This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212

Parallel Fractional Stochastic Gradient Descent with Adaptive Learning for Recommender Systems

Fatemeh Elahi [1], Mahmood Fazlali [1], Hadi Tabatabaee Malazi [2], Mehdi Elahi [3]
[1] Department of Computer and Data Sciences, Faculty of Mathematical Sciences, Shahid Beheshti University, Tehran, Iran.
[2] Department of Computer Science, Maynooth University, Maynooth, Ireland.
[3] Department of Information Science and Media Studies, University of Bergen, Bergen, Norway.

Abstract—The structural change toward the digital transformation of online sales elevates the importance of parallel processing techniques in recommender systems, particularly in the pandemic and post-pandemic era. Matrix factorization (MF) is a popular and scalable approach in collaborative filtering (CF) to predict user preferences in recommender systems. Researchers apply Stochastic Gradient Descent (SGD) as one of the most famous optimization techniques for MF. Paralleling SGD methods help address big data challenges due to the wide range of products and the sparsity in user ratings. However, these methods' convergence rate and accuracy are affected by the dependency between the user and item latent factors, specifically in large-scale problems. Besides, the performance is sensitive to the applied learning rates. This paper proposes a new parallel method to remove dependencies and boost speed-up by using fractional calculus to improve accuracy and convergence rate. We also apply adaptive learning rates to enhance the performance of our proposed method. The proposed method is based on Compute Unified Device Architecture (CUDA) platform. We evaluate the performance of our proposed method using real-world data and compare the results with the close baselines. The results show that our method can obtain high accuracy and convergence rate in addition to high parallelism.

Index Terms—Parallel Matrix Factorization, Recommender Systems, Collaborative Filtering.

1 Introduction

The recent pandemic has accelerated the structural changes toward online stores. While at least 60% of people plan to cut their consumer electronics expenditures, the reports show a 16% surge in web traffic to online tech retailers [1]. The digital transformation puts an extra burden on digital sale platforms to apply efficient recommender systems to cope with big data challenges due to the growing number of users, their diverse consumption culture, and a wide range of offered products. Addressing these challenges enhances recommender systems to reduce users' decision-making time [2] while assisting them in facing information overload [3], [4]. Moreover, ongoing research tries to address the traditional challenges of recommender systems, including, cold start [5], [6], sparsity [7], [8], scalability [9], [10], privacy [11], [12].

Collaborative filtering is a popular technique in recommender systems. It is a simple and domain-independent technique based on users' historical behaviors. The main idea relies on the hypothesis that users who have shared similar ratings in the past are likely to have similar preferences for new items. The collaborative models are divided into two main classes: Model- and Memory-based. The primary purpose of the model-based is to design a model for approximating user preferences for unknown data. Memory-based methods use statistical approaches to compute the similarity of items and recommend them to target users based on the preferences of other like-minded ones [13].

The collaborative filtering recommends items without requiring extensive data collection. However, data sparsity in the user-item matrix can be an issue, especially in systems with a significant number of users/items. Data sparsity refers to the phenomenon in which each user rates a small number of items. The limited number of ratings introduces challenges for accurate predictions [14] as well as the convergence of the learning procedure [15]. The challenges are more critical in systems with high data dimensions. Matrix factorization (MF) [16], [17] is a linear algebraic method to solve the data sparsity problem in collaborative filtering recommender systems. A dimensional reduction method maps both users and items into the same latent feature space. It decomposes a user-item interaction matrix R into the product of two lower-dimensional matrices, i.e., user feature matrix P and item feature matrix Q, to estimate the missing values by relying on defined entries. The quality of these estimations is also sensitive to parameters such as the number of latent factors, regularization coefficients, epoch size, and learning rate.

Gradient-based learning methods, e.g., Stochastic Gradient Descent (SGD), are one of the approaches used for matrix factorization [13]. These methods' simplicity and intuitive structure introduce them as a classical approach in optimization theory and other similar fields [18]. The standard gradient descent method has potential drawbacks. First, it shows oscillating behavior. Second, it converges

Corresponding author: Mahmood Fazlali (fazlali@sbu.ac.ir).

Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212

slowly around the optimal point; besides these drawbacks, SGD-based methods require extensive mathematical operations, which can be time-consuming, particularly in large-scale applications. These limitations reduce the applicability of these methods.

Researchers introduce several parallel implementations to accelerate matrix factorization and achieve scalability in high-dimensional data through shared memory and distributed systems. One promising approach is using a Graphical Processing Unit (GPU). GPU is a general-purpose parallel computing device supporting Single Instruction Multiple Data (SIMD) paralleling. GPU can provide considerable computational power to process data-intensive applications such as matrix calculations using Compute Unified Device Architecture; it can also accelerate calculations, particularly in floating-point numbers [19], [20]. A parallel implementation of matrix factorization confronts challenges, including processing high dimensional intermediate matrices, numerous matrix manipulations, a significant number of memory accesses, and dependencies on a user-item pair through the SGD updating procedure. Different approaches are proposed to solve the overwriting problem of SGD-MF paralleling. *Li* et al. [14] proposed a multi-stream Stochastic Gradient Descent (MSGD) approach. Some researchers [20] proposed an approach based on incremental learning for online processing. However, these attempts mainly focused on improving one aspect such as accuracy, convergence speed, or speed-up separately. We consider the importance of steady-state proficiency and convergence rate in addition to speed-up. The strategy for paralleling SGD-MF can affect the accuracy, so a solution is needed to compensate for possible drawbacks and improve the model's accuracy.

This paper focuses on these research questions:

- *RQ1*: How can we eliminate the dependency of users and items for paralleling SGD matrix factorization (SGD-MF)?
- *RQ2*: How can we select a suitable learning rate that ensures to fall within the stability region?
- *RQ3*: How can we improve the convergence speed of the standard SGD method for sparse and noisy real-world data?

We propose Fractional Adaptive Multi-stream Stochastic Gradient Descent (FAMSGD) method. The devised method uses fine-grained parallel programming and exploits a CUDA paralleling strategy [14]. We reduce the impediments by dividing the main procedure into smaller ones. The coarse sub-tasks are assigned to independent thread blocks. Each block divides the devoted sub-task into some fine parts and maps it into individual threads. In other words, the optimization problem is divided into sub-problems assigned to these thread blocks and is solved cooperatively. Besides, the proposed method updates the user and item feature matrices alternatively. The capabilities of fractional derivatives in [21] motivate us to apply them in matrix factorization-based collaborative filtering. For the stability of the model, we are inspired by the work presented in [22]. We select an adaptive learning rate and tune it according to increase the number of iterations (the training stage). This paper's main contributions are as follows:

- *Parallel matrix factorization*: We devise a fine-grained parallel matrix factorization based on fractional calculus for CUDA-supported GPUs.
- *Use an adaptive learning rate to avoid local minimum*: In SGD, the parameters are estimated for every observation, which provides a random property to avoid local minimum and reach the optimum point. However, we reduce parameter estimation for each data sample independently and adaptively to obtain reasonable accuracy.
- *Increase convergence rate*: We take advantage of fractional calculus by applying fractional-order gradient descent to improve the accuracy and convergence speed of a matrix factorization-based recommender system.

The remainder of this paper is structured as follows: Section 2 surveys the related work in paralleling Stochastic Gradient-based Matrix Factorization and the accuracy/convergence of these methods. In Section 3, we introduce primitive concepts and formulate the problem definition. Section 4 describes our proposed method. The performance evaluation and the sensitivity analysis of the model to hyper-parameters are provided in Section 5. Finally, Section 6 concludes the paper.

2 Related work

This section reviews parallel matrix factorization, emphasizing SGD-based methods. It also surveys the methods that leverage fractional calculus.

2.1 Parallel methods in matrix factorization

Several matrix factorization methods have been proposed, focusing mainly on execution time, convergence rate, and accuracy, adding innovative ideas to classical algorithms. Some examples are Alternating Least Square [23], [24], Cyclic Coordinate Descent [25], Stochastic Gradient Descent [26].

SGD is a very prevalent method used to solve different optimization problems. *Kaleem* et al. [27] use SGD as an exemplar and study how the selection of synchronization mechanism impacts the end-to-end performance of graph algorithms with high complexity. SGD is applied in matrix factorization extensively [26], [28]. Different variations of standard SGD have been presented to improve accuracy and convergence. KMSGD [29] is a mini batch-based method that lacks the approach for selecting the optimal value for the learning rate. In [30], an extended SGD-based model is proposed for recommender systems. In [31], the over-parametrized standard SGD approach is presented, which is computationally more expensive than standard SGD.

Several methods have been proposed to parallel SGD and apply it to large-scale problems. However, parallel execution of SGD in matrix factorization causes the problem of losing updates. Researchers tried to address this problem. *Nassar* et al. [32] introduce GPU-MF-SGD to address scalability problems and enhance the speed-up relation. However, this method does not consider the uneven data distribution on extremely sparse matrices. It is not possible to develop this approach on a multi-GPU schema where GPUs need synchronization. A shared lock is used to take turns to GPUs in update processing and causes time-wasting to wait for the completion of updating parameters.

Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212

To avoid this problem, selecting isolate ratings is considered, which causes accuracy loss. The method presented in [33] applies parallel non-negative matrix factorization based on GPU in the Spark platform to take advantage of in-memory computation and GPU acceleration. The SGD method is highly sensitive to learning rate parameters, influencing the convergence speed. Adjusting these values is a challenging issue for state-of-the-art methods. Researchers develop different strategies to adjust the proper value for learning rate in matrix approximation-based collaborative filtering. *Li* et al. [22] designed a method to tune the learning rate automatically based on the noisiness level of ratings in the user-item rating matrix. *Loizou* et al. [34] propose an algorithm inspired by classical *Polyak* step-size (SPS) to adapt the learning rate. They use SPS in the subgradient algorithms to enable fast convergence.

2.2 Fractional calculus

The equation of the standard SGD and its variant counterparts are determined by calculating the first-order partial derivative of the objective function and a set of its input parameters. This procedure has a long convergence process.

