Information Sciences 496 (2019) 464–484

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

An efficient manifold regularized sparse non-negative matrix factorization model for large-scale recommender systems on GPUs

Hao Li[a], Keqin Li[a,b], Jiyao An[a], Weihua Zheng[a,c], Kenli Li[a,*]

[a] *School of Information Science and Engineering, National Supercomputing Center in Changsha, Hunan University, Changsha, 410082, China*
[b] *Department of Computer Science, State University of New York, New Paltz, NY 12561, USA*
[c] *College of Electrical and Information Engineering, Hunan University of Technology, Zhuzhou, Changsha 412007, China*

A R T I C L E   I N F O

Article history:
Received 31 January 2018
Revised 1 July 2018
Accepted 25 July 2018
Available online 27 July 2018

Keywords:
Collaborative filtering recommender systems
Data mining
Euclidean distance and KL-divergence
GPU parallelization
Manifold regularization
Non-negative matrix factorization

A B S T R A C T

Non-negative Matrix Factorization (NMF) plays an important role in many data mining applications for low-rank representation and analysis. Due to the sparsity that is caused by missing information in many high-dimension scenes, e.g., social networks or recommender systems, NMF cannot mine a more accurate representation from the explicit information. Manifold learning can incorporate the intrinsic geometry of the data, which is combined with a neighborhood with implicit information. Thus, manifold-regularized NMF (MNMF) can realize a more compact representation for the sparse data. However, MNMF suffers from (a) the forming of large-scale Laplacian matrices, (b) frequent large-scale matrix manipulation, and (c) the involved *K*-nearest neighbor points, which will result in the over-writing problem in parallelization. To address these issues, a single-thread-based MNMF model is proposed on two types of divergence, i.e., Euclidean distance and Kullback–Leibler (KL) divergence, which depends only on the involved feature-tuples' multiplication and summation and can avoid large-scale matrix manipulation. Furthermore, this model can remove the dependence among the feature vectors with fine-grain parallelization inherence. On that basis, a CUDA parallelization MNMF (CUMNMF) is presented on GPU computing. From the experimental results, CUMNMF achieves a 20X speedup compared with MNMF, as well as a lower time complexity and space requirement.

1. Introduction

With the rapid expansion of the scale of the Internet, it is difficult to select the anticipated data and information out of the tremendous number of bytes. Recommender systems can assist people in avoiding this dilemma, which is inundated with choices. Collaborative filtering (CF), as one of the most appealing methods in recommender systems, models the historical user behaviors on items and does not rely on domain knowledge [18]. The main role of CF is to predict the user with regard to unconcerned data or missing data. Due to the low-rank representation and non-negativity [34], Non-negative

* Corresponding author.
E-mail addresses: lihao123@hnu.edu.cn (H. Li), lik@newpaltz.edu (K. Li), jt_anbob@hnu.edu.cn (J. An), zhengdavid@hnu.edu.cn (W. Zheng), lkl@hnu.edu.cn (K. Li).

https://doi.org/10.1016/j.ins.2018.07.060
0020-0255/© 2018 Published by Elsevier Inc.

Matrix Factorization (NMF) is an ubiquitously used approach in signal processing, data mining and machine learning for many practical big data applications. As a dimensionality reduction technique, NMF can map both the users and items into the same latent feature space with non-negative constraints [17]. Thus, NMF is widely used in CF recommender systems' communities via low-rank representation. NMF considers a low-rank linear combination of a set of basis vectors [17,18,34], and trains the user and item latent factor matrices by explicit information, e.g., rated scores, clicks, and purchase records. However, data sparsity, due to having a finite concerned item set only, makes it difficult for NMF to provide more accurate estimations [17,18,34].

High-dimension data manifold structure (geometrical information) has drawn wide attention, and can improve the accuracy in many data mining and machine learning applications, i.e., image clustering, pattern recognition, and more, [5,7,9,10,23,32,35,43]. The manifold structure of data provides an utility for implicit information, owing to considering the neighborhood point relations in the intrinsic Riemannian structure [5,7,9,10,23,32,35,43]. In CF communities, manifold regularized NMF (MNMF) constructs the geometrical information and cooperates with the nearest *K*-neighbor graph information of the user and item space [9,35]. MNMF considers the neighborhood information as prior information, which can improve the accuracy of low-rank approximations [5,7,9,10,23,32,35,43]. However, in CF recommender systems, on the one hand, MNMF considers only the Gaussian probabilistic distribution as the Euclidean distance approximation [9,35], and the others are lacking; on the other hand, there are disadvantages that prevent MNMF from obtaining large-scale applications in GPU, i.e., large-scale Laplacian matrices, frequent large-scale matrix manipulation, and the parallelization over-writing problem caused by the involved computation with the K-nearest neighbor points.

With the rapid computational power improvements in graphics processing hardware, the GPU has become a widely used processor in many data-intensive applications, i.e., matrix manipulation and others. *Compute Unified Device Architecture* (CUDA) makes GPU efficient at harnessing the computational power of the GPU; at the same time, CUDA has many mature libraries for matrix processing. Currently, MNMF focuses only on small-scale datasets for clustering [5,7,9,10,23,32,35,43], i.e., COLT20 ($1,440 \times 1,024$), PIE ($2,856 \times 1,024$), and a GPU can compute the updating rule of the MNMF by matrix kernels in CUDA. However, this processing approach cannot be applied in large-scale problems directly, because of following reasons: 1) the formation of intermediate matrices cannot be manipulated on a GPU, i.e., the Netflix dataset (intermediate matrices, $480,190 \times 17,770$, $480,190 \times 480,190$, and $17,770 \times 17,770$); 2) the over-writing problem, which results from the dependence among the feature vectors, preventing GPU from obtaining high performance computing. To solve the GPU computing obstacles, a single-thread-based MNMF model is proposed. This model transforms the whole feature matrix manipulation into the involved feature vectors' multiplication and elements summation, and can remove the dependence among the feature vectors; at the same time, it holds the consecutive access characteristics of the feature elements within a feature vector.

On that basis, CUDA parallelization for single-thread-based MNMF (CUMNMF) is proposed on a GPU. The update process of all the feature vectors is independent, and it can be solved by CUDA thread blocks in parallel. The basic update unit of a feature vector is a feature element within the feature vector, and those feature elements within a feature vector are dependent. The update process of all feature elements within a feature vector can be solved by shared memory, and a CUDA thread updates a feature element. More details about this thread scheduling strategy are that the successive threads within a thread block update the successive feature elements within a feature vector. This strategy ensures aligned access of the warp threads and reduces the access latency on the global memory. To the best of our knowledge, CUMNMF is the first work that is proposed to solve the high memory overhead and over-writing problems of the MNMF on a GPU. Our main contributions in this paper are as follows:

- MNMF on Euclidean distance (MNMF$_{Eu}$) is extended to Kullback–Leibler (KL) divergence (MNMF$_{KL}$) for the CF problem, which can solve both the Gaussian and the Poisson probabilistic distribution maximization problems.
- Single-thread-based MNMF is derived, which involves the needed feature vectors' multiplication and elements summation; and at the same time, single-thread-based MNMF holds the consecutive access characteristic on the feature elements within a feature vector. Thus, single-thread-based MNMF can remove the dependence among the feature vectors, and has fine-grained parallelization inherence.
- The fine-grained parallelization inherence of single-thread-based MNMF ensures high performance on a GPU for CUDA parallelization. Thus, the CUMNMF is proposed for GPU computing, including CUMNMF on the Euclidean distance (CUMNMF$_{Eu}$) and Kullback–Leibler (KL) divergence (CUMNMF$_{KL}$).

The remainder of this paper is organized as follows. Sections 2 and 3 describe the related work, and preliminaries, respectively. Section 4 introduces the single-thread-based MNMF model. Section 4 reports the CUMNMF. Section 6 is devoted to the experimental results, i.e., parameter selection, convergence analysis, and the efficiency and effectiveness of the CUMNMF. Finally, Section 7 presents the study's conclusions and a brief discussion of future work.

2. Related work

NMF can represent the learned factors as real meanings, due to the natural representation of two arrays of data as a linear combination of non-negative feature vectors. Thus, NMF has become a useful dimension reduction and feature representation tool to analyze high-dimensional data, and it plays an important role in machine learning [33], anomaly detection for text data[14], community detection [45], accurate recovery in missing data network [37,38], collaborative prediction and recommender systems [20,31,40,42,46,47]. More recently, there many articles in the literature have appeared that are devoted to the application and theory of NMF. Liu et al. [25] proposed large-cone (LCNMF) to obtain an efficient local solution for NMF. Li et al. [21] proposed robust collaborative non-negative matrix factorization (R-CoNMF) for remotely sensed hyper-spectral un-mixing. For the CF problem, MF can solve several limitations of the neighbor-based approach, e.g., sparsity, scalability, and synonymy, which was suggested by Sarwar et al. [30]. He et al. [12] proposed online implicit feedback recommendations based on MF. Non-negativity of feature vectors highlights the users' interests and community tendencies [41]. The fulfilling of non-negativity is accomplished by adapting the rescaled learning-rate. Thereby, the negative components can be cancelled out, and the non-negativity components remain during the update process. However, NMF is based on a linear combination of basis feature vectors in a low-rank subspace, and it does not consider the geometric structure. MNMF can make the local geometry cooperate with a low-rank representation, and it can consider the data neighborhood information [5,7,9,10,23,32,35]. Hence, the dependence among the feature vectors and the intermediate matrices are brought into the updating rule of the MNMF [5,7,9,10,23,32,35]. In the CF problem, the GPU cannot afford the overhead of the intermediate data. Meanwhile, manifold learning in the CF problem considers only data with a Gaussian probabilistic distribution [5,7,9,10,23,32,35].

There are some studies on accelerating NMF in parallel and distributed platforms. More recently, Wu et al. [36] developed GPU accelerating NMF for hyperspectral un-mixing, and the target matrix of the hyperspectral image is a dense matrix. Kannan et al. [13] proposed an HPC-NMF distributed computing model for large-scale dense NMF. Due to non-linear communications cost, the HPC-NMF has weak scalability. Mejía-Roa et al. [26] proposed Multi-GPU NMF (NMF-mGPU) to extract the biological inherent meaning out of the massive experimental information. NMF-mGPU requires each GPU device to be a full copy of both factors. Furthermore, those data copies must be performed through a CPU. To solve the low efficiency memory exchanges that involve inter-GPUs, an automatic multi-GPU partitioning model was proposed by Tal et al. [2], which can improve the memory access efficiency of the NMF-mGPU. The updating process of dense NMF shares the same Hermitian matrix. However, the intermediate Hermitian matrices are different in the sparse NMF, due to the required feature vectors following the sparsity pattern of the sparse rating matrix rather than the pattern of the whole feature matrices in the updating process, which increases the large amount of computational overhead. There are some studies on accelerating sparse NMF on a distributed platform by MPI [13], and on a Cloud platform by MapReduce [24]. However, the computational strategies of the two studies are unsuited to GPUs, and the two studies have large communications overhead. Thus, there is a lack of sparse NMF for large-scale data mining applications, i.e., CF recommender systems, and so on.

General purpose computing on a GPU and listed GPU computing applications, e.g., game physics and computational biophysics, has been introduced by Owens et al. [28]. Pratx and Xing [29] reviewed GPU computing applications in medical physics, including three areas: dose calculation, treatment plan optimization, and image processing. There are some works on analyzing and predicting GPU performance. Guo et al. [11] reported a performance modeling and optimization analysis tool that can provide optimal Sparse Matrix-Vector Multiplication (SPMV) solutions based on CSR, ELL, COO, and HYB formats. To improve the performance of SPMV, Li et al. [22] considered the probability of the row non-zero element probability distribution to efficiently use four sparse matrix storage formats. In the CF recommender systems, Gao et al. [8] proposed a item-based approach for the CF recommender systems on GPU. Kato and Hosino [15] proposed singular value decomposition (SVD) based CF on GPU. However, this approach is not suitable for large-scale CF problems. To solve the problem of having a non-even distribution in a sparse matrix, which can lead to a load imbalance on GPUs and multi-core CPUs, Yang et al. [39] proposed a probability-based partition strategy. Some key problems for data mining are proposed on GPUs and multi-core, i.e., memory optimization, high efficient algorithms parallelization designing, and more [3,4].

3. Preliminaries

In this section, we present the related notations with regard to NMF, in Section 3.1. The preliminaries of NMF and MNMF are reported in Sections 3.2 and 3.3, respectively. GPU computing is presented in Section 3.4.

