2] + (1-\gamma) g_t^2 \]  

(5)

Where \( t \) represents a certain time, \( g_t \) is the gradient of the parameter at time \( t \), \( E[g_{t-1}^2] \) is the cumulative variable, represents the exponentially weighted average of the square of the gradient, and \( \gamma \) is the attenuation index.

The update process of the RMSProp algorithm[25]:

\[ \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g_t^2]}+\epsilon} g_t \]  

(6)

Where \( \theta \) is the parameter value, \( \eta \) is the learning rate, and \( \epsilon \) is a constant added to maintain numerical stability, such as \( 10^{-6} \).

C. CUDA Programming Framework and GPU Hardware

The processing speed of modern stand-alone processors is getting slower and slower, especially when processing big data, the processing speed is slowly reaching the limit. GPU is a commonly used device in modern PCs. It has the advantages of multiple core processors and fast thread switching speed and low cost. Coupled with the CUDA (Compute Unified Device Architecture) programming library...
developed by NVIDIA, GPU becomes the first choice for high-performance concurrent programming.

GPU consists of main hardware components such as Streaming Multiprocessor (SM) and graphics card memory. Multiple SMs form a stream processor cluster, and each SM is composed of multiple scalar processors SP (Scalar Processor) and SPU (Special-Purpose Unit)[26]. The components of the GPU are shown in Fig.1.

CUDA is a software and hardware system that uses GPU as a data parallel computing device, without the aid of graphics API. In the CUDA programming model, the CPU is used as the host for computing and storage resources; the GPU is used as the device for computing and storage resources; the parallel computing function running on the GPU is called the kernel function; the thread is the smallest unit of GPU processing, usually a GPU. Many threads can be composed of multiple threads; the thread grid is composed of multiple thread blocks[27]. The relationship between Thread, Block, and Grid is shown in Fig.2.

### III. IFABG PARALLEL ALGORITHM BASED ON CUDA

#### A. IMPROVED FUNKSVD ALGORITHM

We use RMSProp algorithm training to solve the optimal solution of the two low-rank matrices of the FunkSVD algorithm, which not only solves the problem of sparse data in large data sets, but also alleviates the iterative oscillation problem of the gradient descent method, thereby improving the accuracy of the algorithm.

First, we decompose the original score matrix \( M \) according to the traditional FunkSVD algorithm:

\[
M = U^T I, \quad M \in \mathbb{R}^{r \times l}, \quad U \in \mathbb{R}^{r \times k}, \quad I \in \mathbb{R}^{k \times l}, \quad k = \text{min}(r, l)
\]  

(7)

Where \( M \) is the original scoring matrix; \( U \) is the user's implicit feature matrix; \( I \) is the hidden feature matrix of the item; \( r \) indicates how many users are in \( M \); \( l \) indicates how many items are in \( M \); \( k \) represents the number of hidden features, and the value is much smaller than \( r \) and \( l \).

Then, we define the loss function according to (1)-(3):

\[
L = \min_{u, i} \sum_{(u,i) \in T} (m_{ui} - u_i^T i_j)^2 + \lambda(||u||^2 + ||i||^2)
\]  

(8)

Where \( T \) is the data set.

Next, we need to obtain the partial derivatives of \( u_u \) and \( i_i \):

\[
\frac{\partial L}{\partial u_u} = \frac{\partial}{\partial u_u} \left( \sum_{a,i \in T} \left( m_{ui} - u_u^T i_i \right)^2 + \lambda \left( \sum_u u_u^2 + \sum_i i_i^2 \right) \right) = 2 \sum_{a,i \in T} \left( m_{ui} - u_u^T i_i \right) (-i_i) + 2 \lambda u_u
\]  

(9)

\[
\frac{\partial L}{\partial i_i} = \frac{\partial}{\partial i_i} \left( \sum_{a,i \in T} \left( m_{ui} - u_u^T i_i \right)^2 + \lambda \left( \sum_u u_u^2 + \sum_i i_i^2 \right) \right) = 2 \sum_{a,i \in T} \left( m_{ui} - u_u^T i_i \right) (-u_u) + 2 \lambda i_i
\]  

(10)