The idea of using fractional gradient-based learning methods inspires researchers and shows advantages against other numerical methods in both accuracy and convergence [18], [35]. *Tan* et al. [36] generalize the Modified Least Mean Squares (MLMS) algorithm and applied fractional calculus. Using the weighted sum of an integer and fractional-order derivative is suitable for solving optimization problems. These results lead to generalizing the idea of applying Fractional Stochastic Gradient Descent (FSGD) to recommender systems. *Khan* et al. [21] employ FSGD to improve the matrix factorization algorithm for the recommender system. Their proposed method outperforms the standard one in convergence and accuracy. However, it suffers from selecting an appropriate learning rate parameter. Following their previous work, *Khan* et al. [37] designed a new computational method, called Normalized Fractional SGD (NFSGD) [38], to adaptively tune the value of the learning rate parameter and improve the stability and flexibility of the FSGD method.

Considering that each update phase in SGD-MF is relatively lightweight, numerous updates have to be performed to maximize the GPU utilization and boost the execution time. All the reviewed methods involve a series of iterative steps, and the large volume of data has a negative impact on the convergence time. Moreover, increasing the number of random samples to saturate GPU in each iteration causes the training time to take longer.

2.3 Baseline methods

This section describes the methods used as baseline comparisons. These paralleling matrix factorization methods aim to solve the overwriting problem. Researchers optimize matrix factorization from two perspectives. First is algorithms, in which researchers aim to increase the convergence speed and reduce the number of iterations to train the model. The second is systems, which aim to propose approaches to accelerate computations.

2.3.1 SG-based

CuMF_SGD [39] focuses on system streams and exploits data and model parallelism. The authors proposed two parallel processing schemes, i.e., Batch-Hogwild and Wavefront-update, which are equivalent in performance and accuracy. The Batch-Hogwild model utilizes a lock-free approach based on Hogwild [40]. CuMF_SGD uses half-precision to better utilize GPU resources. CuMF_SGD exploits the spatial data locality using the L1 cache [32], reducing the time to schedule blocks. It is noteworthy that shuffling ratings and partitioning data into batches in preprocessing phase is necessary.

CUMSGD [14] is a CUDA-based paralleling of SGD-MF. It considers multi-stream Stochastic Gradient Descent and copes with the problem of dependencies on the user-item pair by splitting the main objective function into multiple independent sub-objective functions [14]. This method introduces two scheduling strategies for thread blocks (decentralized and centralized). In the decentralized approach, entries in the same row of rating matrix R are selected by thread blocks. In contrast, the centralized approach is designed to divide rows and columns of a sparse matrix evenly.

LIBMF [41], [42] is a fast parallel SGD-MF method for shared memory systems. It focuses on reducing the cache miss rate and addressing the load balance of threads to accelerate matrix factorization. It splits the rating matrix R into independent blocks. At any moment, each thread selects a block to update that does not share any row or column with others.

2.3.2 Non-SG based

Parallel CCD++ [25] is a parallel algorithm for matrix factorization based on a variant of the cyclic coordinate descent concept. By considering the rows of P and Q matrices in the form $(\hat{P}_1, ..., \hat{P}_r)$ and $(\hat{Q}_1, ..., \hat{Q}_r)$, CCD++ updates $(\hat{P}_1, \hat{Q}_1)$ until $(\hat{P}_r, \hat{Q}_r)$ cyclically. Eq. 1 shows the objective function.

$$\sum_{(i,j)\in\Omega} (e_{i,j} + \hat{P}_i\hat{Q}_j - z_i f_j)^2 + \lambda_P \sum_{i=1}^{m} z_i^2 + \lambda_Q \sum_{j=1}^{n} f_j^2. \tag{1}$$

This method converts the main problem into independent sub-problems by reformulating the objective function into m and n sub-problems defined in Eqs. 2 and 3.

$$\min_{z_i} \sum_{j:(i,j)\in\Omega} (e_{i,j} + \hat{P}_i\hat{Q}_j - z_i\hat{Q}_j)^2 + \lambda_P z_i^2 \quad \forall i = 1, ..., m. \tag{2}$$

$$\min_{f_j} \sum_{i:(i,j)\in\Omega} (e_{i,j} + \hat{P}_i\hat{Q}_j - f_j\hat{P}_i)^2 + \lambda_Q f_j^2 \quad \forall j = 1, ..., n. \tag{3}$$

Similarly, by fixing z, solving the following n independent problems leads to finding f to update $\hat{Q}$.

However, our approach is fundamentally different from these collaborative filtering-based approaches as these methods consider only a subset of objectives such as speed-up, convergence, scalability, and accuracy independently and separately. Paralleling fractional calculus's contribution to solving the adaptive SGD-MF problem has not been explored in recommender systems. We are motivated by the effect of fractional calculus methods in increasing convergence and accelerating the standard SGD; adding its related derivative to the standard one increases calculations partly.

Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212

We explore a parallel approach to benefit from this idea on large-scale datasets and resolve the bottleneck of paralleling fractional SGD-MF. In this approach, we focus on accuracy, convergence speed, and scalability at the same time. We consider the sensitivity of the learning rate parameter to the noisiness of ratings in the user-items rating matrix and adjust an adaptive learning rate to design a model. To accelerate and increase the speed-up in computation, we employ a CUDA paralleling approach to utilize heterogeneous computing and large-scale matrix factorization in CF problems. The results indicate that our proposed approach can overcome the problems of previous methods.

3 Problem definition and primitive concepts

We consider $R \in \mathbb{R}^{m\times n}$ as the rating matrix used in a collaborative filtering recommender system where m and n are the numbers of users and items, respectively. Let $R(i,j)$ be the entry (i,j) of R that shows the recorded rating of user i to item j. We define vector Ω_i to represent the items rated by the user i and vector $\overline{\Omega_{\mathrm{j}}}$ show the users rated item j. Two rudimentary matrices, P and Q, are used in matrix factorization, where $P \in \mathbb{R}^{m\times r}$ and $Q \in \mathbb{R}^{n\times r}$ and r represent the number of features. We also define $\hat{R} = PQ^T$ to predict the rating matrix and define the unknown entries. The optimization problem is introduced based on minimizing the euclidean distance function to achieve an approximate solution. This function measures the precision intensity by using L_2 _norm regularization as follows:

$$\underset{P,Q}{\operatorname{argmin}}\, d(P,Q) = \sum_{(i,j)\in\Omega} \left(R(i,j) - \hat{R}(i,j)\right)^2 + \lambda_p \|P_i\|^2 + \lambda_q \|Q_j\|^2. \tag{4}$$

L_2 _norm regularization is used to control the smoothness of updating parameters. λ_p and λ_q are regularization parameters to avoid the overfitting problem, and $\|.\|$ is the euclidean norm. Ω contains the ordered pairs (indices) of the non-zero elements in the rating matrix. The problem (Eq. 4) involves two parameters of P and Q.

3.1 Stochastic gradient descent approach

The SGD-based approach avoids trapping in a local minimum by randomly selecting an entry (i,j). Once $R(i,j)$ is selected, the objective function defined in Eq. 4 is transformed to Eq. 5.

$$d(P_i,Q_j) = \left(R(i,j) - \hat{R}(i,j)\right)^2 + \lambda_p \|P_i\|^2 + \lambda_q \|Q_j\|^2. \tag{5}$$

On that basis, we determine the values of λ_p, λ_q, and the equation to update P_i is formulated in Eq. 6.

$$\begin{aligned} {P_i}^t &= {P_i}^{t-1} - \mu \frac{\partial d(P_i,Q_j)}{\partial P_i} \\ &= {P_i}^{t-1} + \alpha_{in}[(e(i,j){Q_j}^{t-1} - \lambda_p {P_i}^{t-1}]. \end{aligned} \tag{6}$$

The recursive equation for updating Q_j is calculated similarly (Eq. 7), where $\alpha_{in} = 2\mu$ is the primitive learning rate and $e(i,j) = R(i,j) - \hat{R}(i,j)$ is the error between the exact and estimated value of the (i,j) entry.

$${Q_j}^t = {Q_j}^{t-1} + \alpha_{in}[(e(i,j){P_i}^{t-1} - \lambda_q {Q_j}^{t-1}]. \tag{7}$$

3.2 Fractional stochastic gradient descent

Fractional Stochastic Gradient Descent (FSGD) is a method that covers a broad applicability scope in both nonlinear and linear systems. It can adapt the scheme variables of rigid systems more precisely because they are built on the complex mathematical grounds of fractional calculus. The recursive equations (Eq. 6 and Eq 7) are modified by applying FSGD. The equations to update feature vectors are shown in Eq. 8 and Eq. 9 .

$${P_i}^t = {P_i}^{t-1} + \alpha_{in}[(e(i,j){Q_j}^{t-1} - \lambda_p {P_i}^{t-1}] - \alpha_{fr} D^{fr}_{P_i}, \tag{8}$$

$${Q_j}^t = {Q_j}^{t-1} + \alpha_{in}[(e(i,j){P_i}^{t-1} - \lambda_q {Q_j}^{t-1}] - \alpha_{fr} D^{fr}_{Q_j}, \tag{9}$$

where $D^{fr}_{P_i}$ and $D^{fr}_{Q_j}$ are fractional derivative equations to P_i and Q_j, respectively. By computing $D^{fr}_{P_i}$, $D^{fr}_{Q_j}$, and replacing in Eq. 8 and Eq. 9, we can achieve Eqs. 10 and Eq. 11.

$$\begin{aligned} {P_i}^t \cong& {P_i}^{t-1} + \alpha_{in}[(e(i,j){Q_j}^{t-1} - \lambda_p {P_i}^{t-1}] - \\ &\alpha_{fr}[-\frac{2\Gamma(2)}{\Gamma(2-fr)} R(i,j){Q_j}^{t-1}({P_i}^{t-1})^{1-fr} + \\ &2\lambda_p {P_i}^{t-1} \frac{\Gamma(2)}{\Gamma(2-fr)}({P_i}^{t-1})^{1-fr}], \end{aligned} \tag{10}$$

$$\begin{aligned} {Q_j}^t \cong& {Q_j}^{t-1} + \alpha_{in}[(e(i,j){P_i}^{t-1} - \lambda_q {Q_j}^{t-1}] - \\ &\alpha_{fr}[-\frac{2\Gamma(2)}{\Gamma(2-fr)} R(i,j){P_i}^{t-1}({Q_j}^{t-1})^{1-fr} + \\ &2\lambda_q {Q_j}^{t-1} \frac{\Gamma(2)}{\Gamma(2-fr)}({Q_j}^{t-1})^{1-fr}]. \end{aligned} \tag{11}$$

We calculate $D^{fr}_{P_i} d(P_i,Q_j)$ and $D^{fr}_{Q_j} d(P_i,Q_j)$ based on Caputo fractional derivative definition. According to the definition, for a function in the form $(x-a)^m$, we have:

$${}^{c}_{a}D^{\alpha}_{x} f(x) = {}^{RL}_{a} D^{-(n-\alpha)}_{x} f^{(n)}(x) = \frac{\Gamma(1+m)}{\Gamma(1+m+v)}(x-a)^{m+v}. \tag{12}$$

The Caputo derivative for a constant function equals zero. Γ is gamma function and defined as $\Gamma(x) = (x-1)!$. Three specific parameters appear in the update equation (i.e., α_{in}, α_{fr} and fr), which are used to control the convergence of the adaptive algorithm.