3.1. Notation

To start this section, we introduce some notations. We denote matrices by uppercase letters and vectors by bold-faced lowercase letters. Let $V \in \mathbb{R}_+^{m\times n}$ be a sparse rating matrix, where m and n are the numbers of users and items, respectively. $W \in \mathbb{R}_+^{m\times r}$ and $H \in \mathbb{R}_+^{n\times r}$ are denoted as the user and item feature matrix, respectively. Ω and $\overline{\Omega}$ are denoted as the set of indices of non-zeros in V (row oriented) and the set of indices of non-zeros in V (column oriented), respectively. We use Ω_i and $\overline{\Omega}_j$ to denote the column indices and row indices in the ith row and jth column, respectively. We denote the ith row of W by $\mathbf{w}_i$ and the kth column of W by $\overline{\mathbf{w}}_k$. We denote the jth row of H by $\mathbf{h}_j$ and the kth column of H by $\overline{\mathbf{h}}_k$. The kth elements of $\mathbf{w}_i$ and $\mathbf{h}_j$ are represented by $w_{i,\,k}$ and $h_{j,\,k}$, respectively. The graph Laplacian matrix about the row of V and column of V are denoted by L and $\overline{L}$, respectively. In CF MF recommenders, most of the algorithms in the MF literature take WH^T as the low-rank approximation matrix to predict the non-rated or zero entries of the sparse rating matrix V, and the approximation process is accomplished by minimizing the distance function to measure the degree of approximation [12,14,20,21,25,30,31,33,37,38,40–42,45–47]. To keep the non-negativity of the two feature matrices, NMF can conduct this factorization process into using non-negative constraints, i.e., $W, H \geq 0$.

Definition 1 (NMF)**.** We are given two matrices V and $\widetilde{V} \in \mathbb{R}_{+}^{m\times n}$ and a divergence function $\mathcal{D}(V\|\widetilde{V})$, which is to evaluate the distance between the two matrices. The NMF problem is to find $W \in \mathbb{R}_{+}^{m\times r}$ and $H \in \mathbb{R}_{+}^{n\times r}$ such that $\mathcal{D}(V\|\widetilde{V})$ is minimized, where $\widetilde{V} = WH^T$.

3.2. A brief review of NMF

The distance between V and WH^T evaluated by the Euclidean distance $\mathcal{D}_{Eu}(V||\widetilde{V})$ (NMF$_{Eu}$) and KL-divergence $\mathcal{D}_{KL}(V||\widetilde{V})$ (NMF$_{KL}$) are defined as

$$\begin{aligned}\arg\min_{W,H} d_{Eu} &= \mathcal{D}_{Eu}(V||\widetilde{V})\\ &= \|V - \widetilde{V}\|^2\\ &= \sum_{(i,j)\in\Omega} (v_{i,j} - \widetilde{v}_{i,j})^2, s.t. W, H \geq 0,\end{aligned} \tag{1}$$

$$\begin{aligned}\arg\min_{W,H} d_{KL} &= \mathcal{D}_{KL}(V||\widetilde{V})\\ &= \sum_{(i,j)\in\Omega} \left(v_{i,j}\log(\frac{v_{i,j}}{\widetilde{v}_{i,j}}) - v_{i,j} + \widetilde{v}_{i,j}\right), s.t. W, H \geq 0,\end{aligned} \tag{2}$$

respectively [20,31,40,42,46,47]. Problems (1) and (2) are non-convex optimization problems, and the most common solution method is alternative minimization [20,31,40,42,46,47]. Applying gradient decent to problems (1) and (2), the updating rules for the NMF$_{Eu}$ and NMF$_{KL}$ are obtained as follows:

$$\begin{cases} w_{i,k} \leftarrow w_{i,k} + \eta_{i,k}\big((VH)_{i,k} - (WH^TH)_{i,k}\big);\\ h_{j,k} \leftarrow h_{j,k} + \overline{\eta}_{j,k}\big((V^TW)_{j,k} - (HW^TW)_{j,k}\big),\end{cases} \tag{3}$$

$$\begin{cases} w_{i,k} \leftarrow w_{i,k} + \eta_{i,k}\big((\frac{V}{WH^T}H)_{i,k} - \sum_i h_{i,k}\big);\\ h_{j,k} \leftarrow h_{j,k} + \overline{\eta}_{j,k}\big(((\frac{V}{WH^T})^TW)_{j,k} - \sum_j w_{j,k}\big),\end{cases} \tag{4}$$

respectively, where $\eta_{i,\,k}$ and $\overline{\eta}_{j,k}$ are the corresponding learning rate for $w_{i,\,k}$ and $h_{j,\,k}$, respectively. With initially non-negative W and H, NMF diagonally manipulates the rescaled learning rate to maintain the non-negativity of W and H for the NMF$_{Eu}$ and NMF$_{KL}$, which are as follows:

$$\mathcal{D}_{Eu}(V||\widetilde{V})\begin{cases} \eta_{i,k} \leftarrow \dfrac{w_{i,k}}{(WH^TH)_{i,k}};\\ \eta_{j,k} \leftarrow \dfrac{h_{j,k}}{(HW^TW)_{j,k}},\end{cases} \tag{5}$$

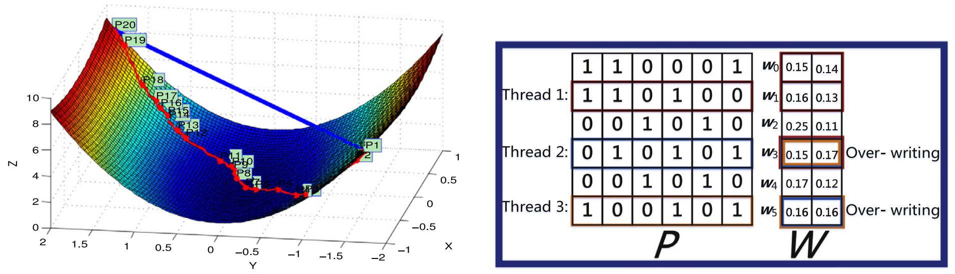
$$\mathcal{D}_{KL}(V||\widetilde{V})\begin{cases} \eta_{i,k} \leftarrow \dfrac{w_{i,k}}{\sum_i h_{i,k}};\\ \eta_{j,k} \leftarrow \dfrac{h_{j,k}}{\sum_j w_{j,k}},\end{cases} \tag{6}$$

respectively. Applying the rescaled learning rate (5) and (6) to (3) and (4), respectively, the negative components $\{-(WH^TH)_{i,k}, -HW^TW)_{j,k}\}$ in (3) and $\{-\sum_i h_{i,k}, -\sum_j w_{j,k}\}$ in (4) can be cancelled out. Thereby, the learning process is reformulated into

$$\mathcal{D}_{Eu}(V||\widetilde{V})\begin{cases} w_{i,k} = \dfrac{w_{i,k}}{(WH^TH)_{i,k}};\\ h_{j,k} = \dfrac{h_{j,k}}{(HW^TW)_{j,k}},\end{cases} \tag{7}$$

$$\mathcal{D}_{KL}(V||\widetilde{V})\begin{cases} w_{i,k} = \dfrac{(\frac{V}{WH^T}H)_{i,k}}{\sum_i h_{i,k}};\\ h_{j,k} = \dfrac{((\frac{V}{WH^T})^TW)_{j,k}}{\sum_j w_{j,k}},\end{cases} \tag{8}$$

respectively. The convergence proof of updating rules (7) and (8) are proved in [19]. We observe that the update rules (7) and (8) are designed for dense NMF. However, in the CF problems, the sparse rating matrix V has high sparsity with a non-even



(a) Toy example of Manifold learning

(b) Over-writing problem resulting from dependence among the feature vectors

Fig. 1. Manifold learning and the corresponding overwriting problem on MNMF: (a) a toy example for manifold learning, and (b) the overwriting problem in parallelization.

distribution non-zero entries. Consequently, the updating rules (7) and (8) should be revised correspondingly. Some prior studies [16,44] have developed NMF-based weighted low-rank approximation for the CF problems. Zhang et al. [44] proposed the weighted NMF (WNMF) and introduced an indicator matrix into the updating process to fit the CF problems. The corresponding sparse NMF updating rules for NMF$_{Eu}$ and NMF$_{KL}$ are given by

$$\mathcal{D}_{Eu}(V\|\widetilde{V})\begin{cases} w_{i,k} \leftarrow w_{i,k}\dfrac{(G\circ VH)_{i,k}}{\big((G\circ WH^T)H\big)_{i,k}}; \\ h_{j,k} \leftarrow h_{j,k}\dfrac{\big((G\circ V)^T W\big)_{j,k}}{\big((G\circ WH^T)^T W\big)_{j,k}}, \end{cases} \tag{9}$$

$$\mathcal{D}_{KL}(V\|\widetilde{V})\begin{cases} w_{i,k} \leftarrow w_{i,k}\dfrac{\big(G\circ \frac{V}{WH^T}H\big)_{i,k}}{(GH)_{i,k}}; \\ h_{j,k} \leftarrow h_{j,k}\dfrac{\big((G\circ \frac{V}{WH^T})^T W\big)_{j,k}}{(G^T W)_{j,k}}, \end{cases} \tag{10}$$

respectively, where the symbol $\circ$ denotes the Hadamard product (element-wise matrix multiplication), and $G \in \mathbb{R}^{m\times n}$, of which the element is equal to 1 if the corresponding entry in V is known and 0 otherwise.

3.3. Manifold regularized non-negative matrix factorization

NMF can approximate the original samples $V = \{\overline{v}_0, \cdots, \overline{v}_{n-1}\}$, $\overline{v}_k \in \mathbb{R}_+^m$, $k \in \{0, \cdots, n-1\}$ as a linear combination of basis vectors in a low-dimensional subspace. V can be represented by a non-negative matrix $W \in \mathbb{R}_+^{m\times r}$ and $H \in \mathbb{R}_+^{n\times r}$, in which $H \in \mathbb{R}_+^{n\times r}$ is the coefficient matrix. Thus, it can be deemed to be a function $f(\overline{\mathbf{v}}_j) = \mathbf{h}_j$, subject to $\overline{\mathbf{v}}_j = W\mathbf{h}_j^T$, to preserve the local geometry of the distribution of samples V. Suppose that V is sample from a probability distribution $\mathcal{P}_V$ on a manifold $\mathcal{M}$ that is embedded by penalizing the gradient $\nabla_{\mathcal{M}} f$ along the manifold $\mathcal{M}$

$$\int_{V\in\mathcal{M}} \|\nabla_{\mathcal{M}} f\|^2 d\mathcal{P}_V, \tag{11}$$

where the integral is taken over the probability distribution $\mathcal{P}_V$. However, both the manifold $\mathcal{M}$ and marginal distribution $\mathcal{P}_V$ are unknown in practice, and thus, we use an empirical estimate of the penalty (11). The prior work [5] has demonstrated that the local geometric structure can be approximated by some data points along the manifold $\mathcal{M}$. We consider a graph with N vertices, where each vertex corresponds to a data point. For each data point $\overline{\mathbf{v}}_j$, we find its K nearest neighbors and place edges between $\overline{\mathbf{v}}_j$ and its neighbors. Fig. 1 (a) illustrates a toy example for manifold learning. As Fig. 1 (a) shown, there are 20 points scattered among the manifold $\mathcal{M}$, i.e., points $\{P_1, \cdots, P_{20}\}$. The distance between points P_1 and P_{20} along the manifold $\mathcal{M}$ can be approximated by the length of the red line, which is linked by the 20 points. Note that the blue line (the direct distance between points P_1 and P_{20} in Euclidean space rather than in the manifold $\mathcal{M}$) in Fig. 1 (a) presents an obvious difference from the red line (the space in manifold $\mathcal{M}$). The distance along the manifold $\mathcal{M}$ between two points is measured by the Euclidean distance, which is adopted by Guan et al. [10]. Thus, in CF communities, the dependence in the user and item latent space can provide implicit information.

The Graph Laplacian of samples V can be used for the regularization in manifold learning. The main idea is to construct a weight matrix P whose vertexes correspond to the samples, and the pairwise edge weight $p_{i,j}$ reflects the approximation degree of two samples. There are many strategies to define the row weight matrix P on the graph, as follows [5,7,9,10,23,32,35]:

- **0-1 weighting**. $p_{i,j} = 1$ if and only if nodes i and j are connected by an edge.
- **Heat kernel weighting**. if nodes i and j are connected, then place $p_{i,j} = e^{-\frac{\|\mathbf{v}_i - \mathbf{v}_j\|^2}{\sigma}}$. The heat kernel has an intrinsic connection to the Laplace Beltrami operator on differentiable functions on a manifold.
- **Dot-product weighting**. If nodes i and j are connected, then place $p_{i,j} = \mathbf{v}_i^T \mathbf{v}_j$.