Subsequently, we respectively substitute the partial derivatives of \( u_u \) and \( i_i \) into (5) to obtain the cumulative variable at a certain time \( t \):

\[
E_t^\gamma \frac{\partial L}{\partial u_u} \bigg|_{t+1} = \gamma E_t^\gamma \frac{\partial L}{\partial u_u} \bigg|_{t} + (1 - \gamma) (\frac{\partial L}{\partial u_u})_t^2
\]  

(11)

\[
E_t^\gamma \frac{\partial L}{\partial i_i} \bigg|_{t+1} = \gamma E_t^\gamma \frac{\partial L}{\partial i_i} \bigg|_{t} + (1 - \gamma) (\frac{\partial L}{\partial i_i})_t^2
\]  

(12)
Finally, we substitute the state variables into (6) to obtain the update rules of \( u_i \) and \( i_l \):

\[
(u_i)_{t+1} = (u_i)_t - \eta \sum_{a \in D} \left( \frac{\partial L}{\partial u_a} \right)_t,
\]

\[
= (u_i)_t - \eta \left( \sum_{a \in D} (m_a - u_a^T i_l)(-i_l) + \lambda u_a \right)_t + \epsilon
\]

\[
= (u_i)_t - \frac{\eta}{V} \left( \frac{\partial L}{\partial u_i} \right)_{t-1} + (1 - \gamma) \left( \sum_{a \in D} (m_a - u_a^T i_l)(-i_l) + \lambda u_a \right)_t + \epsilon
\]

\[
(i_l)_{t+1} = (i_l)_t - \eta \sum_{a \in D} \left( \frac{\partial L}{\partial i_l} \right)_t,
\]

\[
= (i_l)_t - \frac{\eta}{N} \left( \frac{\partial L}{\partial i_l} \right)_{t-1} + (1 - \gamma) \left( \sum_{a \in D} (m_a - u_a^T i_l)(-i_l) + \lambda i_l \right)_t + \epsilon
\]

(11)

The specific steps to improve the FunkSVD algorithm are as follows:

(a) Prepare scoring matrix \( M = (m_{ui})_{m \times l} = [m_1, m_2, ..., m_l] \), where \( m_l \) is the user's rating of the item.

(b) Decompose \( M \) into \( U \) and \( I \), where \( U \) is the user feature matrix and \( I \) is the item feature matrix.

(c) According to (11), iteratively update \( U \) and \( I \) to solve the optimal value. Until the stop condition is met, the optimal value is reached. The stopping condition is \( L < \beta \), where \( \beta \) is a minimum value.

(d) Predict the missing score by calculating the inner product of the updated user feature vector and the item feature vector.

The pseudo code of the improved FunkSVD algorithm is shown below.

**Algorithm 1**

**Input:** Rating matrix \( M \), hidden factor \( k \), regularization coefficient \( \lambda \), attenuation index \( \gamma \), learning rate \( \eta \), constant \( \epsilon \), minimum value \( \beta \).

**Output:** The updated scoring matrix \( N \).

1. Scan the data set, get the number of users \( l \), the number of items \( r \), and construct the user item score matrix \( M \).
2. Construct user feature matrix \( U \) and item feature matrix \( I \), the sizes are \( k \times l \) and \( k \times r \), and initial values are assigned.
3. Define the loss function \( L \).
4. Construct user deviation vector \( u \) and item deviation vector \( i_l \), with the size of \( 1 \times l \) and \( 1 \times r \) respectively, and assign initial values.
5. Construct the prediction score matrix \( N \), the size is \( r \times l \).
6. While True
7. If True
8. Construct \( L \) according to (8)
9. Else
10. Break;
11. For all \((u, i_l)\)
12. update the gradient according to (9)
13. according to (10)-(11), update \( u, i_l\)
14. End for
15. The output optimal \( u_i \) and \( i_l \) are stored in the user feature matrix \( U \) and the item feature matrix \( I \), and the Cartesian product of \( U \) and \( I \) is calculated to obtain an approximate score matrix \( N \) to predict the missing score between the user and the item.
16. END

**B. IFABG PARALLEL ALGORITHM**

CPU and GPU each have their advantages. CPU is suitable for processing logically complex operations, while GPU is suitable for processing logically simple and computationally intensive operations. The IFABG algorithm performs data set input and data format conversion on the CPU (host), divides the data set into parallel groups, initializes parameters such as learning rate, initializes the feature matrix, calls GPU kernel functions, iteratively controls. On the GPU side (device), the parallel iterative update process is mainly carried out, and the prediction scoring matrix is constructed in parallel. The work distribution of the IFABG algorithm under the framework of the CPU-GPU collaboration is shown in Fig.3.