In both discussed equations (Eq. 10 and Eq. 11), the main goal is to estimate each user's interest in items by reconstructing the user-item rating matrix. However, the dependency between user-item feature vectors is a prominent and clear point in update processing. Consider the following example. We randomly select three entries belonging to a specified rating set of the same row, such as $\{(i_1,j_0),(i_1,j_a),(i_1,j_b)\}$, where $\{j_0,j_a,j_b\} \in \Omega_{i_1}$. The procedure of updating $\hat{R}(i_1,j_0), \hat{R}(i_1,j_a), \hat{R}(i_1,j_b)$ is inherently sequential. To parallel the update procedures in the same iteration, we consider three different threads $\{T_0,T_1,T_2\}$; the related feature vectors are updated based on Eqs. 13, 14, and 15.

1) T_0 for $\hat{R}(i_1,j_0)$:

$$\begin{aligned} {P_{i_1}}^t &\cong {P_{i_1}}^{t-1} + \alpha_{in}[(e(i_1,j_0){Q_{j_0}}^{t-1} - \lambda_p {P_{i_1}}^{t-1}] - \alpha_{fr} D^{fr}_{P_{i_1}}, \\ {Q_{j_0}}^t &\cong {Q_{j_0}}^{t-1} + \alpha_{in}[(e(i_1,j_0){P_{i_1}}^{t-1} - \lambda_q {Q_{j_0}}^{t-1}] - \alpha_{fr} D^{fr}_{Q_{j_0}}. \end{aligned} \tag{13}$$

Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212

2) T_1 for $\hat{R}(i_1, j_a)$:

$$\begin{aligned} {P_{i_1}}^t \cong & {P_{i_1}}^{t-1} + \alpha_{in}[(e(i_1, j_a){Q_{j_a}}^{t-1} - \lambda_p {P_{i_1}}^{t-1}] - \alpha_{fr} D^{fr}_{P_{i_1}}, \\ {Q_{j_a}}^t \cong & {Q_{j_a}}^{t-1} + \alpha_{in}[(e(i_1, j_a){P_{i_1}}^{t-1} - \lambda_q {Q_{j_a}}^{t-1}] - \alpha_{fr} D^{fr}_{Q_{j_a}}. \end{aligned} \tag{14}$$

3) T_2 for $\hat{R}(i_1, j_b)$:

$$\begin{aligned} {P_{i_1}}^t \cong & {P_{i_1}}^{t-1} + \alpha_{in}[(e(i_1, j_b){Q_{j_b}}^{t-1} - \lambda_p {P_{i_1}}^{t-1}] - \alpha_{fr} D^{fr}_{P_{i_1}}, \\ {Q_{j_b}}^t \cong & {Q_{j_b}}^{t-1} + \alpha_{in}[(e(i_1, j_b){P_{i_1}}^{t-1} - \lambda_q {Q_{j_b}}^{t-1}] - \alpha_{fr} D^{fr}_{Q_{j_b}}. \end{aligned} \tag{15}$$

Notice that three threads update Q_{j_0}, Q_{j_a}, and Q_{j_b} independently and simultaneously. P_{i_1} is updated simultaneously by all threads. In other words, when the thread blocks select random entries in the same row, there is an over-writing problem in updating the user-feature matrix. Selecting entries in the same column of R is problematic for Updating the item-feature matrix. This problem has to be solved.

4 Proposed parallel FAMSGD approach

This section introduces our proposed parallel method, Fractional Adaptive Multi-stream Stochastic Gradient Descent (FAMSGD).

4.1 problems to parallel matrix factorization

This paper uses two paralleling techniques based on a single GPU to reduce the execution time and solve the overwriting problem explained in Section 3.2. It also addresses two CUDA programming concerns.

1- *Coalesced memory access:* Some SGD-MF algorithms access entries of the rating matrix and user/item identities randomly. Such approaches suffer from memory discontinuity, resulting in a high cache miss rate and a severe performance reduction.

2- *Locking problem:* If the rating matrix (R) is unbalanced, i.e., the unknown entries are divided non-uniformly across all positions, it can cause different thread blocks to bear other computational loads. Therefore, some thread blocks with fewer computations will idle, waiting for others.

According to the discussed issues, we need to answer these questions:

- How can we design a CUDA-based parallel approach to get the highest efficiency?
- How can we adapt large-scale datasets with GPU memory?

4.2 Parallel fractional calculus

We use SGD-MF to predict user preferences in recommender systems. The described SGD method in Section 3.1 is suitable for large datasets because of the low complexity of the algorithm. This algorithm is inherently serial and is non-trivial for paralleling, particularly in large-scale applications. However, as we mentioned earlier, the attempt faces an over-writing problem when different threads select entries that belong to the same row or column of the rating matrix at the same time. To solve the issue, we need to design an approach in which the threads select entries in the same row of the rating matrix (R) and update only Q. Similarly, choosing entries in the same column of R results in updating P. We adopt the idea presented by *Li* et al. [14] for splitting the distance function $d(P, Q)$, defined in Eq. 4 as shown in Eq. 16,

$$d = \sum_{i=1}^{m} |\Omega_i| d_i = \sum_{j=1}^{n} |\overline{\Omega}_j| \overline{d}_j, \tag{16}$$

where

$$d_i = \frac{1}{|\Omega_i|} \sum_{j \in \Omega_i} (e^2(i,j) + \lambda_p \|P_i\|^2 + \lambda_q \|Q_j\|^2), \tag{17}$$

and

$$\overline{d_j} = \frac{1}{|\overline{\Omega}_j|} \sum_{i \in \overline{\Omega}_j} (e^2(i,j) + \lambda_p \|P_i\|^2 + \lambda_q \|Q_j\|^2). \tag{18}$$

The goal is to minimize the distance function (i.e., minimize the functions defined in Eq. 17 and Eq. 18) by minimizing $\sum_{i=1}^{m} |\Omega_i| d_i$ and $\sum_{j=1}^{n} |\overline{\Omega}_j| \overline{d}_j$, respectively. For example, if we maintain P_i unchanged in the equation $\overline{d}_j$ for $i \in \overline{\Omega}_j$, then $\overline{d}_j$, $j \in \{1, ..., n\}$ are independent of each other. This is the same for d_i, $i \in \{1, ..., m\}$. Hence, by utilizing gradient descent in a specific epoch based on fixed parameters, we can solve the following optimization problem, where α_{in} and α_{fr} are the values of learning rates.

$$\underset{\mathbf{P_i}}{\operatorname{argmin}}\, \mathbf{d_i}: \quad {P_i}^t = {P_i}^{t-1} - (\alpha_{in}, \alpha_{fr}) \begin{pmatrix} \nabla d_i({p_i}^{t-1}) \\ \nabla^{fr} d_i({P_i}^{t-1}) \end{pmatrix}. \tag{19}$$

$$\underset{\mathbf{Q_j}}{\operatorname{argmin}}\, \overline{\mathbf{d}}_\mathbf{j}: \quad {Q_j}^t = {Q_j}^{t-1} - (\alpha_{in}, \alpha_{fr}) \begin{pmatrix} \nabla \overline{d}_j({Q_j}^{t-1}) \\ \nabla^{fr} \overline{d}_j({Q_j}^{t-1}) \end{pmatrix}. \tag{20}$$

4.3 Adaptive learning rates

We use two distinct learning rates of α_{in} and α_{fr} in Eq. 19 and Eq. 20. These learning rates significantly impact the accuracy and convergence rate of gradient-based learning methods such as FAMSGD. Assigning large values to these learning rates results in an imprecise outcome, while small values lead to performance degradation.

We utilize the Ada-Error approach [22] to adjust the adaptive learning rate parameter (α_{in} and α_{fr}) in each epoch. In this approach, the values of learning rate parameters are sensitive to the noisiness of ratings in the user-items rating matrix R. The main idea is to allocate lower learning rates to entries with larger training errors. In other words, the lower the error, the higher the learning rate. The learning rate in the t^{th} iteration is defined in Eqs. 21 and 22, where Lr_{in} and Lr_{fr} are the predefined values for the learning rates in the process of learning, $E^{(t-1)}(i,j) = \sum_{iter=0}^{t-1} (R(i,j) - \hat{R}(i,j)^{(iter)})^2$, ϵ is a very small constant value to avoid dividing by zero, and β is the other constant which prevents learning rates become infinitely small in higher iterations. The update equations of the FASGD method in Eq. 10 and Eq. 11 are obtained based on the definition of FSGD and by substituting formulas in Eqs. 19 and 20. We present $\alpha_{in}^{(t)}(i,j)$ and $\alpha_{fr}^{(t)}(i,j)$ in the form of α_{in} and α_{fr} respectively for simplification.

$$\alpha_{in}^{(t)}(i,j) = \frac{Lr_{in}}{\sqrt{E^{(t-1)}(i,j) + \epsilon}} + \beta, \tag{21}$$

Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212

Algorithm 1 FAMSGD

Input: Rating matrix: R, initial learning rates: Lr_{in} and Lr_{fr}, initial feature matrices: P and Q, $E_1^{(0)}(i,j) = E_2^{(0)}(i,j) = 0$ for $(i,j) \in \Omega_{train}$, regularization parameters: λ_p and λ_q, number of training iterations: N.
Output: P, Q.

```
 1: for $t = 1$ to $N$ do
 2:     for $j = 0$ to $n - 1$ do
 3:         for each element with an index $i$ in $\overline{\Omega_j}$ do
 4:             $E_1^{(t)}(i,j) \leftarrow E_1^{(t-1)}(i,j) + (R(i,j) - \hat{R}(i,j))^2$.
 5:             update $\boldsymbol{\alpha_{in}}$ by Eq. 21
 6:             update $\boldsymbol{\alpha_{fr}}$ by Eq. 22
 7:             update $\mathbf{Q_j}$ by Eq. 11
 8:         end for
 9:     end for
10:     for $i = 0$ to $m - 1$ do
11:         for each element with an index $j$ in $\Omega_i$ do
12:             $E_2^{(t)}(i,j) \leftarrow E_2^{(t-1)}(i,j) + (R(i,j) - \hat{R}(i,j))^2$
13:             update $\boldsymbol{\alpha_{in}}$ by Eq. 21
14:             update $\boldsymbol{\alpha_{fr}}$ by Eq. 22
15:             update $\mathbf{P_i}$ by Eq. 10
16:         end for
17:     end for
18: end for
19: return $P$, $Q$.
```

$$\alpha_{fr}^{(t)}(i,j) = \frac{Lr_{fr}}{\sqrt{E^{(t-1)}(i,j) + \epsilon}} + \beta. \tag{22}$$

Algorithm 1 presents the pseudo code of the FAMSGD procedure, where each training epoch contains two independent phases. In the first phase, the cumulative error is calculated, and the value of learning rates is recomputed. Then, the item-feature matrix is updated (steps 2-9). Similarly, after updating learning rates in the second phase, the user-item matrix will be updated (steps 10-17).

4.4 Decentralized and centralized paralleling

We report two different scheduling approaches applied to utilize thread block efficiently. These strategies specify how thread blocks act in updating user and item feature matrices. We know that the large rating matrix used in recommender systems is sparse, and many entries have unknown values, so we transfer only known entries with their positions to GPU global memory. We employ two paralleling methods, i.e., decentralized and centralized approaches to update factor matrices in parallel.

The decentralized one is a row-wise updating method in which the thread blocks select entries in the same row of the rating matrix, as described in Algorithm 2. Steps 2 to 12 of this algorithm specify the update procedure in matrix Q according to defined equations 11, 21, and 22. In a training epoch, a thread block selects a special column Q_j. Then, the error related to entry $R(i,j)$ is computed and added to the previous related cumulative error. In steps 7 and 8, the learning rate values are updated based on their initial value and the current error adaptively. Finally, each thread block updates the selected column. After completing the updating procedure of selected columns, the new ones that have not been updated in the current step are selected. It is important to note that the thread blocks access entries randomly, and there is no guarantee that a thread block updates the same

Algorithm 2 Decentralized Updating Q

Input: Rating matrix: R, initial learning rates: Lr_{in} and Lr_{fr}, initial feature matrices: P and Q, $E^{(0)}(i,j) = 0$ for $(i,j) \in \Omega_{train}$, regularization parameters: λ_p and λ_q, number of training iterations: N.
Output: Q.

```
 1: $tid \leftarrow$ Thread Block Id.
 2: for $t = 1$ to $N$ do
 3:     for $i = 0$ to $m - 1$ do
 4:         Parallel section:
 5:             select $j \in \Omega_i$. % i is the index of row used to
                update $Q_j$.
 6:             $E^{(t)}(i,j) \leftarrow E^{(t-1)}(i,j) + (R(i,j) - \hat{R}(i,j))^2$
 7:             update $\boldsymbol{\alpha_{in}}$ by Eq. 21
 8:             update $\boldsymbol{\alpha_{fr}}$ by Eq. 22
 9:             update $\mathbf{Q_j}$ by Eq. 11
10:         end parallel section
11:     end for
12: end for
13: return $Q$.
```

feature vector in two consecutive updating procedures. On the other hand, to take advantage of shared memory in GPU, we need to transfer the related vectors from global to the shared memory to update a special column Q_j. But we know only a small part of memory is devoted to the shared memory, so it compels the thread block to return Q_j back to global memory.

We describe a simple example of decentralized updating Q. We suppose we have two thread blocks $\{TB_0, TB_1\}$. They select two entries in row $i = 0$, i.e., $\{(0,1),(0,3)\}$, where $\{1,3\} \in \Omega_0$ and update $\{Q_1, Q_3\}$. If there are column indices in Ω_0 different from $\{1,3\}$, then they will be selected by two thread blocks and updated. Later, we explain how the user-feature matrix P will be updated in the decentralized method. We select two entries $\{(1,0),(2,0)\}$, where $\{1,2\} \in \overline{\Omega_0}$. The two thread blocks $\{TB_0, TB_1)\}$ update $\{P_1, P_2\}$.

Algorithm 3 describes the column-wise centralized method through steps 2 to 12. In a training epoch, each column is selected by one thread block, and each thread block will choose an index $i \in \overline{\Omega_j}$ independently in step5. In step 6, the related cumulative error up to iteration t is computed. In the next step, the value of learning rates is updated. Finally, each tread block updates its Q_j vector. If columns have not yet been processed, the thread blocks will continue by selecting these according to the total number of thread blocks (nb). We describe a simple example of centralized updating Q. Suppose we have two thread blocks $\{TB_0, TB_1\}$, they update $\{Q_0, Q_1\}$ for some $i_1 \in \overline{\Omega_0}, i_2 \in \overline{\Omega_1}$, respectively.

The updating process for user-feature matrices will be the same, i.e., the thread blocks update two vectors P_0 and P_1 for some $j_1 \in \Omega_0, j_2 \in \Omega_1$.

4.5 GPU kernel planning

Memory coalescing points to unifying multiple memory access into a single transaction. A warp including 32 consecutive threads can access the sequential 128-byte in global memory in a single transaction if the memory access is not unaligned and sparse. We set the value of r as a coefficient of

Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212

Algorithm 3 Centralized Updating Q

Input: Rating matrix: R, initial learning rates: Lr_{in} and Lr_{fr}, initial feature matrices: P and Q, $E^{(0)}(i,j) = 0$ for $(i,j) \in \Omega_{train}$, regularization parameters: λ_p and λ_q, number of training iterations: N.
Output: Q.

```
1:  $tid \leftarrow$ Thread Block Id.
2:  for $t = 1$ to $N$ on the interval $nb$ do
3:      for $j = tid$ to $n - 1$ do
4:          Parallel section:
5:              select $i \in \overline{\Omega_j}$. % i is the index of row used to update $Q_j$.
6:              $E^{(t)}(i,j) \leftarrow E^{(t-1)}(i,j) + (R(i,j) - \hat{R}(i,j))^2$
7:              update $\boldsymbol{\alpha_{in}}$ by Eq. 21
8:              update $\boldsymbol{\alpha_{fr}}$ by Eq. 22
9:              update $\mathbf{Q_j}$ by Eq. 11
10:         end parallel section
11:     end for
12: end for
13: return $Q$.
```

32 to process the global memory in a coalescing way. In both updating methods, P_i and Q_j are accessed continuously by thread blocks on GPU global memory via 32, 64, or 128-byte memory transactions. Both paralleling strategies (Algorithms 2 and 3) first compute $E(i,j)$ to update learning rates, P_i and Q_j. So, vector dot multiplication is a pre-computing step in each updating epoch. Each thread block contains r threads, and all threads participate cooperatively according to scheduling strategies. Therefore, storing the vectors P_i and Q_j in the shared memory of thread blocks is essential.

5 Experiments

This section presents the performance evaluation of the devised method. We evaluate the performance of FAMSGD and compare it with other parallel matrix factorization methods mentioned in Section 2.3. We analyze the sensitivity of the proposed method to the main hyper-parameters. We also report the speed-up of FAMSGD and discuss the performance results.

5.1 Evaluation settings

Datasets: This research utilizes three benchmark datasets (Boardgamegeek (BGG), Netflix, and Yahoo!Music). We split the datasets with the ratio of $70\% : 30\%$ by sampling randomly to prepare train/test sets. On the BGG dataset as a weighted bipartite graph, the ratings are preprocessed and varied from 1 to 10 but on Netflix and Yahoo!Music, the ratings are in the interval $[1-5]$. The statistics of datasets are available in Table 1. The density of the rating matrix is computed using the following equation:

$$density = \frac{num_{ratings}}{num_{users} * num_{items}}, \tag{23}$$

where $num_{ratings}$ are the number of recorded ratings, num_{users} and num_{items} are the numbers of users and items. By applying Eq. 23, the density value of datasets is computed and reported in Table 1. It indicates the input matrix R is very sparse, requiring the calculations to perform only based on definite entries.

Platform: We use a server with four AMD Opteron 6386 SE processors in this study. There are 16 cores on each processor (64 CPU cores on a board) and 128 GB of main memory. The parallel processing is done on GPU K20x and GeForce RTX 2080 Ti. All comparisons are based on the same platform.

Parameter settings: Hyper-parameters such as λ_P and λ_Q are common between FAMSGD and baseline methods, while the parameters fr, β, ϵ are special hyper-parameters related to the FAMSGD algorithm. The parameter β_{cu} is related to the CuMF_SGD method. The values selected for learning rates (Lr_{in}, Lr_{fr}) are initial. Their values are determined adaptively during the training process. We tune the values using the validation procedure mainly. Moreover, to have a fair comparison with earlier methods [14], [25], [39], [41], we borrow some of the values from previously suggested ones. We also check the range of parameters in the algorithms and run them many times to fine-tune them regarding their best results. Two feature matrices (P and Q) are initialized randomly using a uniform distribution. Table 1 lists the parameters' values for each dataset.

Implementations: CCD++ [25], LIBMF [41], CuMF_SGD [39] have publicly available implementations. We implement CUMSGD based on the algorithms presented in [14].