The computational strategy in [9] is adopted in our work. Correspondingly, $\overline{P}$ is the column weight matrix . Then, the threshold to determine the K value is introduced to keep the sparsity and symmetry of the matrices P and $\overline{P}$. Define $L = Q - P$, where Q is a diagonal matrix whose entries are row $q_{i,i} = \sum_j p_{i,j}$. Similarly, define $\overline{L} = \overline{Q} - \overline{P}$, where $\overline{Q}$ is a diagonal matrix whose entries are row $\overline{q}_{i,i} = \sum_j \overline{p}_{i,j}$. L and $\overline{L}$ are called graph Laplacian matrices [5,7,9,10,23,32,35], which is a discrete approximation to the Laplace–Beltrami operator. Thus, the discrete approximation can be computed as follows:

$$\begin{aligned}
\mathcal{R}_k &= \frac{1}{2}\sum_{i=0}^{m-1}\sum_{j=0}^{m-1}\left(f_k(\mathbf{v}_i) - f_k(\mathbf{v}_j)\right)^2 p_{i,j} \\
&= \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} f_k(\mathbf{v}_i)^2 p_{i,j} - \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} f_k(\mathbf{v}_i) f_k(\mathbf{v}_j) p_{i,j} \\
&= \sum_{i=0}^{m-1} w_{i,k}^2 q_{i,i} - \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} w_{i,k} w_{j,k} p_{i,j} \\
&= \overline{\mathbf{w}}_k^T Q \overline{\mathbf{w}}_k - \overline{\mathbf{w}}_k^T P \overline{\mathbf{w}}_k \\
&= \overline{\mathbf{w}}_k^T L \overline{\mathbf{w}}_k,
\end{aligned} \tag{12}$$

and similarly,

$$\begin{aligned}
\overline{\mathcal{R}}_k &= \frac{1}{2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\left(f_k(\overline{\mathbf{v}}_i) - f_k(\overline{\mathbf{v}}_j)\right)^2 \overline{p}_{i,j} \\
&= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} f_k(\overline{\mathbf{v}}_i)^2 \overline{p}_{i,j} - \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} f_k(\overline{\mathbf{v}}_i) f_k(\overline{\mathbf{v}}_j) \overline{p}_{i,j} \\
&= \sum_{i=0}^{n-1} h_{i,k}^2 \overline{q}_{i,i} - \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} h_{i,k} h_{j,k} \overline{p}_{i,j} \\
&= \overline{\mathbf{h}}_k^T \overline{Q} \overline{\mathbf{h}}_k - \overline{\mathbf{h}}_k^T \overline{P} \overline{\mathbf{h}}_k \\
&= \overline{\mathbf{h}}_k^T \overline{L} \overline{\mathbf{h}}_k,
\end{aligned} \tag{13}$$

where $\mathcal{R}_k$ and $\overline{\mathcal{R}}_k$ can be used to measure the smoothness of the mapping function f_k along the geodesics in the intrinsic geometry of a data set. By minimizing $\mathcal{R}_k$ and $\overline{\mathcal{R}}_k$, we obtain a mapping function f_k that is sufficiently smooth on the data manifold. An intuitive explanation of minimizing $\mathcal{R}_k$ is that if two data points $\mathbf{v}_i$ and $\mathbf{v}_j$ are close, then $f_k(\mathbf{v}_i)$ and $f_k(\mathbf{v}_j)$ are similar to each other. Manifold regularization can incorporate the $\mathcal{R}_k$ and $\overline{\mathcal{R}}_k$ terms and minimize the following two functions (Euclidean distance, KL-divergence):

$$\begin{aligned}
\mathcal{O}_{Eu} &= \mathcal{D}_{Eu}(V||\widetilde{V}) + \lambda_W \sum_{k=0}^{r-1} \mathcal{R}_k + \lambda_H \sum_{k=0}^{r-1} \overline{\mathcal{R}}_k \\
&= \|V - WH^T\|_2^2 + \lambda_W Tr(W^T L W) + \lambda_H Tr(H^T \overline{L} H),
\end{aligned} \tag{14}$$

and

$$\begin{aligned}
\mathcal{O}_{KL} &= \mathcal{D}_{KL}(V||\widetilde{V}) + \lambda_W \sum_{k=0}^{r-1} \mathcal{R}_k + \lambda_H \sum_{k=0}^{r-1} \overline{\mathcal{R}}_k \\
&= \sum_{(i,j)\in\Omega}\left(v_{i,j}\log(\frac{v_{i,j}}{\widetilde{v}_{i,j}}) - v_{i,j} + \widetilde{v}_{i,j}\right) + \lambda_W Tr(W^T L W) + \lambda_H Tr(H^T \overline{L} H),
\end{aligned} \tag{15}$$

respectively, where $Tr(\bullet)$ is the trace norm, which is the summation of the diagonal elements in a square matrix. The updating rules are presented in Table 1, and the convergence proof is presented in [5,7,9,10,23,32,35].

The updating rules in Table 1 can solve the optimization problems (1) and (2); however, there exist some extra computation, the intermediate data explosion problems, as follows:

- Sparse matrix G: G is used to distinguish a zero and non-zero value, and the space complexity of G is $O(|\Omega|)$.

Table 1
Updating rules of two divergence types MNMF, i.e., Euclidean distance, KL-divergence.

Divergence types	Euclidean distance $\mathcal{D}_{Eu}(V\|\|\widetilde{V})$ $\sum_{(i,j)\in\Omega}\left(v_{i,j}-\widetilde{v}_{i,j}\right)^2$		KL-divergence $\mathcal{D}_{KL}(V\|\|\widetilde{V})$ $\sum_{(i,j)\in\Omega}\left(v_{i,j}\log(\frac{v_{i,j}}{\widetilde{v}_{i,j}})-v_{i,j}+\widetilde{v}_{i,j}\right)$	
Manifold Regularized Update Rules	$W \leftarrow W \circ \frac{G\circ VH+\lambda_W PW}{(G\circ WH^T)H+\lambda_W QW}$	(16)	$W \leftarrow W \circ \frac{G\circ \frac{V}{WH^T}H+\lambda_W PW}{GH+\lambda_W QW}$	(18)
	$H \leftarrow H \circ \frac{(G\circ V)^T W+\lambda_H \overline{P}H}{(G\circ WH^T)^T W+\lambda_H \overline{Q}H}$	(17)	$H \leftarrow H \circ \frac{(G\circ \frac{V}{WH^T})^T W+\lambda_H \overline{P}H}{G^T W+\lambda_H \overline{Q}H}$	(19)

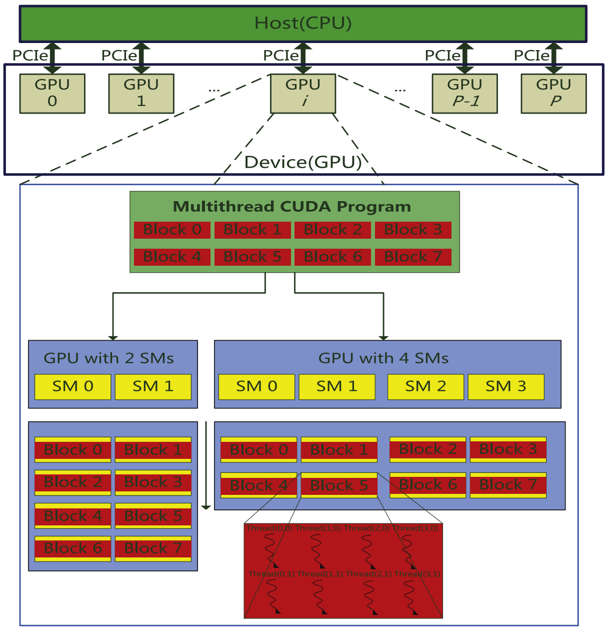


Fig. 2. Multi-GPU and CUDA kernel and thread batching.

- Intermediate matrix WH^T: The computing and space complexity of WH^T are $O(mnr)$ and $O(mn)$, respectively.
- Intermediate matrices PW, QW, $\overline{P}H$, and $\overline{Q}H$: The space complexity of $\{PW, QW\}$ and $\{\overline{P}H, \overline{Q}H\}$ are $O(m^2)$ and $O(n^2)$, respectively.

We observe that there are many highly efficient dense and sparse matrix multiplication kernels in CUDA. Due to the aforementioned high memory overhead problems, the GPU cannot afford the memory overhead. On the other hand, the Laplacian matrices $\{P, Q\}$, $\{\overline{P}, \overline{Q}\}$, and intermediate matrices $\{PW, QW\}$, $\{\overline{P}H, \overline{Q}H\}$ can take the data geometric structure into NMF. However, they bring in the dependence among the feature vectors $\{\mathbf{w}_i | i \in \{0, 1, \cdots, m-1\}\}$, and $\{\mathbf{h}_j | i \in \{0, 1, \cdots, n-1\}\}$, respectively. A toy example of the over-writing problem is illustrated in Fig. 1 (b). As Fig. 1 (b) shown, the threads $\{1, 2, 3\}$ update $\{\mathbf{w}_1, \mathbf{w}_3, \mathbf{w}_5\}$ and fetch $\{\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_3, \mathbf{w}_5\}$ simultaneously. This step results in the over-writing problem on $\{\mathbf{w}_3, \mathbf{w}_5\}$.

3.4. GPU based computing

Multiple GPUs are connected to the *host* by *peripheral communication interconnect express* (PCIe) in Fig. 2. CUDA can perform computation on the device and data transfers between the *device* and *host* concurrently. More state-of-the-art GPUs, i.e., K20m, K40c, and P100, have two or more copy engines, which can operate data transfers to and from a device and transfer among the devices concurrently. CUDA provides synchronization instructions, which can ensure the correct execution on multi-GPUs, and each GPU of the multi-GPU has a consumer/producer relationship. Memory copies between two different devices can be performed after the instruction of CUDA peer-to-peer memory access has been enabled.

The GPU resides on a *device*, and a GPU consists of many *stream multiprocessors* (SMs). Each SM contains a number of *stream processors* (SPs). CUDA is a programming interface that can enable a GPU to be compatible with various programming languages and applications. In CUDA, *kernels* are functions that are executed on GPU. A kernel function is executed by a batch of threads. The batch of threads is organized as a grid of thread blocks. The thread blocks map to SMs. As shown in Fig. 2, the greater the number of SMs in a GPU, the higher the degree of parallelism the GPU has. Thus, a GPU with more SMs will execute a CUDA program in less time than a GPU with fewer SMs. Threads in a block are organized into small groups of 32 called *warps*, for execution on the processors, and the warps are implicitly synchronous; however, threads in different blocks are asynchronous. CUDA threads access data from multiple memory hierarchies. Each thread has private local memory, and each thread block has shared memory that is visible to all threads within the thread block, and all threads can access global memory.

4. A single-thread-based MNMF approach

In this section, the derivations of the single-thread-based MNMF$_{Eu}$ and NMF$_{KL}$ are shown in Section 4.1. In Section 4.2, the space and time complexity are presented.

4.1. Single-thread-based MNMF for CF problems

4.1.1. Euclidean distance (MNMF$_{\mathrm{Eu}}$)

For MNMF, the Euclidean distance between the sparse rating matrix V and low-rank approximation WH^T is given by

$$\begin{aligned}
&\underset{V\leftrightarrow W,H}{\varepsilon} \\
&= \mathcal{O}_{Eu} \\
&= \|\mathcal{P}_\Omega(V) - \mathcal{P}_\Omega(\widetilde{V})\|^2 + \lambda_H Tr(H^T\overline{L}H) + \lambda_W Tr(W^T LW) \\
&= \sum_{(i,j)\in\Omega} (v_{i,j} - \widetilde{v}_{i,j})^2 + \sum_{k=0}^{r-1}\left(\lambda_H(\sum_{j=0}^{n-1} h_{j,k}^2 \overline{q}_{j,j} - \sum_{j=0}^{n-1}\sum_{l=0}^{n-1} h_{j,k}h_{l,k}\overline{p}_{j,l}) + \lambda_W(\sum_{i=0}^{m-1} w_{i,k}^2 q_{i,i} - \sum_{i=0}^{m-1}\sum_{l=0}^{m-1} w_{i,k}w_{l,k}p_{i,l})\right) \\
&= \sum_{(i,j)\in\Omega} (v_{i,j} - \mathbf{w}_i\mathbf{h}_j^T)^2 + \sum_{k=0}^{r-1}\left(\lambda_H(\sum_{j=0}^{n-1} h_{j,k}^2 \overline{q}_{j,j} - \sum_{j=0}^{n-1}\sum_{l=0}^{n-1} h_{j,k}h_{l,k}\overline{p}_{j,l}) + \lambda_W(\sum_{i=0}^{m-1} w_{i,k}^2 q_{i,i} - \sum_{i=0}^{m-1}\sum_{l=0}^{m-1} w_{i,k}w_{l,k}p_{i,l})\right) \\
&= \sum_{(i,j)\in\Omega} \left(v_{i,j} - \sum_{k=0}^{r-1} w_{i,k}h_{j,k}\right)^2 + \sum_{k=0}^{r-1}\left(\lambda_H(\sum_{j=0}^{n-1} h_{j,k}^2 \overline{q}_{j,j} - \sum_{j=0}^{n-1}\sum_{l=0}^{n-1} h_{j,k}h_{l,k}\overline{p}_{j,l}) + \lambda_W(\sum_{i=0}^{m-1} w_{i,k}^2 q_{i,i} - \sum_{i=0}^{m-1}\sum_{l=0}^{m-1} w_{i,k}w_{l,k}p_{i,l})\right),
\end{aligned} \tag{20}$$

where $\mathcal{P}_\Omega$ is the projection operator on the index set Ω. We note that in (20), only the entries in Ω and their approximation are considered. The problem (20) can be split into many independent sub-problems, i.e., $\varepsilon = \sum_{i=0}^{m-1} \varepsilon_i$, $\varepsilon = \sum_{j=0}^{n-1} \overline{\varepsilon}_j$, where

$$\begin{cases}
\varepsilon_i = \sum_{j\in\Omega_i}\left(v_{i,j} - \sum_{k=0}^{r-1} w_{i,k}h_{j,k}\right)^2 + \sum_{k=0}^{r-1}\lambda_W\left(w_{i,k}^2 q_{i,i} - 2\sum_{l=0,l\neq i}^{m-1} w_{i,k}w_{l,k}p_{i,l} - w_{i,k}^2 p_{i,i}\right); \\
\overline{\varepsilon}_j = \sum_{j\in\overline{\Omega}_j}\left(v_{i,j} - \sum_{k=0}^{r-1} w_{i,k}h_{j,k}\right)^2 + \sum_{k=0}^{r-1}\lambda_H\left(h_{j,k}^2 \overline{q}_{j,j} - 2\sum_{l=0,l\neq j}^{n-1} h_{l,k}h_{j,k}\overline{p}_{j,l} - h_{j,k}^2 \overline{p}_{j,j}\right).
\end{cases} \tag{21}$$