From a programmer's point of view, the CUDA programming model is a collection of threads that can run in parallel. The CUDA program consists of a host program running on the host CPU and one or more parallel kernel functions executed on the GPU. Each kernel function is executed by multiple blocks, and each block contains multiple threads. In this paper, two kernel functions are designed for IFABG. The first kernel function is used to calculate the iterative update process of parameters in parallel, and the other kernel function is used to calculate the dot product between the user's special diagnosis vector and the item feature vector. It is stored in the prediction score matrix so that the main function can be called to predict the score. Before the GPU starts to calculate, the input data (original scoring matrix) needs to be transferred from the host memory to the GPU memory to become the global memory. Therefore, in the GPU calculation process, there is no need to frequently read data from the memory.
1) KERNEL FUNCTION 1
In contrast to the serial CPU implementation, multiple user feature vectors and item feature vectors can be updated in parallel in the GPU. First, the user feature matrix \(U\) and the item feature matrix \(I\) are divided into \(u\) threads (\(u\) column user feature vector) and \(i\) threads (\(i\) column item feature vector). In the GPU, \(u\) threads can be updated in parallel, and \(i\) threads can be updated in parallel until all elements are processed. The number of \(u\) and \(i\) depends on the value of the implicit factor \(k\) of the decomposition feature. Therefore, when the value of \(k\) is an integer multiple of the number of threads in the thread warp, the GPU memory access capability can be fully utilized. The process of updating parameters in CPU and GPU is shown in Fig.4.

2) KERNEL FUNCTION 2
Once kernel function 1 is executed on all grids, there are updated values of all parameters of user feature matrix \(U\) and item feature matrix \(I\), using the parameter update values, kernel function 2 performs the Cartesian product operation of matrix \(U\) and matrix \(I\) in parallel. This restores the prediction score matrix, followed by prediction. Performing a Cartesian product operation requires multiplying each element of each column of \(U\) with the corresponding column of \(I\), which is very time-consuming for serial, but parallel operations can be performed on different column elements in GPU, which greatly reduces the computation time. The process of calculating the inner product in the CPU and GPU is shown in Fig.5.

IFABG Algorithm
Input: Rating matrix \(M\), hidden factor \(k\), regularization coefficient \(\lambda\), attenuation index \(\gamma\), learning rate \(\eta\), constant \(\varepsilon\), minimum value \(\beta\).
Output: The updated scoring matrix \(N\).
1. Scan the data set, get the number of users \(l\), the number of items \(r\), and construct the user item score matrix \(M\).
2. Construct user feature matrix \(U\) and item feature matrix \(I\), the sizes are \(k \times l\) and \(k \times r\), and initial values are assigned.
3. Define the loss function \(L\).
4. Construct user deviation vector \(u\) and item deviation vector \(i\), with the size of \(1 \times l\) and \(1 \times r\) respectively, and assign initial values.
5. Construct the prediction score matrix \(N\), the size is \(r \times l\).
6. while not reach the stop condition do
7. For all \((u_0, i)\)
8. assign both \(u_0\) and \(i\) to \(k\) threads in the GPU for \(k\) GPU threads and each data point
9. update the gradient according to (9)
10. according to (10)-(11), update \(u_0\) \(i\)
11. end for
12. Predict score for every \(k\) GPU threads
13. Calculate the product of each element in each column of the user feature matrix \(U\) and each column of the item feature matrix \(I\) parallely
14. The product result is stored in the prediction score matrix \(N\)
16. END

IV. ANALYZE EXPERIMENTAL RESULTS
This paper proposes an IFABG algorithm, which realizes the improvement of the traditional FunkSVD algorithm, and uses GPU to realize the parallelization of the improved algorithm. The experimental environment uses 64-bit Ubuntu16.04, the C++ program compiler is gcc4.6.3, the programming language is C++, and the CUDA SDK version is 6.0. The hardware configuration is CPU dual-core 3.2GHz, memory 4GB, GPU memory 6GB.