Evaluation: To measure the accuracy of the prediction model and compare it with the baseline methods, we adopted Root Mean Square Error (RMSE) and Precision metrics. We compute the prediction errors based on the standard deviation of residuals on the test dataset using Eq. 24, where Ω_{test} includes the indices of non-zero elements in the test dataset; $R(i,j)$, $\hat{R}(i,j)$ represent the actual and predicted value, respectively.

$$RMSE_{test} = \sqrt{\frac{\sum_{(i,j)\in\Omega_{test}} \left(R(i,j) - \hat{R}(i,j)\right)^2}{n_{test}}} \tag{24}$$

Precision is calculated through Eq. 25, where I_r lists top N recommended items and I_u lists relevant items recommended in top N.

$$Precision@N = \frac{|I_r \cap I_u|}{|I_r|} \tag{25}$$

We have two types of figures in the experiments. The first type shows the impact of each parameter value on the proposed algorithm. We discuss them entirely and get a model with a steady-state based on the results. The second type of figure compares the proposed method with baseline methods. In this type, we reported the best results obtained from executing different methods (the baseline methods and FAMSGD) considering relative numbers. This means we considered relative numbers for each algorithm and chose the best result.

5.2 Sensitivity analysis

We aim to set an appropriate value for each hyper-parameter by analyzing the model's sensitivity to each one.

5.2.1 *Sensitivity to regularization parameters*

A proper regularization coefficient can prevent the learned models from over-fitting problems. λ_p and λ_q are regularization parameters defined in the SGD problem. Figure 1 shows

Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212

TABLE 1: The statistics and parameters for each dataset

Dataset	m	n	#Interactions	Density	Lr_{in}	Lr_{fr}	fr	λ_P	λ_Q	β_{cu}	β	ϵ
BGG	388,413	87,208	47,351,708	.14	.01	.01	.9	.01	.01	.1	.0001	.0001
Netflix	2,649,429	17,770	100,480,507	1.17	.05	.05	.75	.05	.05	.3	.0001	.0001
Yahoo!Music	1,823,178	136,735	717,871,377	.29	.01	.01	.9	.05	.05	.2	.0001	.0001

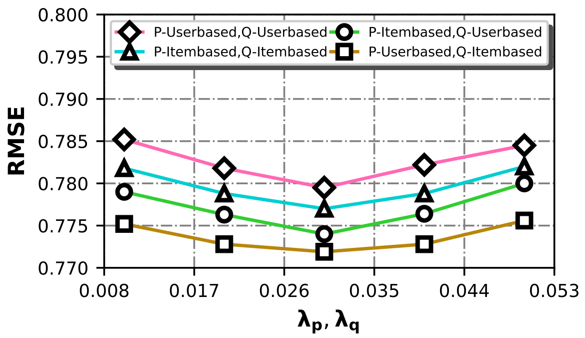


Fig. 1: Sensitivity analysis to regularization parameter for different update procedures in FAMSGD on the Netflix dataset ($r = 128$, $Lr_{in} = Lr_{fr} = 0.01$, $\beta = 1e-4$).

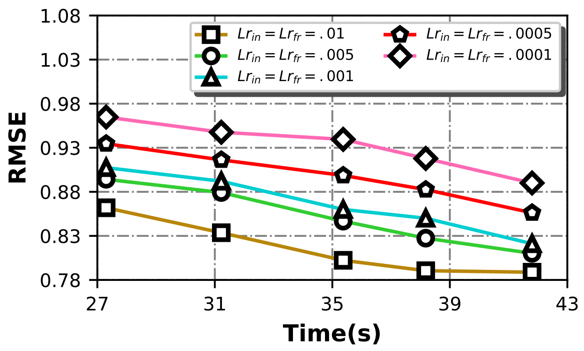


Fig. 2: Learning rate sensitivity analysis for different update procedures in FAMSGD method on the Netflix dataset ($r = 128$, $\lambda_p = \lambda_q = 0.03$, $\beta = 1e-4$).

the accuracy of four update combinations of FAMSGD over different values of regularization coefficient. The figure shows that by differing the values of λ_p and λ_q from 0.01 to 0.05, FAMSGD gets the smallest value for test RMSE when $\lambda_p = \lambda_q = 0.03$. The figure also shows that four updating procedures have similar behavior by increasing the parameter value λ. This indicates that FAMSGD is not significantly sensitive to the regularization parameter since RMSE values do not change considerably by altering the regularization parameters.

5.2.2 Sensitivity to learning rates (Lr_{in} and Lr_{fr})

The learning rates are employed in standard, and fractional gradient descent as the step sizes that influence convergence. They have an influential role in the accuracy, convergence rate, and stability. We consider adaptive learning rates for both integer and fractional-order in SGD. It eventually leads to better results. Figure 2 compares the accuracy of FAMSGD by considering different initial values for Lr_{in} and Lr_{fr}. The values vary from 0.01 to 0.0001. We compute the test RMSE of the proposed method during the time. The results show that FAMSGD achieves the best results when initial learning rates equal 0.01. Figure 2 indicates that at any point in time, with decrements of the primitive value of learning rates from 0.01 to 0.0001, the amplitude of error change is very small. This is because tuning the value of learning rate parameters during the training model is inspired by the AdaError strategy. We apply AdaError in the FAMSGD model to employ the corresponding learning rate value in each iteration; it reassures us about the convergence to the optimal point. There is a trade-off between optimization accuracy and convergence speed. When Lr_{in} and Lr_{fr} are initialized to a very small value, it has an undesirable and inevitable effect on optimization accuracy. On the other hand, setting a very large value for Lr_{in} and Lr_{fr} affects the convergence rate. Nevertheless, these problems are solved in determining the value of learning rates.

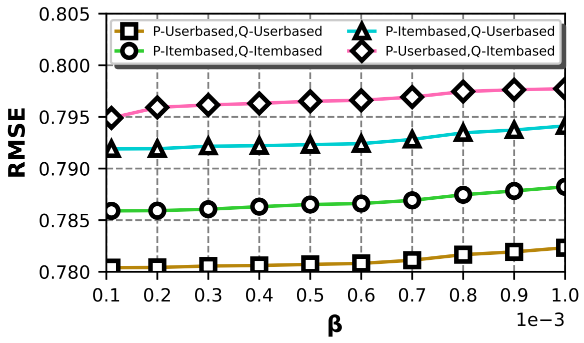


Fig. 3: β sensitivity analysis for different update procedures in FAMSGD on the Netflix dataset ($r = 128$, $\lambda_p = \lambda_q = 0.03$, $Lr_{in} = Lr_{fr} = 0.01$).

5.2.3 Sensitivity to controller coefficient (β)

Two constant parameters, i.e., β and ϵ are required to update adaptive learning rates (Lr_{in} and Lr_{fr}) in each iteration. The purpose of using this parameter is to control the decrements of the learning step and inhibit it from becoming too small. Figure 3 shows that by increasing the value of β from 1e−4 to 1e−3, the test RMSE increases about 0.003. It refers to the stability of the model. As we see in the four updating procedures of FAMSGD, the smaller value of parameter β causes smaller learning steps. Consequently, it improves the test RMSE slightly and reassures us about better convergence.

5.2.4 Sensitivity to fractional-order (fr)

The fractional-order fr is an essential factor distinguishing FSGD from the SGD approach. The value of fractional-order

Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212

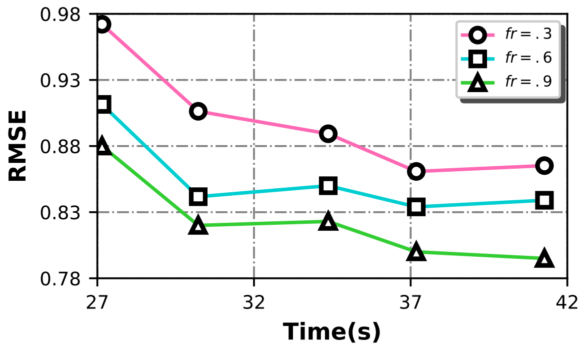


Fig. 4: Sensitivity analysis to fractional-order parameter on the Netflix dataset ($r = 128, \lambda_p = \lambda_q = 0.03, Lr_{in} = Lr_{fr} = 0.01, \beta = 1e-4$).

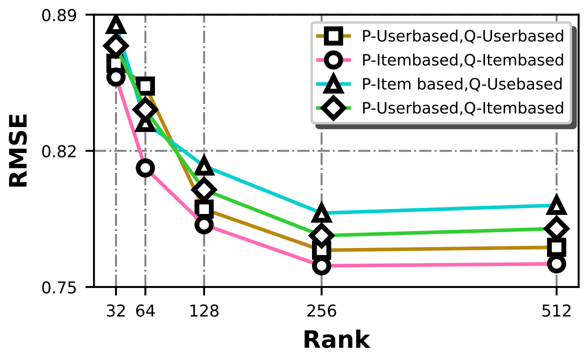


Fig. 5: Sensitivity analysis to the number of features (rank) in different update procedures of FAMSGD method on the Netflix dataset ($\lambda_p = \lambda_q = 0.03$, $\alpha_{in} = \alpha_{fr} = 0.01$, $\beta = 1e-4$, $fr = 0.9$).

is selected in the interval $(0, 1)$. The question is: what is an appropriate parameter value to achieve a steady-state performance?

Graphical diagrams represent the error changes during the time. Figure 4 shows that increasing the value of fr from 0.3 to 0.9 improves the accuracy gradually, and the optimal result is obtained with $fr = 0.9$.

5.2.5 *Sensitivity to rank (r)*

The Number of features r in a latent vector is a momentous parameter that affects accuracy and convergence behavior. Figure 5 compares the accuracy of FAMSGD in four different updating procedures through test RMSE with different rank values. We can see that the test RMSE of FAMSGD decreased consistently by increasing the rank value from 32 to 512. On the other hand, it confirms that increasing the value of rank in the presented method does not cause over-fit problems. It can get acceptable results in generalization performance.

In each case of sensitivity analysis, we change only the value of the noted parameter and keep other parameters fixed (i.e., equal to the optimum values obtained from the previous analysis) to investigate the effect of changes in the parameter from an accuracy and performance viewpoint.

5.3 FAMSGD performance

5.3.1 *Memory allocation*

The training process is accomplished by executing kernel functions that use hyper-parameters. It has been mentioned that static values are the number of factors (r), the total iterations (N), the initial learning rates (Lr_{in} and Lr_{fr}), auxiliary parameters in adaptive learning β and ϵ, the regularization coefficients λ_p and λ_q, the fractional-order fr. From the paralleling point of view, allocating constant memory to these parameters is better. They can be cached to be used during running time. We store frequently used values in registers to accelerate building the prediction model. Registers make it possible to access memory faster than global memory in GPU considerably. It is also useful to store feature vectors and instantaneous accumulated errors of definite entries in the shared memory of thread blocks. For example, we need to update p_i,q_j for entry $R(i, j)$. So, we temporarily store these vectors and $E(i, j)$ in shared memory.