The weight matrices P and $\overline{P}$ are symmetric. Hence, in (21), the involved coefficient of $\sum_{l=0,l\neq i}^{m-1} w_{i,k}w_{l,k}p_{i,l}$ and $\sum_{l=0,l\neq j}^{n-1} h_{l,k}h_{j,k}\overline{p}_{j,l}$ is 2. We observe that minimizing ε is equivalent to minimizing ε_i and $\overline{\varepsilon}_j$ alternatively. Thus, we apply gradient descent to minimize ε_i and $\overline{\varepsilon}_j$, respectively, by omitting the constant 2. Therefore, an additive update-rule to minimize ε_i and $\overline{\varepsilon}_j$ is derived as follows:

$$\begin{cases} w_{i,k} \leftarrow w_{i,k} - \eta_{i,k}\dfrac{\partial \varepsilon_i}{\partial w_{i,k}} \\ \quad = w_{i,k} + \eta_{i,k}\left(\sum\limits_{j\in\Omega_i} h_{j,k}(v_{i,j} - \sum\limits_{k=0}^{r-1} w_{i,k}h_{j,k}) + \sum\limits_{l=0}^{m-1} \lambda_W p_{i,l}w_{l,k} - \lambda_W q_{i,i}w_{i,k}\right) \\ \quad = w_{i,k} + \eta_{i,k}\left(\sum\limits_{j\in\Omega_i} h_{j,k}v_{i,j} + \sum\limits_{l=0}^{m-1} \lambda_W p_{i,l}w_{l,k} - \lambda_W q_{i,i}w_{i,k} - \sum\limits_{j\in\Omega_i} h_{j,k}\widetilde{v}_{i,j}\right); \\ h_{j,k} \leftarrow h_{j,k} - \overline{\eta}_{j,k}\dfrac{\partial \overline{\varepsilon}_j}{\partial h_{j,k}} \\ \quad = h_{j,k} + \overline{\eta}_{j,k}\left(\sum\limits_{i\in\overline{\Omega}_j} w_{i,k}(v_{i,j} - \sum\limits_{k=0}^{r-1} w_{i,k}h_{j,k}) + \sum\limits_{l=0}^{n-1} \lambda_H \overline{p}_{j,l}h_{l,k} - \lambda_H \overline{q}_{j,j}h_{j,k}\right) \\ \quad = h_{j,k} + \overline{\eta}_{j,k}\left(\sum\limits_{i\in\overline{\Omega}_j} w_{i,k}v_{i,j} + \sum\limits_{l=0}^{n-1} \lambda_H \overline{p}_{j,l}h_{l,k} - \lambda_H \overline{q}_{j,j}h_{j,k} - \sum\limits_{i\in\overline{\Omega}_j} w_{i,k}\widetilde{v}_{i,j}\right). \end{cases} \tag{22}$$

If the variable $\eta_{i,\,k}$ and $\overline{\eta}_{j,k}$ are set as

$$\begin{cases} \eta_{i,k} = \dfrac{w_{i,k}}{\sum\limits_{j\in\Omega_i} h_{j,k}\widetilde{v}_{i,j} + \lambda_W q_{i,i}w_{i,k}}; \\ \overline{\eta}_{j,k} = \dfrac{h_{j,k}}{\sum\limits_{i\in\overline{\Omega}_j} w_{i,k}\widetilde{v}_{i,j} + \lambda_H \overline{q}_{j,j}h_{j,k}}, \end{cases} \tag{23}$$

respectively. With initial non-negative feature matrices W and H, the negative items $\{-\lambda_W q_{i,i}w_{i,k} - \sum_{j\in\Omega_i} h_{j,k}\widetilde{v}_{i,j}\}$ and $\{-\lambda_H \overline{q}_{j,j}h_{j,k} - \sum_{i\in\overline{\Omega}_j} w_{i,k}\widetilde{v}_{i,j}\}$ can be cancelled out, and the updating rule of (22) is adjusted into the following updating rule:

$$\begin{cases} w_{i,k} \leftarrow w_{i,k}\dfrac{\sum\limits_{j\in\Omega_i} h_{j,k}v_{i,j} + \sum\limits_{l=0}^{m-1} \lambda_W p_{i,l}w_{l,k}}{\sum\limits_{j\in\Omega_i} h_{j,k}\widetilde{v}_{i,j} + \lambda_W q_{i,i}w_{i,k}}; \\ h_{j,k} \leftarrow h_{j,k}\dfrac{\sum\limits_{i\in\overline{\Omega}_j} w_{i,k}v_{i,j} + \sum\limits_{l=0}^{n-1} \lambda_H \overline{p}_{j,l}h_{l,k}}{\sum\limits_{i\in\overline{\Omega}_j} w_{i,k}\widetilde{v}_{i,j} + \lambda_H \overline{q}_{j,j}h_{j,k}}. \end{cases} \tag{24}$$

4.1.2. KL-divergence (MNMF$_{\mathrm{KL}}$)

The KL-divergence between the sparse matrix V and low-rank approximation WH^T is given by

$$\begin{aligned} &\underset{V\leftrightarrow WH^T}{\varepsilon} \\ &= \mathcal{O}_{KL} \\ &= \mathcal{D}_{KL}\big(\mathcal{P}_\Omega(V)||\mathcal{P}_\Omega(\widetilde{V})\big) + \frac{\lambda_H}{2}Tr(H^T\overline{L}H) + \frac{\lambda_W}{2}Tr(W^TLW) \\ &= \sum_{(i,j)\in\Omega}\left(v_{i,j}\log(\frac{v_{i,j}}{\widetilde{v}_{i,j}}) - v_{i,j} + \widetilde{v}_{i,j}\right) + \sum_{k=0}^{r-1}\left(\frac{\lambda_H}{2}\left(\sum_{i=0}^{n-1} h_{j,k}^2\overline{q}_{j,j} - \sum_{i=0}^{n-1}\sum_{l=0}^{n-1} h_{j,k}h_{l,k}\overline{p}_{l,j}\right) + \frac{\lambda_W}{2}\left(\sum_{i=0}^{m-1} w_{i,k}^2 q_{i,i} - \sum_{i=0}^{m-1}\sum_{l=0}^{m-1} w_{i,k}w_{l,k}p_{l,i}\right)\right). \end{aligned} \tag{25}$$

We note that in (25), only the entries in Ω and their approximation are taken into considerations. The problem (25) can be split into many independent sub-problems, i.e., $\varepsilon = \sum_{i=1}^{m} \varepsilon_i$, $\varepsilon = \sum_{j=1}^{n} \overline{\varepsilon}_j$, where

$$\begin{cases} \varepsilon_i &= \sum\limits_{j\in\Omega_i}\left(v_{i,j}\log(\frac{v_{i,j}}{\widetilde{v}_{i,j}}) - v_{i,j} + \widetilde{v}_{i,j}\right) + \sum\limits_{k=0}^{r-1}\frac{\lambda_W}{2}\left(w_{i,k}^2 q_{i,i} - 2\sum\limits_{l=0,l\neq i}^{m-1} w_{i,k}w_{l,k}p_{l,i} - w_{i,k}^2 p_{i,i}\right); \\ \overline{\varepsilon}_j &= \sum\limits_{j\in\overline{\Omega}_j}\left(v_{i,j}\log(\frac{v_{i,j}}{\widetilde{v}_{i,j}}) - v_{i,j} + \widetilde{v}_{i,j}\right) + \sum\limits_{k=0}^{r-1}\frac{\lambda_H}{2}\left(h_{j,k}^2 \overline{q}_{j,j} - 2\sum\limits_{l=0,l\neq j}^{n-1} h_{l,k}h_{j,k}\overline{p}_{l,j} - h_{j,k}^2 \overline{p}_{j,j}\right). \end{cases} \tag{26}$$

The weight matrices P and $\overline{P}$ are symmetric. Hence, in (26), the involved coefficient of $\sum_{l=0,l\neq i}^{m-1} w_{i,k}w_{l,k}p_{i,l}$ and $\sum_{l=0,l\neq j}^{n-1} h_{l,k}h_{j,k}\overline{p}_{j,l}$ is 2. Minimizing ε_i and $\overline{\varepsilon}_j$ alternatively is equivalent to minimize ε. Thus, we apply gradient descent to minimize ε_i and $\overline{\varepsilon}_j$, respectively, and we omit constant 2. The updating rules to minimize ε_i and $\overline{\varepsilon}_j$ are derived as follows:

$$\begin{cases} \arg\min\limits_{w_{i,k}} \varepsilon_i \\ \qquad w_{i,k} \leftarrow w_{i,k} - \eta_{i,k}\frac{\partial \varepsilon_i}{\partial w_{i,k}} \\ \qquad\qquad = w_{i,k} + \eta_{i,k}\left(\sum\limits_{j\in\Omega_i} h_{j,k}\frac{v_{i,j}}{\sum\limits_{k=0}^{r-1} w_{i,k}h_{j,k}} + \sum\limits_{l=0}^{m-1}\lambda_W p_{i,l}w_{l,k} - \lambda_W q_{i,i}w_{i,k} - \sum\limits_{j\in\Omega_i} h_{j,k}\right); \\ \arg\min\limits_{h_{j,k}} \overline{\varepsilon}_j \\ \qquad h_{j,k} \leftarrow h_{j,k} - \overline{\eta}_{j,k}\frac{\partial \overline{\varepsilon}_j}{\partial h_{j,k}} \\ \qquad\qquad = h_{j,k} + \overline{\eta}_{j,k}\left(\sum\limits_{i\in\overline{\Omega}_j} w_{i,k}\frac{v_{i,j}}{\sum\limits_{k=0}^{r-1} w_{i,k}h_{j,k}} + \sum\limits_{l=0}^{m-1}\lambda_H \overline{p}_{j,l}h_{l,k} - \lambda_H \overline{q}_{j,j}h_{j,k} - \sum\limits_{i\in\overline{\Omega}_j} w_{i,k}\right). \end{cases} \tag{27}$$

If the variable $\eta_{i,k}$ and $\overline{\eta}_{j,k}$ are set as

$$\begin{cases} \eta_{i,k} = \dfrac{w_{i,k}}{\sum\limits_{j\in\Omega_i} h_{j,k} + \lambda_W q_{i,i}w_{i,k}}; \\ \overline{\eta}_{j,k} = \dfrac{h_{j,k}}{\sum\limits_{i\in\overline{\Omega}_j} w_{i,k} + \lambda_H \overline{q}_{j,j}h_{j,k}}, \end{cases} \tag{28}$$

respectively. With initial non-negative feature matrices W and H, the negative items $\{-\lambda_W q_{i,i}w_{i,k} - \sum_{j\in\Omega_i} h_{j,k}\}$ and $\{-\lambda_H \overline{q}_{j,j}h_{j,k} - \sum_{i\in\overline{\Omega}_j} w_{i,k}\}$ can be cancelled out, and the updating rule of (27) is adjusted into the following updating rule:

$$\begin{cases} w_{i,k} \leftarrow w_{i,k}\dfrac{\sum\limits_{j\in\Omega_i} h_{j,k}\frac{v_{i,j}}{\widetilde{v}_{i,j}} + \sum\limits_{l=0}^{m-1}\lambda_W p_{i,l}w_{l,k}}{\sum\limits_{j\in\Omega_i} h_{j,k} + \lambda_W q_{i,i}w_{i,k}}; \\ h_{j,k} \leftarrow h_{j,k}\dfrac{\sum\limits_{i\in\overline{\Omega}_j} w_{i,k}\frac{v_{i,j}}{\widetilde{v}_{i,j}} + \sum\limits_{l=0}^{n-1}\lambda_H \overline{p}_{j,l}h_{l,k}}{\sum\limits_{i\in\overline{\Omega}_j} w_{i,k} + \lambda_H \overline{q}_{j,j}h_{j,k}}. \end{cases} \tag{29}$$

From the updating rules (24) and (29), we observe that the updating rules of W and H are symmetric. Thus, we consider only the updating rule of W in the following sections. Algorithm 1 describes the serial version of single-thread-based MNMF. As described in Algorithm 1, the updating rule of single-thread-based MNMF follows the sparsity pattern of the sparse rating matrix, and it can escape from forming the large-scale intermediate matrices. We observe that the intermediate matrices *Up* and *Down* make an accumulation to accomplish single-thread-based MNMF via 3 loops, i.e., the 1th loop (Lines 7–10, Algorithm 1), the 2nd loop (Lines 17, 18, Algorithm 1), and the 3rd loop (Lines 24, 25, Algorithm 1). Thus, *Up* and *Down* can