A. EXPERIMENTAL DATA
The experiment uses MovieLens, from which data sets of different sizes are selected for experimentation. The score range of each data set is 1 to 5. The higher the score, the more satisfactory, as shown in Table I.

B. EVALUATION INDEX
1) ACCURACY
To measure the accuracy of prediction scores between IFABG and different algorithms, this paper uses Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), two evaluation indicators to measure.

MAE measures the accuracy of the prediction based on the mean deviation between the predicted score and the real user score. The smaller the value, the smaller the gap between the predicted score and the real user score. The smaller the value, the smaller the gap between the predicted score and the real user score.

FIGURE 4. The process of updating the parameters of the CPU and GPU.

FIGURE 5. Process of calculating inner product in CPU and GPU.
predicted value and the true value, and the higher the recommendation accuracy. The formula is as follows:

\[ \text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |r_{ui'} - r_{ui}| \]  

(12)

Where \( r_{ui'} \) represents the predicted score of user \( u \) on item \( i \), and \( r_{ui} \) represents the actual score of user \( u \) on item \( i \).

RMSE measures the accuracy of prediction based on the root mean square error between the predicted score and the user's actual score. The smaller the RMSE, the higher the accuracy of the recommendation and the higher the quality of the recommendation algorithm. The formula is as follows:

\[ \text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (s_{ui'} - s_{ui})^2} \]  

(13)

Where \( s_{ui'} \) represents the predicted score of user \( u \) on item \( i \), and \( s_{ui} \) represents the actual score of user \( u \) on item \( i \).

2) EXECUTION TIME

This paper evaluates the effectiveness of different algorithms by comparing the running time. Running time refers to the time it takes for an algorithm to execute, which is used to evaluate the running speed of the algorithm. The shorter the running time, the higher the effectiveness of the algorithm.

3) SPEEDUP RATIO

To verify the acceleration performance of the IFABG algorithm, this paper uses the acceleration ratio to measure the performance and effect of the algorithm. Speedup refers to the ratio of time consumed by the same task in a single-processor system and a parallel-processor system. The formula is as follows:

\[ S = \frac{T_{\text{CPU}}}{T_{\text{GPU}}} \]  

(14)

Where \( T_{\text{CPU}} \) refers to the running time of the algorithm in the CPU, and \( T_{\text{GPU}} \) refers to the running time of the algorithm in the GPU.

C. EXPERIMENTAL ANALYSIS

The setting of parameters plays an important role in the IFABG algorithm. Eigenvalue \( (k) \) is usually 10-100[28]. The regularization parameter \( (\lambda) \) is set to 0.2[24]. The attenuation rate \( (\gamma) \) is set to 0.9[29], and the learning rate \( (\eta) \) needs to be adjusted[30].

1) PARAMETER ADJUSTMENT

For the optimization algorithm in deep learning, the value of the learning rate \( \eta \) is very important. According to the experience of tuning, the learning rate of RMSProp algorithm is often less than the learning rate of gradient descent method. In order to select a more appropriate learning rate, we change the number of iterations under the same data set to verify the accuracy of IFABG under the learning rate of 0.002[30] and 0.01[31].

Fig.6 shows the comparison of the accuracy of the IFABG algorithm under different learning rates. We can see from figure that in the case of the same data set and different iteration times, the accuracy of the improved algorithm is obviously unstable when the learning rate is 0.01. However, when the learning rate is 0.002, the accuracy of the algorithm is higher, and gradually stabilizes as the number of iterations increases. Therefore, we set the learning rate to 0.002.