5.3.2 *Time complexity*

The number of features is one important factor that affects the model's accuracy. It is necessary to compute temporary accumulated errors to adapt learning rates and update vectors during updating feature matrices. The higher values of (r) increase the execution time. In this step, dot product and vector addition are the main operations to compute error and update feature vectors to predict ratings. The time complexity of vector multiplication and error summation is $O(r)$ and $O(n_{test})$, respectively (where n_{test} is the number of data points in the test dataset). An optimized reduction in both of them decreases the time complexity to $O(\log_2(r))$ and $O(\log_2(n_{test}))$.

We decreased the thread synchronization mechanism described before. Figure 6 demonstrates the effect of the optimization approach on speed-up, where the y-axis presents the amount of speed-up based on different values of r. The speed-up is calculated approximately (Amdahl's Law) by Eq. 26.

$$S = \frac{T_{seq}}{T_{parallel}} = \frac{N_{epoch} \cdot t_s}{N_{epoch} \cdot t_p}, \tag{26}$$

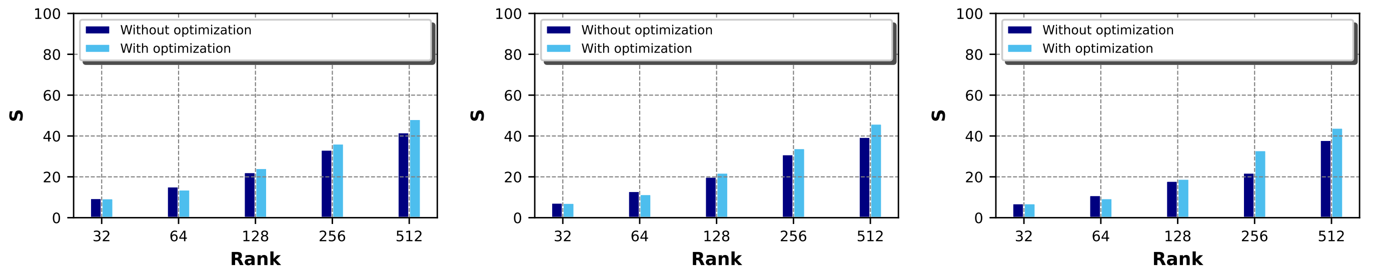
where N_{epoch}, t_s, t_p represents the number of training iterations and the time of executing an epoch in sequential and parallel form, respectively. The figure shows that by increasing the value of r (especially larger than 128), the improvement of the synchronization method is more noticeable and shows the difference between the cost of warp synchronization instructions and warp scheduling.

5.3.3 *Paralleling speed-up*

We implemented parallel FAMSGD on two different GPU models (K20x and GeForce RTX 2080 Ti) and computed the value of speed-up based on the sequential implementation time on a single thread CPU. Increasing the number of features r and executing the update procedure through some training epochs is necessary to get more accuracy. As a result, the run-time increases extremely. It is worthy of utilizing the benefits of GPU comprehensively. We set the number of threads in each thread block equal to the rank(r). To focus on the speed-up performance, we select

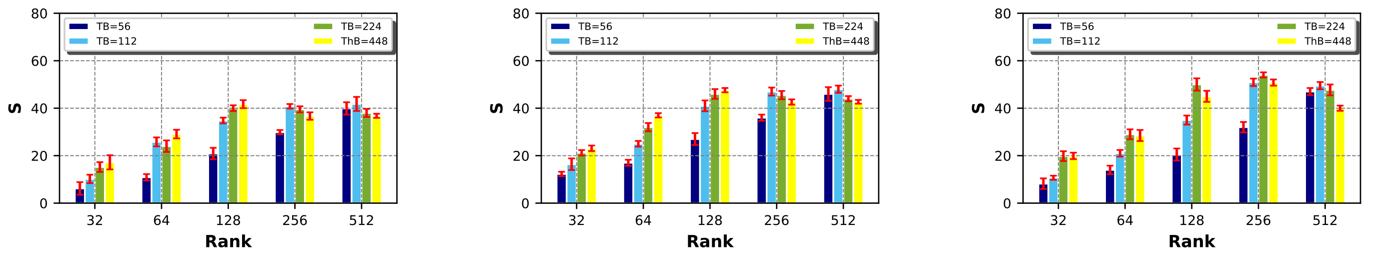
Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212



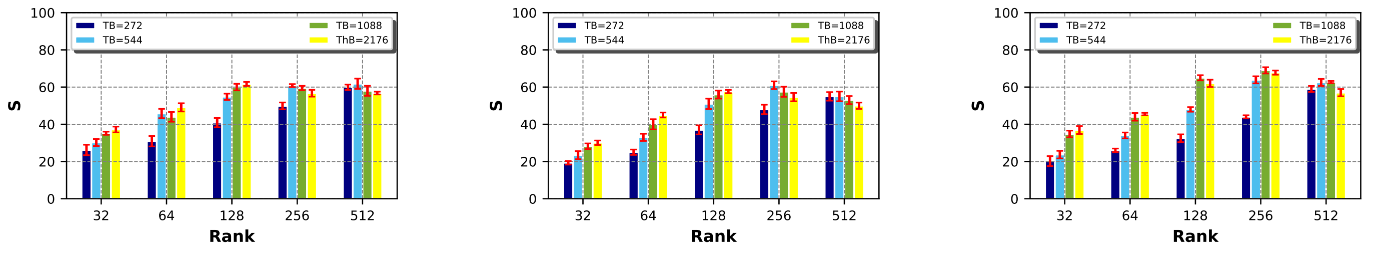
(a) BGG (b) Netflix (c) Yahoo!Music

Fig. 6: Performance comparison between reduced synchronization (with optimization) and the maximum synchronization (Without optimization) instruction method.



(a) BGG (b) Netflix (c) Yahoo!Music

Fig. 7: FAMSGD speed-up comparison on K20x GPU with a different number of thread blocks.



(a) BGG (b) Netflix (c) Yahoo!Music

Fig. 8: FAMSGD speed-up comparison on GeForce RTX 2080 Ti with a different number of thread blocks.

various values for $r \in \{32, 64, 128, 256, 512\}$. We also set the value of r as a multiple of warp size to execute warp synchronization. A thread block can access each feature vector continuously due to the coalesced access on GPU global memory via 32, 64, or 128-byte memory transactions.

In the paralleling approach, we can control the occupancy by tuning the number of threads and thread blocks, namely, the value of r affects occupancy in addition to accuracy. It is possible to get a better result by combining two different updating procedures rather than a single one. This is because of the nature of decentralized and centralized strategies. We aggregate these approaches since updating procedures based on decentralized road-maps need more data access time than a centralized one from GPU global memory. Data load balancing is more feasible through a decentralized updating procedure than a centralized one.

In Figures 7 and 8, the y-axis shows the speed-up performance of the FAMSGD method by implementing it through the different number of thread blocks on GPUs. In each figure, the number of thread blocks is adjusted according to the number of GPU's stream multiprocessors. For K20x GPU, when the number of thread blocks equals 56, the speed-up (S) increases as r changes from 32 to 512; the model can obtain maximum speed-up and full occupancy for $r = 512$. It is also verified about GeForce RTX 2080 Ti. In other words, the high occupancy value is reached by approaching the number of active threads to the total number of threads.
Implementing across GPU architectures demonstrates the capability of the proposed algorithm across other GPU architectures. GeForce RTX 2080 Ti GPU obtains better speed-up than K20x because of more stream multiprocessors (SMs), increased memory clock rate, and higher bandwidth than the other one. The result confirms firmly applicability of the proposed that can be employed on different GPUs.

5.4 Comparison of parallel FAMSGD with baselines

This section compares parallel FAMSGD with baseline methods. Figure 9 shows the result of implementing baseline parallel matrix factorization methods, including

Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212

a non-SGD-based method (CCD++) and three SGD-based methods (CuMF_SGD, CUMSGD, LIBMF) on all datasets. All CUDA-based approaches implemented on two GPUs and the suffix "_K" and "_G" are used in the figures for K20x and GeForce RTX 2080 Ti, respectively. The comparison is based on RMSE over processing time for a constant value of $r = 128$.
The minimization problem defined in Eq. 4 is non-convex; on the other hand, The CCD++ method is inherently more greedy than the SG-based methods to find the optimal point. It is possible that CCD++ gets stuck in the local minimum during the training process and converges pre-maturely. The behavior of this optimization method is visible in Figure 9. It is shown that CCD++ is very fast initially, even at the speed of the same as FAMSGD, but it becomes slower. On the contrary to CCD++, the SG-based method escapes from this trap due to random behavior.

The number of floating-point instructions in SGD based methods is lower than memory access instructions. LIBMF tries to resolve the memory-bound limitation by maintaining all frequently used data samples and related feature vectors in the CPU cache. Figure 9 shows that LIBMF loses its efficiency when processing large datasets due to the limited cache capacity. LIBMF has some computational complexity in prepossessing; also, it includes some time-consuming steps: complex scheduler, random shuffling, and sorting the blocks. CuMF_SGD (Batch-Hogwild!) is a system-level GPU optimization method outperforming LIBMF. Although it has less complexity in the prepossessing phase than LIBMF, the prepossessing time is considered a major part of processing time in this method. Access to the rating matrix is accelerated by utilizing spatial data locality in LIBMF. We observe that FAMSGD strives to optimize the update scheme and accelerate the process among the four SG-based parallel matrix factorization methods. It converges faster than the other methods while it can obtain the best RMSE accuracy, especially for large datasets. We use the weighted sum of standard and fractional derivatives to solve the optimization problem in Eq. 4, which boosts exploring the solution space and reaching an optimum point.