Algorithm 1 The serial Version of the single-thread-based MNMF to update W.

```
Input: Initial feature matrices $W$ and $H$, sparse rating matrix $V$, $\overline{\Omega}_j$ and $\Omega_i$, the number of training epoches $epo$.
Output: $W$.
1:  for $iter$ from 1 to $epo$ do
2:    set $Down \leftarrow 0$, $Up \leftarrow 0$;
3:    for $i$ from 0 to $m-1$ do
4:      for $j \in \Omega_i$ do
5:        $\widetilde{v}_{i,j} \leftarrow \sum_{k=0}^{r-1} w_{i,k} h_{j,k}$;
6:        for $k$ from 0 to $r-1$ do
7:          $Up_{i,k} \leftarrow Up_{i,k} + h_{j,k} v_{i,j}$; %Update rule (24).
8:          $Down_{i,k} \leftarrow Down_{i,k} + h_{j,k} \widetilde{v}_{i,j}$;%Update rule (24).
9:          $Up_{i,k} \leftarrow Up_{i,k} + h_{j,k} v_{i,j} / \widetilde{v}_{i,j}$; %Update rule (29).
10:         $Down_{i,k} \leftarrow Down_{i,k} + h_{j,k}$;%Update rule (29).
11:       end for
12:     end for
13:   end for
14:   for $i$ from 0 to $m-1$ do
15:     for $l$ from 0 to $m-1$ do
16:       for $k$ from 0 to $r-1$ do
17:         $Up_{i,k} \leftarrow Up_{i,k} + \lambda_W p_{i,l} w_{l,k}$; %Update rule (24).
18:         $Up_{i,k} \leftarrow Up_{i,k} + \lambda_W p_{i,l} w_{l,k}$; %Update rule (29).
19:       end for
20:     end for
21:   end for
22:   for $i$ from 0 to $m-1$ do
23:     for $k$ from 0 to $r-1$ do
24:       $Down_{i,k} \leftarrow Down_{i,k} + \lambda_W q_{i,l} w_{l,k}$;%Update rule (24).
25:       $Down_{i,k} \leftarrow Down_{i,k} + \lambda_W q_{i,l} w_{l,k}$;%Update rule (29).
26:     end for
27:   end for
28:   for $i$ from 0 to $m-1$ do
29:     $w_{i,k} \leftarrow w_{i,k} (Up_{i,k} / Down_{i,k})$;
30:   end for
31: end for
32: return $W$;
```

remove the dependence among each w_i in Algorithm 1. At the same time, P and $\overline{P}$ are sparse and symmetric matrices, and Q and $\overline{Q}$ are diagonal matrices. In the derivation process, to keep the consistency with dense MNMF, we do not use symbols to denote the coordinates of non-zero values in matrices P and $\overline{P}$.

4.2. Complexity analysis

In this section, we analyze the time and space complexities of the single-thread-based MNMF.

Theorem 1 (Time complexity of the serial version single-thread-based MNMF)**.** *Let epo and r denote the number of update epochs and the dimension of the latent feature space, respectively. K is denoted as the K-top nearest neighbor rows within V, which is defined in Section 3.3. The time complexity of the single-thread-based MNMF (Algorithm 1) is* $O\big((3|\Omega|r + 2mr + mKr)epo\big)$.

Proof. The computation process is presented in Algorithm 1 (Lines 5, 7, 8, 9, 10, 17, 18, 24, 25, and 29) for updating W. The time complexity of the reduction operation (Line 5) is $O(|\Omega|r)$. The time complexity of updating Up (Lines 7, 9, 17, and 18) is $O(|\Omega|r + mKr)$. The time complexity of the diagonal matrix manipulation QW is $O(mr)$ (Lines 24, 25). Thus, the time complexity of updating $Down$ (Lines 8, 10, 24, and 25) is $O(|\Omega|r + mr)$. The time complexity of updating W (Line 29) is $O(mr)$. Thus, the total time complexity is $O(3|\Omega|r + 2mr + mKr)$ on one updating epoch for updating W. □

Theorem 2 (Space complexity of the serial version single-thread-based MNMF)**.** *The space complexity of the single-thread-based MNMF (Algorithm 1) is* $O\big(|\Omega| + mr + nr + mK + m + 2max(m,n)r\big)$.

Proof. The space complexity of the sparse matrix V is $O(|\Omega|)$. The space complexity of W and H are $O(mr)$ and $O(nr)$, respectively. Up and $Down$ can remove the dependence among the w_i as intermediate matrices, and the space complexity of *Up* and *Down* is $O\big(2max(m,n)r\big)$. The space complexity of *P* and *Q* is $O(mK+m)$. Thus, the total space complexity is $O\big(|\Omega|+mr+nr+mK+m+2max(m,n)r\big)$. □

Theorem 3 (Space complexity of CUDA parallelization single-thread-based MNMF (CUMNMF))**.** *The space complexity of CUMNMF is* $O\big(|\Omega|+mr+nr+mK+m+2max(m,n)r\big)$.

Proof. The space complexity of the sparse matrix V is $O(|\Omega|)$. The space complexity of the sparse matrices P and Q is $O(mK+m)$. The space complexity of W and H are $O(mr)$ and $O(nr)$, respectively. In CUMNMF, a thread block updates a $\mathbf{w}_i$ and a $\mathbf{h}_j$, and the thread k within the thread block updates $w_{i,k}$ and $h_{j,k}$. *Up* and *Down* lie in global memory as intermediate matrices, which can remove the dependence among the w_i, and the complexity of *Up* and *Down* is $O(2max(m,\ n))$. $\sum_{k=0}^{r-1} w_{i,k}h_{j,k}$ (Algorithm 1 Line 5) can be solved by shared memory. More details will be presented in Section 5. *W, H, P, Q* and the sparse matrix V lie in the global memory. Thus, the space complexity of CUMNMF is $O\big(|\Omega|+mr+nr+mK+m+2max(m,n)r\big)$. □

According to the theoretical analysis, the single-thread-based MNMF has the same function as the original MNMF, and the linear memory and time complexity overhead with the rank r.

5. CUMNMF

In this section, a CUMNMF approach is presented in Section 5.1, and some optimizing approaches about CUDA programming are reported in Section 5.2

5.1. Parallelization approach

From Section 3, updating W and H is symmetric. Thus, in the remainder of this paper, we only present CUDA parallelization approaches to updating W. Furthermore, we prove that CUMNMF can work under a row or column oriented sparse matrix compressive format, i.e., CSR, CSC, which can save on the GPU global memory overhead.

In this section, we show the scheduling strategies for the thread blocks. The *scheduling strategy* is how thread blocks select entries in the rating matrix V. In the centralized approach, first, a thread block selects a column index j, and the kth thread within the thread block T is denoted by T_k. Then, the thread block selects an entry index (i, j), where $i \in \overline{\Omega}_j$. Each thread block selects a feature vector $\mathbf{w}_i$, and C thread blocks update the selected feature vectors in parallel.

Algorithm 2 shows the thread block scheduling strategy of the centralized updating approach. Furthermore, we adopt

Algorithm 2 CUDA centralized updating W.

Input: Initial W and H, rating matrix V, regularization parameter λ_W, training epoches *epo*, and total number of thread blocks C.
Output: W.

```
1:  $T \leftarrow$ Thread block $id$.
2:  **for** $loop$ from $epo$ to 0 **do**
3:      set $Down \leftarrow 0$, $Up \leftarrow 0$;
4:      **parallel**:
5:      Update $Up_{i,k}$ by $T_k$ (Lines 7, 9 in Algorithm 1).
6:      Update $Down_{i,k}$ by $T_k$ (Lines 8, 10 in Algorithm 1).
7:      **end parallel**
8:      **parallel**:
9:      Update $Up_{i,k}$ by $T_k$ (Lines 17, 18 in Algorithm 1).
10:     **end parallel**
11:     **parallel**:
12:     Update $Down_{i,k}$ by $T_k$ (Lines 24, 25 in Algorithm 1).
13:     **end parallel**
14:     **parallel**:
15:     Update $w_{i,k}$ by $T_k$ (Line 29 in Algorithm 1).
16:     **end parallel**
17: **end for**
18: **return** $W$.
```

the histogram-based solution [1], which can elevate the load balance on the thread blocks.

5.2. Aligned memory access

We set the parameter r to be an integral multiple of 32 (warpsize), for *warp* synchronization execution and coalesced global memory access. Fig. 3 illustrates coalesced access on W and H in global memory. As shown in Fig. 3, a thread block

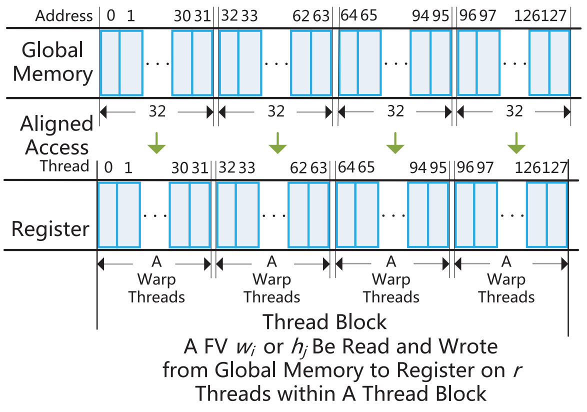


Fig. 3. Aligned access to a feature vector $\mathbf{w}_i$ or $\mathbf{h}_j$ in global memory from and to registers in threads within a thread block.

Table 2
The statistics and parameters for the data sets.

Data set	Movielen-1M	Movielen-10M	Netflix	Webscope-R1
m	6, 040	71, 567	480, 189	1, 948, 883
n	3, 900	10, 681	17, 770	98, 212
$\sharp$*Training*	900, 101	9, 301, 274	99, 072, 112	104, 215, 016
$\sharp$*Probing*	100, 108	698, 780	1, 408, 395	11, 364, 422

can access the feature vectors $\mathbf{w}_i$ and $\mathbf{h}_j$ continuously, because of the coalesced access on the GPU global memory via 32-, 64- or 128-byte memory transactions [27]. A *warp* (warpsize=32) accesses the successive 128-bytes in global memory at the same time if the memory access is aligned. For a 128-byte memory transaction, a warp fetches aligned 16 double-type elements from global memory to local memory two times. We set the number of threads per thread block to be $r \in \{32, 64, 128, 256, 512, 1024\}$, to synchronize the *warp* execution.

6. Experiments

We present the experimental settings and data set information in Section 6.1, and we test our experimental results to answer the following questions:

- *Q1: Parameter Selection (Section 6.2).* How do P100 GPUs obtain the highest performance according to its own parameters, under various conditions, i.e., the rank of feature matrix, different volumes of sparse matrix compressive format?
- *Q2: Scalability (Section 6.3).* How does CUMNMF scale with regard to various conditions, i.e., the rank of the feature matrix, different volumes of sparse matrix compressive format?
- *Q3: Convergence (Section 6.4).* How quickly and accurately do the state-of-the-art approaches factorize the real-world sparse matrix?

6.1. Experimental settings

RMSE Root Mean Square Error (RMSE) is generally accepted to evaluate the quality and accuracy of a rating recommender system [18], and it is defined as

$$RMSE = \sqrt{\frac{\sum\limits_{(i,j)\in\Gamma} (v_{i,j} - \widetilde{v}_{i,j})^2}{|\Gamma|}}, \tag{30}$$

where Γ denotes the test sets. RMSE is the square root of the average of the square differences between the estimated and the real preference, and RMSE more heavily penalizes the estimations that are far off than the Mean Error Average (MEA).

Data Sets The 4 data sets are used for the experiments, e.g., MovieLens-1M, MovieLens-10M[1], Netflix[2], Webscope_R1[3]. The statistics of the 4 data sets are presented in Table 2.