2) ACCURACY COMPARISON

To verify that the IFABG algorithm can improve the running speed while ensuring the accuracy, we compared the IFABG algorithm with the traditional FunkSVD algorithm, the algorithm in [32], and the algorithm in [33]. We conduct experiments under different datasets and different implicit eigenvalues, respectively. The algorithm in [32] is a GPU-based Singular Value Decomposition Feature Basis Function (SVD-CBFM) algorithm. GPU is used to parallelize the algorithm and improve the calculation speed of the algorithm. [33] used GPU to parallelize the SGD algorithm to solve the matrix factorization problem. Fig.7 shows the comparison of the accuracy of different algorithms under different data sets, and Fig.8 shows the comparison of the accuracy of different algorithms under different implicit eigenvalues.
We can see from Fig. 7 that under the same eigenvalue, with the increase of the data amount, the accuracy of each algorithm shows a decreasing trend. But it can be clearly seen that the accuracy of the IFABG algorithm is significantly higher than the traditional FunkSVD algorithm, and not lower than the other two algorithms. We can see from Fig. 8 that with the increase of implicit eigenvalues, the accuracy of the IFABG algorithm does not drop significantly and is still higher than other algorithms. However, the accuracy of SVD-CBMF shows a downward trend because the algorithm does not add a regularization term to prevent over-coupling. Although the accuracy of the CU2REC algorithm does not show a significant downward trend, the accuracy is not as high as that of the IFABG algorithm. To sum up, it can be well illustrated that the IFABG algorithm has good prediction accuracy when dealing with big data.

3) RUN TIME COMPARISON

To verify the effectiveness of the IFABG algorithm, we compared the running time of each algorithm under the same conditions. Under the same conditions, the shorter the running time of the algorithm, the higher the effectiveness of the algorithm. The comparison results are shown in Fig. 9.

We observe that as the dataset gets larger, the running time of both the parallel algorithm and the serial algorithm increases, but the running time of the parallel algorithm is significantly less than that of the serial algorithm. This is because the IFABG algorithm not only reduces the time of iterative computation, but also reduces the time of prediction score computation. And the running time of the IFABG algorithm is less than the other two parallel algorithms. The IFABG algorithm uses the GPU to shorten the running time and improve the effectiveness of the algorithm. To sum up, the IFABG algorithm has a good effect in processing big data and is better than other algorithms.

4) SPEEDUP COMPARISON

In order to measure the parallel ability of the IFABG algorithm, experiments were conducted on three data sets of different sizes, and the feature value k was changed to 20-100. The higher the speedup, the stronger the parallelism of the algorithm. The speedup result is shown in Fig. 10.

We can see from the Fig. 10 that the speedup ratio gradually increases as the number of data increases, and the speedup ratio for the ml-1m data set is much greater than the other two data sets. This is because the hidden feature value will affect the degree of decomposition. If the hidden feature value is k, then the size of the user feature vector and the item feature vector are both k. The traditional algorithm needs to be updated k times, while IFABG only needs one thread to update the vector. In this way, IFABG only needs to be operated once. Similarly, when calculating the inner product between the feature vectors, the traditional algorithm needs to be performed multiple times while the IFABG algorithm only needs to be operated once. Therefore, with the gradual increase of the hidden feature value, the speed-up ratio also gradually increases, which further illustrates the advantages of the IFABG algorithm in processing big data.

In summary, combined with the experimental result shows that the IFABG algorithm is superior to traditional algorithms in terms of accuracy and running time, and has better parallel computing capabilities, can handle large amounts of data well, and has good effectiveness.

V. CONCLUSION

To improve the accuracy of the traditional FunkSVD algorithm and the effectiveness of the algorithm, we propose a GPU-based parallel recommendation algorithm. After experimental comparison, we found that the accuracy of the IFABG algorithm is about 30% higher than that of the traditional FunkSVD algorithm, and it can better handle large amounts of sparse data. In addition, the IFABG algorithm has good parallel capabilities, with a speedup of 19.27.

Future work is to further improve the scalability of the IFABG algorithm, apply the algorithm to practical applications, and improve the practicability of the algorithm.

FIGURE 8. Comparison of the convergence speed and accuracy of the three algorithms under different implicit eigenvalues.

FIGURE 9. Comparison of running time of different algorithms.

FIGURE 10. Speedup of IFABG algorithm.
REFERENCES

[1] J. Yan, Q. T. Zeng and F. Q. Zhang, “Summary of Recommendation Algorithm Research,” JPCS, vol. 1754, no. 1, pp. 012224, 2021.