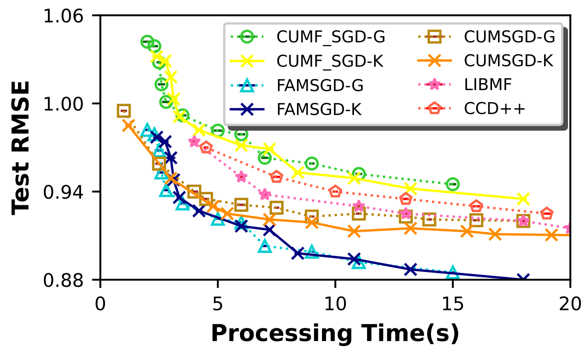
Figure 10 compares the scalability of FAMSGD with other GPU based methods. We use the number of updates per second, shown in Eq. 27 [39], as a metric.

$$\#update/s = \frac{\#Samples * \#Iteration}{ProcessingTime}, \tag{27}$$

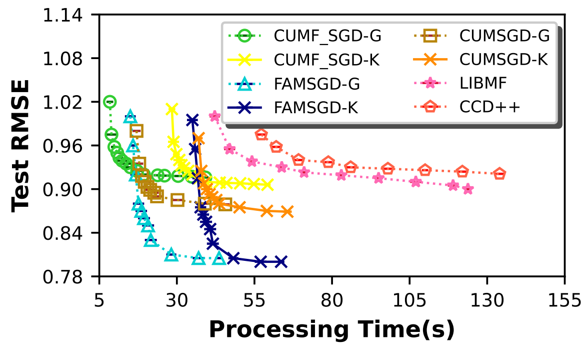
where #Samples and #Iterations indicate the number of ratings in the data matrix and iterations respectively.

It is illustrated that by increasing the number of parallel worker near 400, CuMF_SGD has the highest score based on this metric. Still, it does not show a steady behavior with increasing the number of parallel workers. In FAMSGD and CUMSGD, we observe steady improvements in the number of updates/s by increasing the number of parallel workers, which means better scalability. While all three methods have approximately the same number of updates per second for greater number of parallel workers.

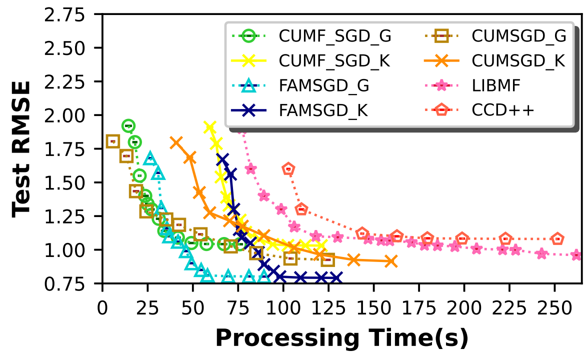
Table 2 compares the recommendation accuracy (precision and RMSE) of the proposed method with four other methods on datasets. RMSE is computed by comparing the predicted value to the true rating for each known user-item entry, while precision@N is the proportion of recommended items in the top-N set that are relevant. The substantial performance of the proposed method is clearly seen on both Precision@N and RMSE when N increases from 1 to 20. As you can see in the figure, FAMSGD outperforms all algorithms considering two metrics. The first reason for the superior performance of the FAMSGD method is the complex mathematics behind fractional calculus, which enables the FAMSGD method to adapt the scheme variables of rigid systems more precisely [21]. The second reason is using an adaptive learning rate that reduces the oscillating behavior of the model near the local minimum and develops a weighting strategy to prevent the model from overreacting to the noise.



(a) BGG



(b) Netflix



(c) Yahoo!Music

Fig. 9: Comparison of FAMSGD accuracy over processing time with the baseline methods.

Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212

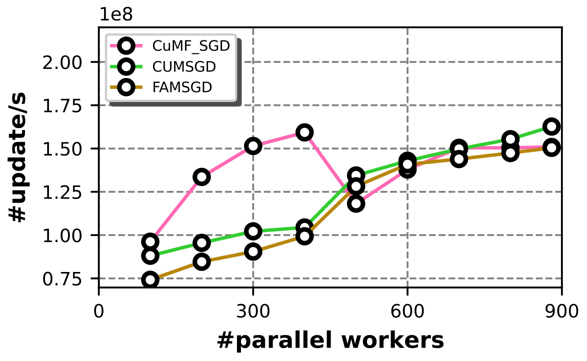


Fig. 10: Performance comparison of GPU based methods on Yahoo!Music dataset.

TABLE 2: Precision comparison between the proposed method and the baseline methods (CCD++, (CuMF_SGD [39], CUMSGD [14], LIBMF [41]) on the Netflix and Yahoo!Music datasets

		Precision@N				RMSE
Data	Method	N=1	N=5	N=10	N=20	
BGG	CCD++	0.1349	0.1423	0.1102	0.0862	0.9549
	CuMF_SGD	0.3651	0.3094	0.2539	0.1161	0.9207
	CUMSGD	0.3728	0.3283	0.2691	0.1209	0.9106
	LIBMF	0.369	0.3106	0.2512	0.0994	0.9176
	FAMSGD	**0.3723**	**0.3152**	**0.2783**	**0.1388**	**0.8846**
Netflix	CCD++	0.1559	0.1163	0.1032	0.0752	0.9234
	CuMF_SGD	0.3114	0.2369	0.1893	0.1361	0.91144
	CUMSGD	0.2918	0.2173	0.1649	0.1105	0.9123
	LIBMF	0.2761	0.1855	0.1416	0.1032	0.9282
	FAMSGD	**0.3513**	**0.2642**	**0.2138**	**0.1508**	**0.8012**
Yahoo!Music	CCD++	0.2861	0.1814	0.1239	0.1047	0.8521
	CuMF_SGD	0.3252	0.2319	0.2054	0.1786	0.9293
	CUMSGD	0.3141	0.2412	0.1918	0.1432	0.8319
	LIBMF	0.3399	0.2715	0.2172	0.1739	0.8452
	FAMSGD	**0.3482**	**0.2457**	**0.2304**	**0.1824**	**0.8044**

5.5 Discussion

The evaluation results show some aspects of accuracy, convergence, and execution time comparison between parallel FAMSGD with other methods:

1) Some of the previous works have a preprocessing phase devoted a major part of processing time. This phase includes the subset of these procedures: shuffling and partitioning the rating matrix R into some blocks, sorting these blocks, and using a complicated scheduler to assign the blocks to processing units. So, proposing an approach to resolve these prerequisites has a noticeable effect on improving the execution time, especially in online systems.

2) Recommender systems are used commonly in big data environments, and it is crucial to design a scalable algorithm. SGD is an efficient method used in MF. It is a memory-bound method, so memory access utilization affects the performance of SGD-MF. It prompted us to design a method that aligns the memory access of feature vectors and access the memory coalesce.

3) Recommender systems are highly dynamic systems. User preferences, the number of items, and the users change continuously. The sparsity in the rating matrix results in load imbalance if the dataset is partitioned and the blocks are assigned to processing units uniformly.

4) In real-world recommender systems, the user-item rating matrices are sparse and noisy. It is important to train a model considering different levels of noisiness in data. Setting appropriate learning rates prevents the model from overreacting to noises or being prone to noisy data points.

5) It is important to highlight that adding fractional derivative in addition to standard one in gradient descent matrix factorization increases the execution time. Therefore, it is necessary to employ special approaches in high-performance computing to accelerate the method.

6 Conclusion

The gradient-based learning approaches are promising matrix factorization methods that address the data sparsity problem in collaborative filtering recommender systems. However, these methods have slow convergence rates. Besides, their convergence rates need further improvements. Parallel fractional calculus was applied in different science such as mechanics, but there is no paralleling fractional SGD for recommender systems. Considering the nature of the recommender systems, such as sparsity in data and the need for tuning parameters, we cannot use the previous parallel fractional calculus for the recommender systems. Here, we have parallelized and modified fractional SGD to be adaptive in calculating the learning rate for the recommender systems in big sparse data. This increases the accuracy of SGD-MF as well as converging fast. We investigated the performance of FAMSGD using a variety of sensitivity analyses on the hyperparameters. We discussed run-time optimization and utilizing the GPU power. Multiple techniques were applied in the implementation to use memory and to set up GPU parameters. We also compared its performance with the state-of-the-art baseline methods. The results show the improvements of FAMSGD in precision and RMSE using real-world datasets.

Acknowledgments

The authors would like to thank the Javaneh grant project for supporting this research in the context of project 02-99-02-000343.

References

[1] Rae Yule Kim. The impact of covid-19 on consumers: Preparing for digital sales. *IEEE Engineering Management Review*, 48(3):212–218, 2020.

[2] I. Fernández-Tobías, M. Braunhofer, M. Elahi, F. Ricci, and I. Cantador. Alleviating the new user problem in collaborative filtering by exploiting personality information. *User Modeling and User-Adapted Interaction*, 26(2-3):221–255, 2016.

[3] Leyang Xue, Peng Zhang, and An Zeng. Enhancing the long-term performance of recommender system. *Physica A: Statistical Mechanics and its Applications*, 531:121731, 2019.

[4] Maryam Khanian Najafabadi, Azlinah Hj Mohamed, and Mohd Naz'ri Mahrin. A survey on data mining techniques in recommender systems. *Soft Computing*, 23(2):627–654, 2019.

[5] Junmei Feng, Zhaoqiang Xia, Xiaoyi Feng, and Jinye Peng. Rbpr: A hybrid model for the new user cold start problem in recommender systems. *Knowledge-Based Systems*, 214:106732, 2021.

[6] Deepak Kumar Panda and Sanjog Ray. Approaches and algorithms to mitigate cold start problems in recommender systems: a systematic literature review. *Journal of Intelligent Information Systems*, pages 1–26, 2022.

Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212

[7] Sajad Ahmadian, Nima Joorabloo, Mahdi Jalili, and Milad Ahmadian. Alleviating data sparsity problem in time-aware recommender systems using a reliable rating profile enrichment approach. *Expert Systems with Applications*, 187:115849, 2022.

[8] Nima Joorabloo, Mahdi Jalili, and Yongli Ren. Improved recommender systems by denoising ratings in highly sparse datasets through individual rating confidence. *Information Sciences*, 601:242–254, 2022.

[9] Fan Yang, Huaqiong Wang, and Jianjing Fu. Improvement of recommendation algorithm based on collaborative deep learning and its parallelization on spark. *Journal of Parallel and Distributed Computing*, 148:58–68, 2021.