Platform An NVIDIA Tesla P100 GPU, each has 56 SMs. There are 64 SPs per SM, which operate at a 1.33 GHz clock rate, and there is 16GB of global memory with 4096 bits of bandwidth and a 715-MHz memory clock rate. The computing

[1] http://files.grouplens.org/datasets/movieLens
[2] http://www.netflixprize.com
[3] https://webscope.sandbox.yahoo.com/catalog.php?datatype=r

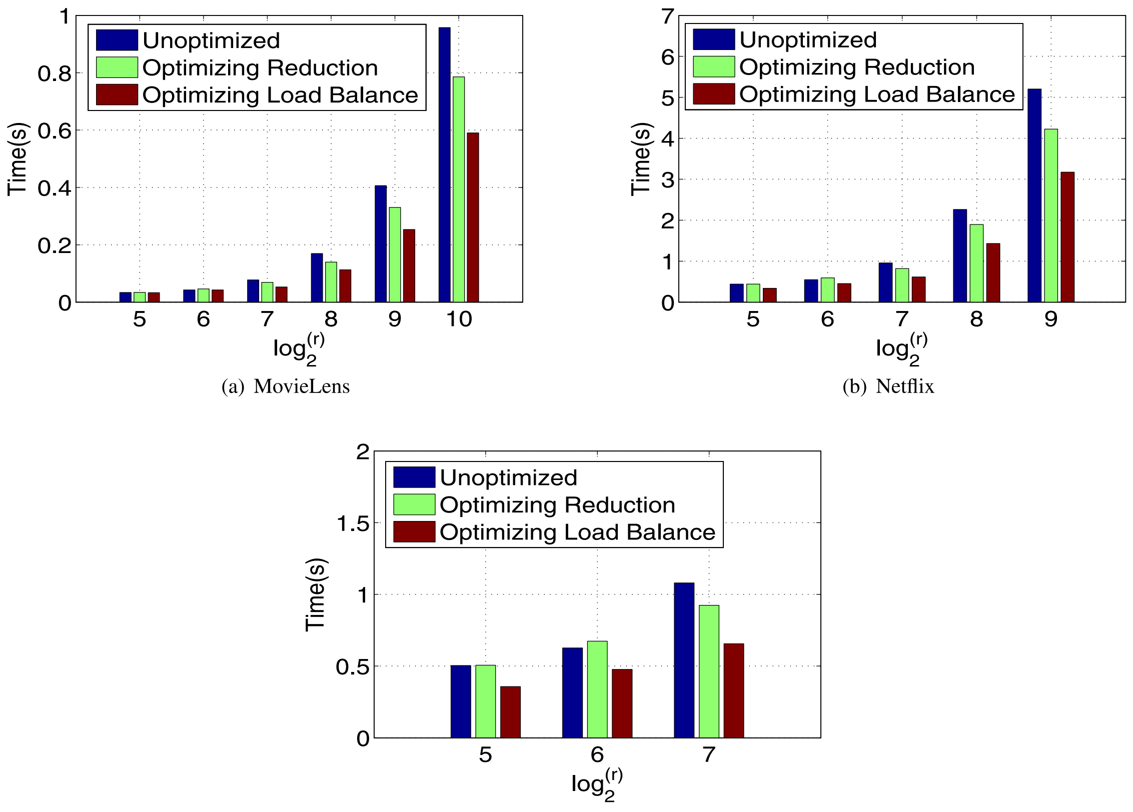


Fig. 4. Comparisons with the maximum synchronization instruction method (Unoptimized), the reduced synchronization instruction method (Optimizing Reduction) and the improved load balance (Optimizing Load Balance) for CUMNMF$_{Eu}$ with $K = 15$, on a P100 GPU.

capacity of a P100 GPU is 6.0 for CUMNMF. The CPU server is equipped with 2 Intel(R) Xeon(R) E5-2680 v4 CPUs, running on 2.4 GHz, for CCD++[4], and OpenMP parallelization version for NMF$_{Eu}$ (HPC-NMF [13]), and the corresponding revision version for NMF$_{KL}$. For simplicity, in the following sections, NMF$_{KL}$ is denoted as the serial version of NMF on KL-divergence, and the OpenMP parallelization version of NMF$_{KL}$. There are 14 cores in each processor, and the computation and communication are based on OpenMP.

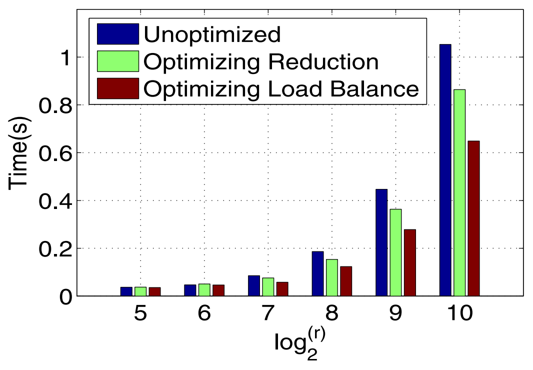
Parameters We present the parameters used for each data set in Table 2. The initial value settings (Uniform distribution) of $\{W, H\}$ are the same as HPC-NMF [13], NMF$_{Eu}$, NMF$_{KL}$ [16,19,44], and CCD++ [40]. We observe that CCD++ is double-precision floating point arithmetic, and the serial part of CCD++ is denoted as CCD. Thus, the double-precision floating point arithmetic is considered in the programming.

6.2. Performances of CUMNMF

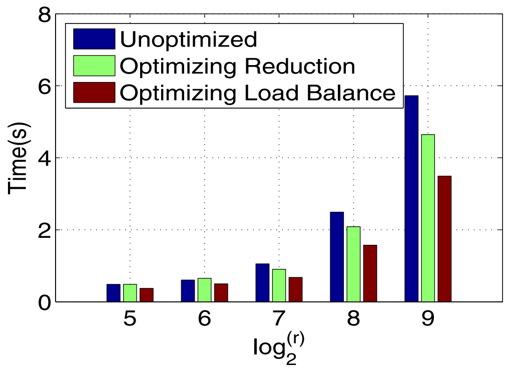
Increasing the rank (r) of the feature matrix can improve the accuracy of CF MF to a certain degree [17,18]. The time complexity of the sequential multiplication of two vectors is $O(r)$. The two-vector multiplication of CUMNMF requires $O\big(\log_2(r)\big)$. Thus, a larger r can improve the GPU speedup performance further. At the same time, threads within a thread block need synchronization overhead for vector multiplication and reduction, and decreasing the synchronization overhead can improve the computational efficiency of CUMNMF significantly. From the CUDA updating process described in Section 5.2, a thread block that has r threads can update a $\mathbf{h}_j$ only once by $\{\mathbf{w}_i, \mathbf{h}_j, \widetilde{v}_{i,j}\}$. The value of $\widetilde{v}_{i,j}$ requires the pre-computation of $\mathbf{w}_i\mathbf{h}_j^T$, and vector dot multiplication requires the cooperation of the r threads within the thread block. Thus, the $\mathbf{w}_i$ and $\mathbf{h}_j$ are stored in the shared memory of the thread block. A thread synchronization instruction can be saved from $\log_2(r)$ to $\log_2(r) - 5$ by the warp synchronization mechanism presented in [27]. Furthermore, we adopt the histogram-based solution [1], which can elevate the load balance on the thread blocks.

The appropriate r of the feature matrices can guarantee reasonable accuracy [17,18]. In our work, because we focus on the GPU accelerating performance on various value of r of the feature matrices rather than the choice of an appropriate

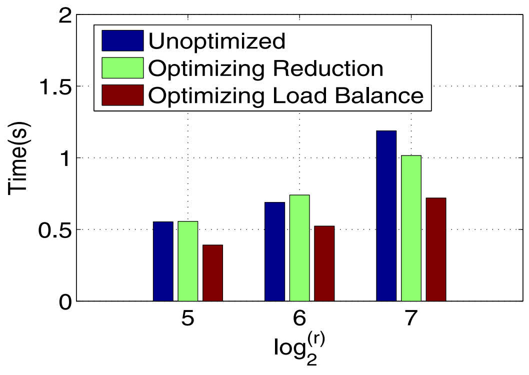
[4] http://www.cs.utexas.edu/~rofuy/libpmf/.



(a) MovieLens



(b) Netflix



(c) Webscope-R1

Fig. 5. Comparisons with the maximum synchronization instruction method (Unoptimized), the reduced synchronization instruction method (Optimizing Reduction) and improved load balance (Optimizing Load Balance) for CUMNMF$_{KL}$ with $K = 15$, on a P100 GPU.

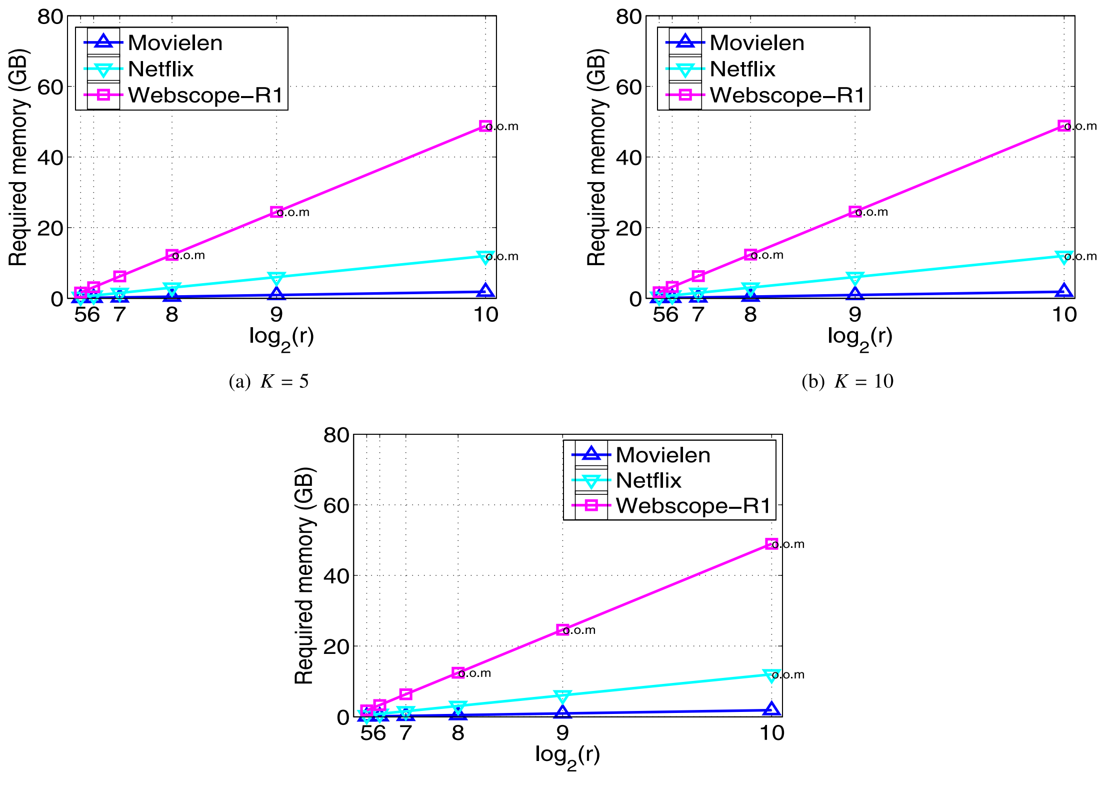
Table 3
The *occupancy* for CUMNMF on the P100 GPU.

r	32	64	128	256	512	1024
Register Per thread	21	21	21	21	21	21
Shared memory Per block(bytes)	256	512	1024	2048	4096	8192
Active Thread Per SM	1024	2048	2048	2048	2048	2048
Active Warps Per SM	32	64	64	64	64	64
Active Thread Blocks Per SM	32	32	16	8	4	2
Occupancy of per SM	50%	100%	100%	100%	100%	100%

r, we test five sets of experiments on various value of r, e.g., $r \in \{32, 64, 128, 256, 512, 1024\}$. GPU *occupancy* is the ratio of the number of active threads to the total number of threads, and high *occupancy* means that the GPU is working in high efficiency. The GPU *occupancy* is calculated by *CUDA Occupancy Calculator* [27]. In CUMNMF, a thread block has r threads, and the number of thread blocks is tunable, which can control the *occupancy*. Table 3 lists *occupancy* under various conditions, e.g., $r \in \{32, 64, 128, 256, 512, 1024\}$. According to Table 3, we set the optimal number of thread blocks {1792, 1792, 896, 448, 224, 112} for $r \in \{32, 64, 128, 256, 512, 1024\}$, respectively. Figs. 4 and 5 illustrate the comparisons with the maximum synchronization instruction method (Unoptimized), the reduced synchronization instruction method (Optimizing Reduction), and the improved load balance (Optimizing Load Balance). As shown in Figs. 4 and 5, when r is larger than 128, saving synchronization overhead can improve the GPU accelerating performance. We conclude the reason is that the saved cost of warp synchronization is larger than the cost of warp scheduling. Furthermore, improving the load balance within the thread blocks can improve the overall performance significantly.

6.3. Scalability performance

We measure the scalability of CUMNMF with regard to the rank of the feature matrices, and the scale of the input sparse matrix. As illustrated in Algorithm 1, and the space complexity analysis in Section 4.2, the single-thread-based MNMF including MNMF$_{Eu}$ and MNMF$_{KL}$, have the same space overhead. Thus, the memory scalability is shown in Fig. 6 only. The



(a) $K = 5$ (b) $K = 10$ (c) $K = 15$

Fig. 6. Scalability w.r.t. the rank of the feature matrices *r*, and the scale of the data set. Here, o.o.m.: out of memory on a P100 GPU, and $K \in \{5, 10, 15\}$. The memory requirement of CUMNMF is scalable with *r*.

space complexity of MNMF involves several parameters, i.e., *r*, *K*, and the volume of the compressive format of a sparse rating matrix. As shown in Fig. 6, CUMNMF scales with the rank of the feature matrix *r*, when P100 GPU works at the highest *occupancy* under $r \in \{32, 64, 128, 256, 512, 1024\}$. Fig. 6 illustrates that the space overhead increases linearly while *r* increases. With the same conditions shown in Fig. 6, under the highest *occupancy*, as shown in Figs. 7 and 8, the computational overhead of CUMNMF scales linearly with *r*.