[2] G. Liu, X. Wu, “Using Collaborative Filtering Algorithms Combined with Doc2Vec for Movie Recommendation,” in Proc. IJITNEC, Sichuan, China, 2019, pp. 1461-1464.

[3] D. Chae, J. A. Shin and S. Kim, “Collaborative Adversarial Autoencoders: An Effective Collaborative Filtering Model Under the GAN Framework,” IEEE Access, vol. 7, pp. 37650-37663, 2019, 10.1109/ACCESS.2019.2905876.

[4] E. Y. Yang et al., “FCMF: Federated collective matrix factorization for heterogeneous collaborative filtering,” Knowl Based Syst, vol. 220, pp. 106946, May, 2021, 10.1016/j.knosys.2021.106946.

[5] J. L. Sui, J. Yin, “A Biased Proportional-Integral-Derivative-Incorporated Latent Factor Analysis Model,” Appl. Sci., vol. 427, no. 4, pp. 29-39, Feb, 2020, 10.1016/j.neucom.2020.11.029.

[6] S. J. Wang, G. L. Sun and Y. Y. Li, “SVD++ Recommendation Algorithm Based on Backtracking,” Inf Sci, vol. 11, no. 7, pp. 369, 2020, 10.3390/infol10070369.

[7] B. Walek and V. Fojtik, “A hybrid recommender system for recommending relevant movies using an expert system,” Expert Syst. Appl., vol. 158, pp. 113452, 2020.

[8] D. Yao and X. Deng, “Teaching Teacher Recommendation Method Based on Fuzzy Clustering and Latent Factor Model,” IEEE Access, vol. 18, no. 2, pp. 100054, June, 2020, 10.1109/ACCESS.2020.3039011.

[9] Z. Y. Wang, Y. Liu and S. Chiu, “An efficient parallel collaborative filtering algorithm on multi-GPU platform,” J Supercomput, vol. 72, no. 6, pp. 2080-2094, 2016.

[10] L. Y. Kang, B. Tang, Y. M. Xia and L. Zhang, “Parallel collaborative filtering recommendation algorithm based on GPU acceleration and non-negative matrix factorization,” Computer Science, vol. 46, no. 8, pp. 106-110, 2019.

[11] J. Jin et al., “GPUSGD: A GPU - accelerated stochastic gradient descent algorithm for matrix factorization,” Concurr Comp-Pract Ex, vol. 28, no. 14, pp. 3844-3865, 2016.

[12] J. WU, B. LV and W. CUI, “Recommendation Model based on Mobile Commerce in Cloud Computing,” in Proc. ICIPDS, Hong Kong, China, 2020, pp. 28–31.

[13] L. B. Huai and Y. R. Cui, “An academic paper recommendation algorithm based on distributed graph computing,” Application Research of Computers, vol. 36, pp. 1629-1632, 2019.

[14] B. Kumar, “A novel latent factor model for recommender system,” IJISTEM, vol. 13, no. 3, pp. 497-514, 2016.

[15] F. Shang et al., “VR-SGD: A Simple Stochastic Variance Reduction Method for Machine Learning,” IEEE T Knowl Data en, vol. 32, no. 1, pp. 188-202, 2020.

[16] Shamis N. Abd, Mohammad Alsajri, and Hind Raad Ibraheem, “Rao-SVM Machine Learning Algorithm for Intrusion Detection System,” ijcsm, vol. 1, no. 1, pp. 23-27, Jan. 2020.

[17] M. Aaisha and K. Neeraj, “PROTECTOR: An optimized deep learning-based framework for image spam detection and prevention,” Future Gener Comput Syst, vol. 125, pp. 41-58, 2021.

[18] M. Aaisha and K. Neeraj, “PROTECTOR: An optimized deep learning-based framework for image spam detection and prevention,” Future Gener Comput Syst, vol. 125, pp. 41-58, 2021.

[19] D. Hamidreza and Z. L. Andreas, “A hybrid MGA-MSGD ANN training approach for approximate solution of linear elliptic PDEs,” Math Comput Simul, vol. 190, pp. 398-417, 2021.

[20] Zainab rustum, “Investigating the use of Adaptive Neuro-Fuzzy Inference System in Software Development Effort Estimation”, ijcsm, vol. 2, no. 2, pp. 18-24, Jul. 2021.