[10] Hatem Moumni and Olfa Hamdi-Larbi. An efficient parallelization model for sparse non-negative matrix factorization using cusparse library on multi-gpu platform. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 161–177. Springer, 2021.

[11] Qinyong Wang, Hongzhi Yin, Tong Chen, Junliang Yu, Alexander Zhou, and Xiangliang Zhang. Fast-adapting and privacy-preserving federated recommender system. *The VLDB Journal*, pages 1–20, 2021.

[12] Yongjie Du, Deyun Zhou, Yu Xie, Jiao Shi, and Maoguo Gong. Federated matrix factorization for privacy-preserving recommender systems. *Applied Soft Computing*, 111:107700, 2021.

[13] A. Moradi Dakhel, H. Tabatabaee Malazi, and M. Mahdavi. A social recommender system using item asymmetric correlation. *Applied Intelligence*, 48(3):527–540, 2018.

[14] Hao Li, Kenli Li, Jiyao An, and Keqin Li. Msgd: A novel matrix factorization approach for large-scale collaborative filtering recommender systems on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, 29(7):1530–1544, 2017.

[15] Yilmaz Ar. An initialization method for the latent vectors in probabilistic matrix factorization for sparse datasets. *Evolutionary Intelligence*, pages 1–13, 2019.

[16] T. Tran, K. Lee, Y. Liao, and D. Lee. Regularizing matrix factorization with user and item embeddings for recommendation. In *Proc.of the 27th ACM Intl. Conf. on Information and Knowledge Management*, pages 687–696, 2018.

[17] Jinli Li, Ye Yuan, Tao Ruan, Jia Chen, and Xin Luo. A proportional-integral-derivative-incorporated stochastic gradient descent-based latent factor analysis model. *Neurocomputing*, 427:29–39, 2021.

[18] Y. Wei, Y. Kang, W. Yin, and Y. Wang. Generalization of the gradient method with fractional order gradient direction. *Journal of the Franklin Institute*, 357(4):2514–2532, 2020.

[19] Maryam Mohammadi, Mahmood Fazlali, and Mehdi Hosseinzadeh. Accelerating louvain community detection algorithm on graphic processing unit. *The Journal of Supercomputing*, 77(6):6056–6077, 2021.

[20] Hao Li, Kenli Ge Li, Jiyao An, and Keqin Ge Li. An online and scalable model for generalized sparse non-negative matrix factorization in industrial applications on multi-GPU. *IEEE Transactions on Industrial Informatics*, 2019.

[21] Z.A. Khan, N.I. Chaudhary, and S. Zubair. Fractional stochastic gradient descent for recommender systems. *Electronic Markets*, 29(2):275–285, 2019.

[22] D. Li, C. Chen, Q. Lv, H. Gu, T. Lu, L. Shang, N. Gu, and S. Chu. AdaError: An adaptive learning rate method for matrix approximation-based collaborative filtering. In *Proc.of the 2018 World Wide Web Conf.*, pages 741–751, 2018.

[23] Jing Chen, Jianbin Fang, Weifeng Liu, Tao Tang, and Canqun Yang. clMF: A fine-grained and portable alternating least squares algorithm for parallel matrix factorization. *Future Generation Computer Systems*, 108:1192–1205, 2020.

[24] Jing Chen, Jianbin Fang, Weifeng Liu, and Canqun Yang. BALS: Blocked alternating least squares for parallel sparse matrix factorization on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, 32(9):2291–2302, 2021.

[25] I. Nisa, A. Sukumaran-Rajam, R. Kunchum, and P. Sadayappan. Parallel CCD++ on GPU for matrix factorization. In *Proc.of the General Purpose GPUs*, GPGPU-10, page 73–83, New York, NY, USA, 2017. Association for Computing Machinery.

[26] Prasad Bhavana and Vineet Padmanabhan. Matrix factorization of large scale data using multistage matrix factorization. *Applied Intelligence*, 51(6):4016–4028, 2021.

[27] R. Kaleem, A. Venkat, S. Pai, M. Hall, and K. Pingali. Synchronization trade-offs in gpu implementations of graph algorithms. In *2016 IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS)*, pages 514–523. IEEE, 2016.

[28] Hao Li, Keqin Li, Jiyao An, Weihua Zheng, and Kenli Li. An efficient manifold regularized sparse non-negative matrix factorization model for large-scale recommender systems on GPUs. *Information Sciences*, 496:464–484, 2019.

[29] Si-Thin Nguyen, Hyun-Young Kwak, Seok-Hee Lee, and Gwang-Yong Gim. Using stochastic gradient decent algorithm for incremental matrix factorization in recommendation system. In *2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 308–319. IEEE, 2019.

[30] Xin Luo, Dexian Wang, MengChu Zhou, and Huaqiang Yuan. Latent factor-based recommenders relying on extended stochastic gradient descent algorithms. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(2):916–926, 2019.

[31] Pu Chen and Hung Hsuan Chen. Accelerating matrix factorization by overparameterization. In Ana Fred and Kurosh Madani, editors, *DeLTA 2020 - Proceedings of the 1st International Conference on Deep Learning Theory and Applications*, DeLTA 2020 - Proceedings of the 1st International Conference on Deep Learning Theory and Applications, pages 89–97. SciTePress, 2020. Publisher Copyright: © 2020 by SCITEPRESS - Science and Technology Publications, Lda. All rights reserved; null ; Conference date: 08-07-2020 Through 10-07-2020.

[32] Mohamed A Nassar, Layla AA El-Sayed, and Yousry Taha. GPU_MF_SGD: A novel GPU-based stochastic gradient descent method for matrix factorization. In *Future of Information and Communication Conf.*, pages 271–287. Springer, 2018.

[33] Bing Tang, Linyao Kang, Li Zhang, Feiyan Guo, and Haiwu He. Collaborative filtering recommendation using nonnegative matrix factorization in GPU-accelerated Spark platform. *Scientific Programming*, 2021, 2021.

[34] N. Loizou, S. Vaswani, I.H. Laradji, and S. Lacoste-Julien. Stochastic polyak step-size for sgd: An adaptive learning rate for fast convergence. In *Intl. Conf. on Artificial Intelligence and Statistics*, pages 1306–1314. PMLR, 2021.

[35] N.I. Chaudhary, M.S. Aslam, D. Baleanu, and M.A.Z. Raja. Design of sign fractional optimization paradigms for parameter estimation of nonlinear hammerstein systems. *Neural Computing and Applications*, pages 1–19, 2019.

[36] Yun Tan, Zhiqiang He, and Baoyu Tian. A novel generalization of modified LMS algorithm to fractional order. *IEEE Signal Processing Letters*, 22(9):1244–1248, 2015.

[37] Z.A. Khan, S. Zubair, N.I. Chaudhary, M.A.Z. Raja, F. Khan, and N. Dedovic. Design of normalized fractional SGD computing paradigm for recommender systems. *Neural Computing and Applications*, pages 1–18, 2019.

[38] N.I. Chaudhary, S. Zubair, M.A.Z. Raja, N. Dedovic, et al. Normalized fractional adaptive methods for nonlinear control autoregressive systems. *Applied Mathematical Modelling*, 66:457–471, 2019.

[39] X. Xie, W. Tan, L. Fong, and Y. Liang. CuMF_SGD: Parallelized stochastic gradient descent for matrix factorization on GPUs. In *Proc.of the 26th Intl. Symp. on High-Performance Parallel and Distributed Computing*, pages 79–92, 2017.

[40] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in neural information processing systems*, 24, 2011.

[41] Wei-Sheng Chin, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. A fast parallel stochastic gradient method for matrix factorization in shared memory systems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(1):1–24, 2015.

[42] Wei-Sheng Chin, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. A learning-rate schedule for stochastic gradient methods to matrix factorization. In *Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pages 442–455. Springer, 2015.

Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Parallel and Distributed Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2022.3185212

Fatemeh Elahi is a Ph.D. candidate in Computer Science at the Department of Computer and Data Sciences at Shahid Beheshti University. Her main research interests are Machine Learning, Data science and Parallel Processing. She researches on accelerating Recommender Systems to apply on real applications.

Mahmood Fazlali received BSc in 2000 from Shahid Beheshti University (SBU), MSc. in 2004 from University of Isfahan, and PhD in 2010 from SBU in computer architecture. He performed postdoc researches on reconfigurable computing systems in computer engineering lab at Technical University of Delft (TUDelft). Now, he is assistant professor in department of data and computer sciences at SBU. His research interest includes high performance computing, parallel processing and big data processing. He published more than 40 papers in reputable journals and scientific conferences specially on high performance computing. He is associate editor in Elsevier array journal as well as reviewer for several IEEE, ACM Transactions and, Elsevier and Springer Journals.

Hadi Tabatabaee Malazi received his Ph.D. from the University of Isfahan in computer engineering (distributed systems). He is currently working as an assitant professor at Maynooth University. Before taking up this position, he worked as a research fellow at Trinity College Dublin in the Enable smart city research program. He was also an assistant professor with computer science and engineering faculty at Shahid Beheshti University (2013-2019). His main research areas are social sensing systems, pervasive computing, and edge intelligence.

Mehdi Elahi is an Associate Professor at the University of Bergen (UiB), Norway. Before joining this university, in 2014, Mehdi Elahi has obtained his Ph.D. degree in Computer Science and since then, he has published more than 80 peer-reviewed journal and conference publications. His current #citation is 2300+ and his H-index is 22. His research has been mainly focused on AI, Data Science, and Cognitive Science, with an emphasis on their potential industrial applications such as on Recommender Systems. He has also co-invented and co-coowned an AI-related US-patent. Mehdi Elahi has been involved in the authorship of several EU grant proposals such as the large-scale grants, recently funded with a budget of 30 Million Euro, where he will serve as WP Leader for 8 years. Before that, he has received prestigious research credits from giant IT industries (i.e., Amazon and Google). His research findings have been published in some of the most prestigious reference literature of the field (e.g., Recommender Systems Handbook). One of his journal articles has been the second most cited paper of a top Elsevier journal. He has organized International Data Challenges together with top Companies (i.e., Spotify and XING).

Authorized licensed use limited to: UNIVERSITY OF BIRMINGHAM. Downloaded on May 27,2023 at 10:03:14 UTC from IEEE Xplore. Restrictions apply.
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.