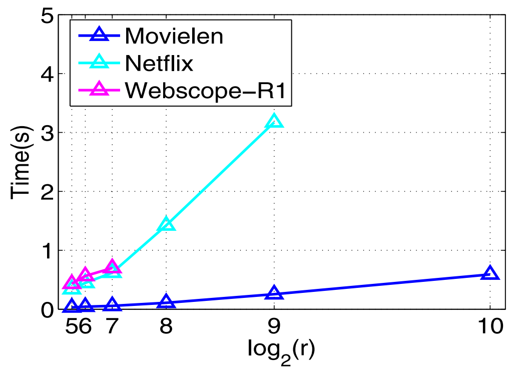
6.4. Comparisons

In this section, convergence analysis is conducted, and the optimal *K* is selected via experiments on $K \in \{5, 10, 15, 20, 25\}$ for MNMF$_{Eu}$ and MNMF$_{KL}$. To present the superiority of MNMF over NMF and CCD, experiments are conducted on Euclidean distance and KL-divergence with the optimal $K = 15$. Based on the fine-grained parallelization of single-thread-based MNMF, CUMNMF can obtain accelerated performance from the GPU. The experiments of selecting the optimal *K* for MNMF and accuracy comparisons among MNMF$_{Eu}$, HPC-NMF (Euclidean distance), CUMNMF$_{Eu}$ on Euclidean distance, and MNMF$_{KL}$, NMF$_{KL}$, CUMNMF$_{KL}$ on KL-divergence are presented in Section 6.4.1. The performance of CUMNMF is shown in Section 6.4.2.

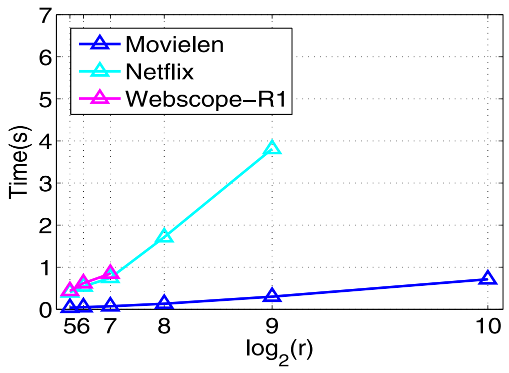
6.4.1. Convergence analysis

Figs. 9(a) and (b) depict the accuracy performance of MNMF$_{Eu}$ and MNMF$_{KL}$ on $K \in \{5, 10, 15, 20, 25\}$. Fig. 10 (a) presents the accuracy comparisons among MNMF$_{Eu}$, NMF$_{Eu}$, and CCD on CPU, and CUMNMF$_{Eu}$ on P100 GPU, and Fig. 10 (b) presents the accuracy comparisons among MNMF$_{KL}$, and NMF$_{KL}$ on CPU, and CUMNMF$_{KL}$ on P100 GPU with optimal $K = 15$. Note that, with the same initialization conditions of the factor matrices *W* and *H* and the regularization parameters λ_W and λ_H, CUMNMF$_{Eu}$ can obtain the same accuracy performance results as MNMF$_{Eu}$, and CUMNMF$_{KL}$ can obtain the same accuracy performance results as MNMF$_{KL}$. From these accuracy performance results from Figs. 9 and 10, we have the following findings:

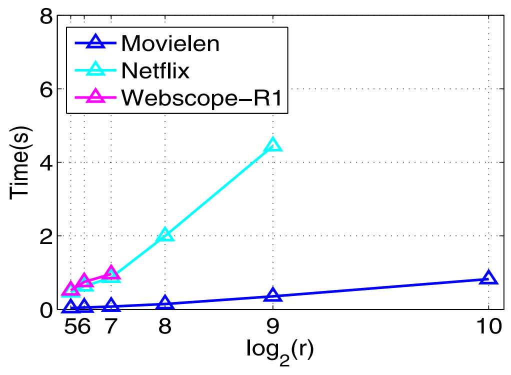
1. Various *K* can improve the accuracy to various degrees. As shown in Fig. 9 (a) and (b), MNMF$_{Eu}$ and MNMF$_{KL}$ with $K \in \{15, 20, 25\}$ outperform the counterparts $K \in \{5, 10\}$; Moreover, the accuracy of MNMF$_{Eu}$ and MNMF$_{KL}$ with $K \in \{15, 20, 25\}$ obtain the same accuracy, which indicate that there is an optimal choices of *K*. For example, the accuracy performance of



(a) $K = 5$



(b) $K = 10$



(c) $K = 15$

Fig. 7. Scalability of the computational time for CUMNMF$_{Eu}$ on $K \in \{5, 10, 15\}$ and P100 GPU.

MNMF with $K = \{15, 20, 25\}$ outperforms the counterparts $K = \{5, 10\}$ obviously; However, the accuracy performance of MNMF with $K = \{15, 20, 25\}$ obtains almost a similar accuracy performance. Thus, there is the an optimal K. To balance the accuracy and training time, $K = 15$ is chosen for the following experiments.

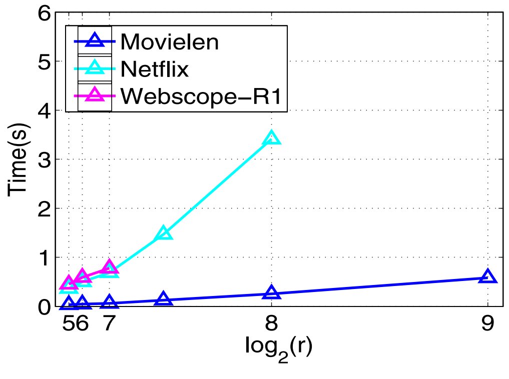
2. As illustrated in Figs. 9 and 10, K plays a large role in the accuracy gain. As shown in Fig. 10 (a), MNMF$_{Eu}$ and CUMNMF$_{Eu}$ with $K = 15$ outperform NMF$_{Eu}$ and CCD, and as shown in Fig. 10 (b), MNMF$_{KL}$ and CUMNMF$_{KL}$ with $K = 15$ outperform NMF$_{KL}$. We conclude that the reason is that MNMF can cooperate the implicit information of a data set, i.e., the geometry structure, neighborhood information, and so on.

6.4.2. Comparison of parallelization approaches

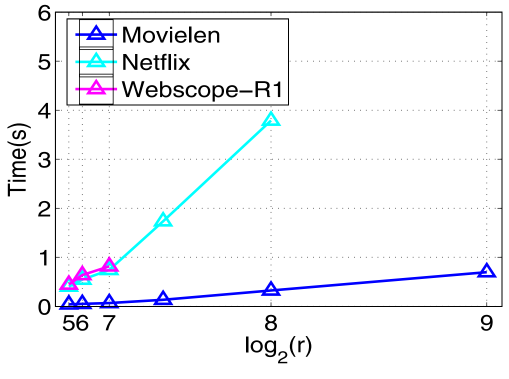
We compare CUMNMF$_{Eu}$ and CUMNMF$_{KL}$ with the state-of-the-art parallel and distributed works, e.g., CCD++, HPC-NMF (Euclidean distance), and NMF$_{KL}$ on $r = 128$ for fairness. Fig. 11 illustrates the accuracy performance (RMSE) versus the training time. As shown in Fig. 11, CUMNMF$_{Eu}$ and CUMNMF$_{KL}$ on P100 GPU run faster than CCD++, HPC-NMF, and NMF$_{KL}$ on the shared memory platform. We conclude that the reason is that the single-thread-based MNMF removes the dependence in each feature vector. Thus, this MNMF has fine-grained parallelization inherence, which is suited to CUDA parallelization on a GPU. Thus, with the high computing power of P100 GPU and the fine-grained parallelization inherence of CUMNMF, CUMNMF$_{Eu}$ can perform better than CCD++, HPC-NMF on a shared memory platform, and CUMNMF$_{KL}$ outperforms than NMF$_{KL}$ on a shared memory platform.

7. Conclusions and future work

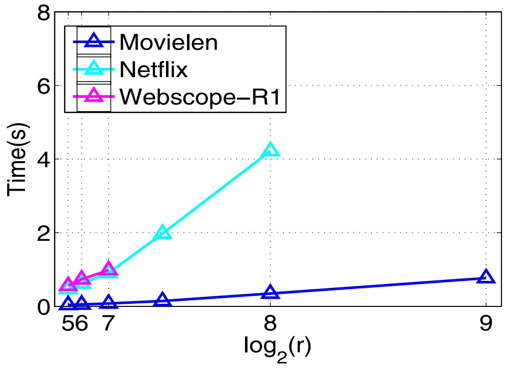
In this paper, we focus on the simplification of the computational process, memory optimization and fine-grained parallelization on MNMF with Euclidean distance and KL-divergence. First, we extend MNMF from the Euclidean distance to the KL-divergence, and the proposed MNMF can solve both the Gaussian and the Poisson probabilistic distribution maximization problems. Second, we derive the single-thread-based MNMF, which involves only the needed feature vector multiplications and element summations; at the same thime, single-thread-based MNMF holds the consecutive access characteristic on the feature elements within a feature vector. Thus, single-thread-based MNMF can remove the dependence among the feature vectors, and has the inherence of fine-grained parallelization. Last but not least, the inherence of fine-grained parallelization ensures the high performance on the GPU for CUMNMF.



(a) $K = 5$

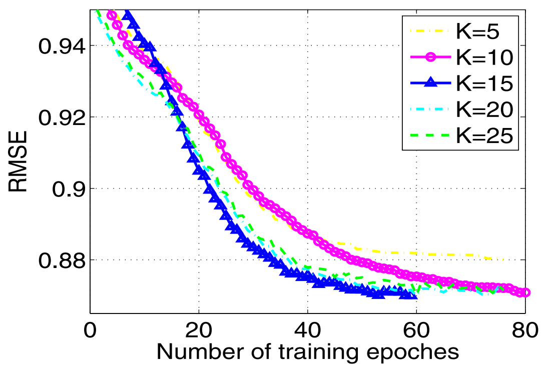


(b) $K = 10$

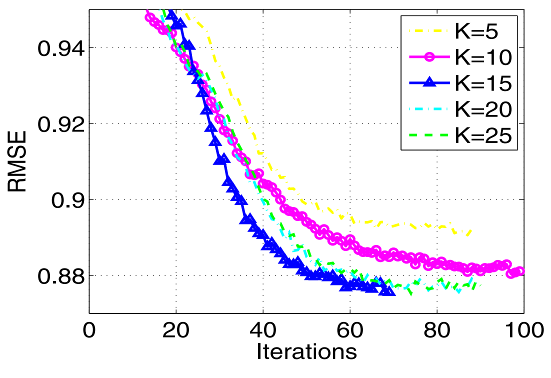


(c) $K = 15$

Fig. 8. Scalability of the computational time for CUMNMF$_{KL}$ on $K \in \{5, 10, 15\}$ and P100 GPU.



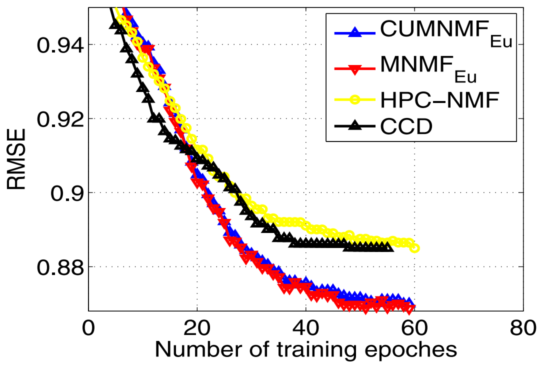
(a) MNMF$_{Eu}$



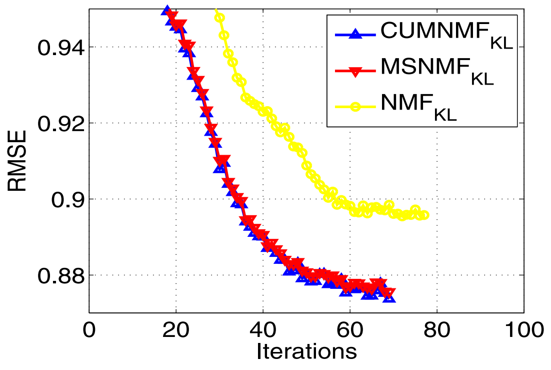
(b) MNMF$_{KL}$

Fig. 9. Selection of the optimal K for MNMF$_{Eu}$, and MNMF$_{KL}$ on Movielen-1M data set.

We observe that the Laplacian matrices P and $\overline{P}$ of MNMF must be symmetric, which prevent MNMF from obtaining high flexibility. Hence, future work will focus on the solving of un-symmetric Laplacian matrices to obtain a flexible MNMF. We would like to improve the convergence rate of the single-thread-based MNMF, and integrate the single-thread-based MNMF with an alternative direction multiplication method (ADMM) [9]. To solve the TB-scale MNMF problem, with fine-grained parallelism of the single-thread-based MNMF, we want to extend CUMNMF on a single GPU to multi-GPU; at the same time, we plan to extend CUMNMF to other distributed platforms, e.g., MapReduce, Hadoop. More recently, Sunway TaihuLight has become a high performance processor [6]. Thus, we want to implant our single-thread-based MNMF into a Sunway TaihuLight.