[21] R. Liu, Y. Liu and J. G. Gu, “AdaNet improvement based on adaptive learning rate optimization,” Journal of Computer Applications, pp. 1-7, 2020.

[22] J. C. Duchi, E. Hazan and Y. Singer, “Adaptive Subgradient Methods Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,” J Mach Learn Res, vol. 12, pp. 2121-2159, 2011, 10.1109/TNN.2011.2146788.

[23] R. V. Kumar Reddy, B. Srinivasa Rao and K. P. Raju, “Handwritten Hindi Digits Recognition Using Convolutional Neural Network with RMSprop Optimization,” in Proc. ICICCS, Madurai, India, 2018, pp. 45-51.

[24] X. Wu, H. Xu, X. Wei, et al., “Damage Identification of Low Emissivity Coating Based on Convolutional Neural Network,” IEEE Access, vol. 8, pp. 156792-156800, 2020, 10.1109/ACCESS.2020.3019484.

[25] Y. Sun, w. Zhang, H. Gu, C. Liu, S. Hong and W. Xu, et al., “Convolutional Neural Network Based Models for Improving Super-Resolution Imaging,” IEEE Access, vol. 7, pp. 43042–43051, 2019, 10.1109/ACCESS.2019.290850.

[26] R. T. Possignolo, E. Ebrahimi, E. K. Ardestani, A. Sankaranarayanan, J. L. Briz and J. Renau, “GPU NCT Process Variation Compensation with Voltage Stacking,” IEEE Trans Very Large Scale Integr VLSI Syst, vol. 26, no. 9, pp. 1713-1726, Sept, 2018, 10.1109/TVLSI.2018.2831665.

[27] K. He, S. X. Tan, H. Wang and G. Shi, “GPU–Accelerated Parallel Sparse LU Factorization Method for Fast Circuit Analysis,” IEEE Trans Very Large Scale Integr VLSI Syst, vol. 24, no. 3, pp. 1140-1150, March 2016, 10.1109/TVLSI.2015.2421287.

[28] T. Xie, S. Li and B. Sun, “Hyper spectral Images Denoising via Nonconvex Regularized Low-Rank and Sparse Matrix Decomposition,” IEEE Int. Conf. Signal Image Process. Appl., vol. 29, pp. 44-56, 2020, 10.1109/TIP.2019.2926736.

[29] M. A. Labbaf Khaniki, M. Behzad Hadi and M. Manthouri, “Feedback Error Learning Controller based on RMSprop and Salt Warp Algorithm for Automatic Voltage Regulator System,” in Proc. ICCKE, Mashhad, Iran, 2020, pp. 425-430.

[30] F. Zou, L. Shen, Z. Jie, et al., “A Sufficient Condition for Convergences of Adam and RMSProp,” in Proc. CVPR, Seattle, Washington, 2019, pp. 11119-11127.

[31] G. Hinton, N. Srivastava, and K. Swersky, “Neural networks for machine learning: Lecture 6A overview of mini-batch gradient descent course,” Video Lectures Coursera, Mountain View, CA, USA, 2012.

[32] J. Zhang and Y. Sun, “Parallel SVD-CBFM based on GPU acceleration,” in Proc. CSQWRWC, Shanxi, China, 2019, pp. 1-3.

[33] D. Kilicigil, N. Greenquist, M. Zahran and A. Bari, “GPUAccelerated Matrix Factorization for Recommender Systems,” in Proc. ICBDA, Guangzhou, China, 2021, pp. 272-281.
YUE Xiaochen was born in 1997 in Weihai, Shandong, China. From 2015 to 2019, she studied at Qingdao Agricultural University and obtained a bachelor's degree. She has been studying at Yantai University since 2019 and is expected to receive a master's degree in 2022. Her main research direction is data mining and big data.

She participated in the Shandong Science and Technology Innovation Competition in October 2020. She is mainly responsible for collating and analyzing data and producing related project description videos. She won a merit scholarship in 2016, a graduate student scholarship in 2019, and an academic scholarship in 2021.

LIU Qicheng born in 1970. PhD, professor. His research interests include parallel algorithm, big data, multi-agent systems, data mining, etc.