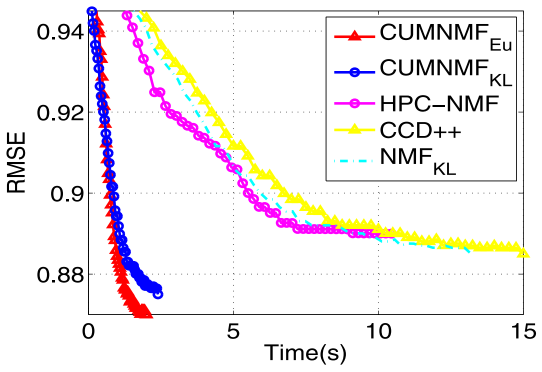


(a) Euclidean distance

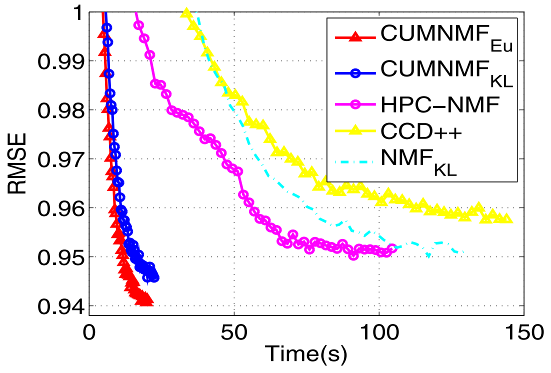


(b) KL-divergence

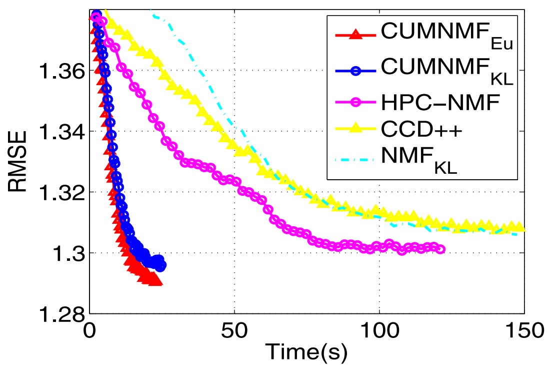
Fig. 10. Convergence comparisons: (a) the comparison among MNMF$_{Eu}$, CUMNMF$_{Eu}$ with $K = 15$, HPC-NMF, and CCD, and (b) the comparison among MNMF$_{KL}$, CUMNMF$_{KL}$ with $K = 15$, NMF$_{KL}$ on Movielen-1M data set.



(a) Movielen



(b) Netflix



(c) Webscope-R1

Fig. 11. Comparison of parallelization approaches.

Acknowledgments

This research was partially funded by National Key Research and Development Program of China (Grant no. 2016YFB0201303), the National Outstanding Youth Science Program of the National Natural Science Foundation of China (Grant no. 61625202), and the Key Program of the National Natural Science Foundation of China (Grant no. 61432005), the Natural Science Foundation of China (Grants nos. 61370097, 61672224), theNatural Science Foundation of Hunan Province, China (Grant no. 2018JJ2063), and the National Key R & D Program of China (Grant No. 2016YT80201900).

References

[1] A. Ashari, N. Sedaghati, J. Eisenlohr, S. Parthasarathy, P. Sadayappan, Fast sparse matrix-vector multiplication on gpus for graph applications, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE Press, 2014, pp. 781–792.
[2] T. Ben-Nun, E. Levy, A. Barak, E. Rubin, Memory access patterns: the missing piece of the multi-gpu puzzle, in: The International Conference for High Performance Computing, Networking, Storage and Analysis, ACM/IEEE, 2015, pp. 1–12.
[3] M. Cafaro, I. Epicoco, G. Aloisio, M. Pulimeno, Cuda based parallel implementations of space-saving on a gpu, in: High Performance Computing & Simulation (HPCS), 2017 International Conference on, IEEE, 2017, pp. 707–714.
[4] M. Cafaro, M. Pulimeno, I. Epicoco, G. Aloisio, Parallel space saving on multi-and many-core processors, Concurrency and Computation: Practice and Experience, 2017.
[5] D. Cai, X. He, J. Han, T.S. Huang, Graph regularized nonnegative matrix factorization for data representation, IEEE Trans. Pattern Anal. Mach. Intell. 33 (8) (2011) 1548–1560.
[6] Y. Chen, K. Li, X. Fei, Z. Quan, K. Li, Implementation and optimization of AES algorithm on the sunway taihulight, in: 17th International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2016, Guangzhou, China, December 16–18, 2016, 2016, pp. 256–261.
[7] A. Ezzat, P. Zhao, M. Wu, X.-L. Li, C.-K. Kwoh, Drug-target interaction prediction with graph regularized matrix factorization, IEEE/ACM Trans. Comput. Biol. Bioinf. 14 (3) (2017) 646–656.
[8] Z. Gao, Y. Liang, Y. Jiang, Implement of item-based recommendation on gpu, in: 2012 IEEE 2nd International Conference on Cloud Computing and Intelligent Systems (CCIS), 2, IEEE, 2012, pp. 587–590.
[9] Q. Gu, J. Zhou, C. Ding, Collaborative filtering: Weighted nonnegative matrix factorization incorporating user and item graphs, in: Proceedings of the 2010 SIAM International Conference on Data Mining, SIAM, 2010, pp. 199–210.
[10] N. Guan, D. Tao, Z. Luo, B. Yuan, Manifold regularized discriminative nonnegative matrix factorization with fast gradient descent, IEEE Trans. Image Process. 20 (7) (2011) 2030–2048.
[11] P. Guo, L. Wang, P. Chen, A performance modeling and optimization analysis tool for sparse matrix-vector multiplication on gpus, IEEE Trans. Parallel Distrib. Syst. 25 (5) (2014) 1112–1123.
[12] X. He, H. Zhang, M.Y. Kan, T.S. Chua, Fast matrix factorization for online recommendation with implicit feedback, in: The International ACM SIGIR Conference, 2016, pp. 549–558.
[13] R. Kannan, G. Ballard, H. Park, Mpi-faun: an mpi-based framework for alternating-updating nonnegative matrix factorization, IEEE Trans. Knowl. Data Eng. pp (99) (2017). 1–1
[14] R. Kannan, H. Woo, C.C. Aggarwal, H. Park, Outlier detection for text data, in: Proceedings of the 2017 SIAM International Conference on Data Mining, SIAM, 2017, pp. 489–497.
[15] K. Kato, T. Hosino, Singular value decomposition for collaborative filtering on a gpu, in: IOP Conference Series: Materials Science and Engineering, 10, IOP Publishing, 2010, pp. 012–017.
[16] Y.-D. Kim, S. Choi, Weighted nonnegative matrix factorization, in: Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on, IEEE, 2009, pp. 1541–1544.
[17] Y. Koren, Factor in the neighbors: scalable and accurate collaborative filtering, ACM Trans. Knowl. Discov. Data (TKDD) 4 (1) (2010) 1.
[18] Y. Koren, R. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, Computer (8) (2009) 30–37.
[19] D.D. Lee, H.S. Seung, Algorithms for non-negative matrix factorization, in: Advances in Neural Information Processing Systems, 2001, pp. 556–562.
[20] H. Li, K. Li, J. An, K. Li, Msgd: a novel matrix factorization approach for large-scale collaborative filtering recommender systems on gpus, IEEE Trans. Parallel Distrib. Syst. PP (99) (2017). 1–1
[21] J. Li, J. M. Bioucas-Dias, A. Plaza, L. Liu, Robust collaborative nonnegative matrix factorization for hyperspectral unmixing, IEEE Trans. Geosci. Remote Sens. 54 (10) (2016) 6076–6090.
[22] K. Li, W. Yang, K. Li, Performance analysis and optimization for spmv on gpu using probabilistic modeling, IEEE Trans. Parallel Distrib. Syst. 26 (1) (2015) 196–205.
[23] X. Li, G. Cui, Y. Dong, Graph regularized non-negative low-rank matrix factorization for image clustering, IEEE Trans. Cybern. (2017).
[24] C. Liu, H.-c. Yang, J. Fan, L.-W. He, Y.-M. Wang, Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce, in: Proceedings of the 19th International Conference on World Wide Web, ACM, 2010, pp. 681–690.
[25] T. Liu, M. Gong, D. Tao, Large-cone nonnegative matrix factorization, IEEE Trans. Neural Netw. Learn Syst. (2016) 1–14.
[26] E. Mejía-Roa, D. Tabas-Madrid, J. Setoain, C. García, F. Tirado, A. Pascual-Montano, Nmf-mgpu: non-negative matrix factorization on multi-gpu systems, BMC Bioinformatics 16 (1) (2015) 1.
[27] C. Nvidia, Nvidia cuda c programming guide, NVIDIA Corporation 120 (2011) 18.
[28] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, J.C. Phillips, Gpu computing, Proc. IEEE 96 (5) (2008) 879–899.
[29] G. Pratx, L. Xing, Gpu computing in medical physics: a review, Med. Phys. 38 (5) (2011) 2685–2697.
[30] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, Application of Dimensionality Reduction in Recommender System–A Case Study, Technical Report, DTIC Document, 2000.
[31] N. Srebro, J. Rennie, T.S. Jaakkola, Maximum-margin matrix factorization, in: Advances in Neural Information Processing Systems, 2004, pp. 1329–1336.
[32] M. Sun, Y. Li, J.F. Gemmeke, X. Zhang, Speech enhancement under low snr conditions via noise estimation using sparse and low-rank nmf with kullback–leibler divergence, IEEE/ACM Trans. Audio, Speech Lang. Process. (TASLP) 23 (7) (2015) 1233–1242.
[33] G. Trigeorgis, K. Bousmalis, S. Zafeiriou, B.W. Schuller, A deep matrix factorization method for learning attribute representations, IEEE Trans. Pattern Anal. Mach. Intell. 39 (3) (2017) 417–429.
[34] Y.-X. Wang, Y.-J. Zhang, Nonnegative matrix factorization: a comprehensive review, IEEE Trans. Knowl. Data Eng. 25 (6) (2013) 1336–1353.
[35] W. Wu, S. Kwong, Y. Zhou, Y. Jia, W. Gao, Nonnegative matrix factorization with mixed hypergraph regularization for community detection, Inf. Sci. (2018).
[36] Z. Wu, S. Ye, J. Liu, L. Sun, Sparse non-negative matrix factorization on gpus for hyperspectral unmixing, IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 7 (8) (2014) 3640–3649.
[37] K. Xie, X. Ning, X. Wang, D. Xie, J. Cao, G. Xie, J. Wen, Recover corrupted data in sensor networks: a matrix completion solution, IEEE Trans. Mob. Comput. 16 (5) (2017) 1434–1448.
[38] K. Xie, L. Wang, X. Wang, G. Xie, J. Wen, Low cost and high accuracy data gathering in wsns with matrix completion, IEEE Trans. Mob. Comput. 17 (7) (2018) 1595–1608.
[39] W. Yang, K. Li, Z. Mo, K. Li, Performance optimization using partitioned spmv on gpus and multicore cpus, IEEE Trans. Comput. 64 (9) (2015) 2623–2636.
[40] H.-F. Yu, C.-J. Hsieh, S. Si, I. Dhillon, Scalable coordinate descent approaches to parallel matrix factorization for recommender systems, in: Data Mining (ICDM), 2012 IEEE 12th International Conference on, IEEE, 2012, pp. 765–774.
[41] F. Zhang, Y. Lu, J. Chen, S. Liu, Z. Ling, Robust collaborative filtering based on non-negative matrix factorization and r 1 -norm, Knowl. Based Syst. (2016).
[42] J.D. Zhang, C.Y. Chow, J. Xu, Enabling kernel-based attribute-aware matrix factorization for rating prediction, IEEE Trans. Knowl. Data Eng. (99) (2017) 1.
[43] L. Zhang, Q. Zhang, B. Du, D. Tao, J. You, Robust manifold matrix factorization for joint clustering and feature extraction, in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4–9, 2017, San Francisco, California, USA., 2017, pp. 1662–1668.

[44] S. Zhang, W. Wang, J. Ford, F. Makdon, Learning from incomplete ratings using non-negative matrix factorization, in: 2006 Proceeding SIAM International Conference of Data Mining (SDM), SIAM, 2006, pp. 43–47.
[45] Y. Zhang, D. Yi, B. Wei, Y. Zhuang, A gpu-accelerated non-negative sparse latent semantic analysis algorithm for social tagging data, Inf. Sci. 281 (2014) 687–702.
[46] Z. Zheng, H. Ma, M.R. Lyu, I. King, Collaborative web service qos prediction via neighborhood integrated matrix factorization, IEEE Trans. Serv. Comput. 6 (3) (2013) 289–299.
[47] B. Zou, C. Li, L. Tan, H. Chen, Gputensor: efficient tensor factorization for context-aware recommendations, Inf. Sci. 299 (2015) 159–177.