An Online and Scalable Model for Generalized Sparse Nonnegative Matrix Factorization in Industrial Applications on Multi-GPU

Hao Li, Kenli Li, *Senior Member, IEEE*, Jiyao An, *Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—Generalized sparse nonnegative matrix factorization (SNMF) has been proven useful in extracting information and representing sparse data with various types of probabilistic distributions from industrial applications, e.g., recommender systems and social networks.However, current solution approaches for generalized SNMF are based on the manipulation of whole sparse matrices and factor matrices, which will result in large-scale intermediate data.Thus, these approaches cannot describe the high-dimensional and sparse matrices in mainstream industrial and big data platforms, e.g., graphics processing unit (GPU) and multi-GPU, in an online and scalable manner. To overcome these issues, an online, scalable, and single-thread-based SNMF for CUDA parallelization on GPU (CUSNMF) and multi-GPU (MCUSNMF) is proposed in this article. First, theoretical derivation is conducted, which demonstrates that the CUSNMF depends only on the products and sums of the involved feature tuples. Next, the compactness, which can follow the sparsity pattern of sparse matrices, endows the CUSNMF with online learning capability and the fine granularity gives it high parallelization potential on GPU and multi-GPU. Finally, the performance results on several real industrial datasets demonstrate the linear scalability of the time overhead and the space requirement and the validity of the extension to online learning. Moreover, CUSNMF obtains speedup of 7X on a P100 GPU compared to that of the state-of-the-art parallel approaches on a shared memory platform.**

***Index Terms*—Big data and industrial applications, graphics processing unit (GPU) and multi-GPU, generalized divergence styles, online learning, recommender systems and social networks, single-thread-based model, sparse nonnegative matrix factorization (SNMF).**

Manuscript received August 2, 2018; revised November 5, 2018; accepted January 15, 2019. Date of publication January 31, 2019; date of current version September 29, 2021. This research was supported in part by the National Key Research and Development Program of China under Grant 2016YFB0201303 and Grant SQ2018YFB020061, in part by the National Outstanding Youth Science Program of the National Natural Science Foundation of China under Grant 61625202, in part by the Natural Science Foundation of Hunan Province, China under Grant 2018JJ2063, in part by the Program of National Natural Science Foundation of China under Grant 61751204, and in part by the International (Regional) Cooperation and Exchange Program of National Natural Science Foundation of China under Grant 61661146006 and Grant 61860206011. Paper no. TII-18-2021. *(Corresponding author: Kenli Li.)*

Hao Li, Kenli Li, and Jiyao An are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China, and also with National Supercomputing Center, Changsha 410082, China (e-mail: lihao123@hnu.edu.cn; lkl@hnu.edu.cn; jt_anbob@hnu.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2019.2896634.

Digital Object Identifier 10.1109/TII.2019.2896634

I. Introduction

FACING data explosion problems in the era of big data, the techniques for real-time and accurate analysis and their scalability to big data and industrial platforms have been explored perseveringly and extensively [1]–[3]. Dimensionality reduction is a widely used model in big data representation, analysis, modeling, and monitoring [4] because it is a simple approach that can be used to represent various types of data and can extract the core information from big data [5]. Nonnegative matrix factorization (NMF) has become one of the most popular models for dimensionality reduction over the past few decades due to the properties of various data types, such as low rank and nonnegativity. This approach factorizes the original matrix into two low-rank factor matrices with nonnegativity constraints, and the two factor matrices are combined to represent the original matrix [6]–[10].

The NMF can approximate the original matrix under low-rank and nonnegativity constraints and can represent various types of probabilistic distributions via maximum likelihood, which is equivalent to solving various divergence minimization problems, e.g., Euclidean distance, Kullback–Leibler (KL), and Itakura-Saito (IS) divergence [11]–[15]. Lee $et\ al.$ [16] proposed using NMF to approximate a face image via multiplicative update (MU) with nonnegative initialization factor matrices to minimize the Euclidean distance and the KL divergence. Thus, the training process of the NMF involves only solving the corresponding optimization problem for a distribution style and updating the factor matrices, which is equivalent to solving a generalized NMF model. After that, the NMF plays a substantial role in hyperspectral image processing [6], [7], image clustering [8], [9], [15], [17], etc.; in such cases, it is called dense NMF (DNMF) because the input data are in the form of a dense matrix and the dataset is of small scale, e.g., COLT20 (1440×1024) or PIE (2856×1024). Currently, performing DNMF in mainstream code libraries, e.g., MATLAB, OpenBLAS in C, Numpy in Python, and Armadillo in C++, involves frequent manipulations of matrices. Those libraries can support manipulations of small matrices on a single server. However, for large-scale datasets, the intermediate generated matrices may result in memory overload. Hence, those libraries must utilize the big data and industrial platforms.

With the rapidly increasing numbers of netizens, commodities, and network nodes, the relationships among those entities are of the extremely sparse and large-scale form [18]–[20] due to missing information and can be modeled as high-dimensional and sparse (HiDS) matrices intuitively. Several proposed approaches extract the data features from the HiDS matrices using dimension reduction, including Bayesian and autoencoder approaches in deep-learning communities [21], the kernel approach [22], and sparse NMF (SNMF). The Bayesian, autoencoder, and kernel methods are nonlinear dimension reduction approaches. Compared to the SNMF, the three approaches can extract more accurate features but require much higher time overhead. The SNMF is a linear approach for dimension reduction, which can realize a balance between time overhead and accuracy. The main difference between the SNMF and DNMF is that the DNMF considers the whole original matrix via the approximation matrix; however, the SNMF updates the two factor matrices by the observed nonzero elements and should distinguish between the observed and unobserved elements, which requires additional matrices and operations [23], [24]. Thus, the SNMF can be used to analyze the users' preferences for commodities in recommender systems [25]; facilitate analysis in bioinformatics [26], [27]; identify potential relationships among users and detect communities in social networks [28], [29]; detect outliers in massive text data, which can facilitate the identification of rumors in Internet environments [30]; and quickly detect anomalies among network nodes in industrial applications [31] based on historical HiDS matrices. Those applications provide important motivation to improve the accuracy and to model data to obtain useful information, which makes substantial contributions to the SNMF via dimension reduction; however, operation when the local server cannot accommodate full HiDS matrices is not considered. Thus, the main motivation of this work is to realize a scalable, fine-grained parallelizing model for the generalized SNMF that caters to the computational characteristics of industrial platforms.

Satisfying the requirements of real-time performances and accurate processing results for big data requires suitable industrial platforms and mathematical models. Recently, the U.S. *Summit* strikes back to the top supercomputer [32], which consists of 27648 Tesla V100 GPUs, 9216 IBM Power CPUs, and 10 PB memory. This situation implies the following.

1) In the era of big data, high-capacity and high-speed processors are required to obtain real-time analysis results [33], [34].
2) The processing approach should be well suited to the computation structures of big data and industrial platforms so that it can obtain optimal performance [35]–[37].

The graphics processing unit (GPU) performs well for stream-like and fine-grained processing styles [38], [39], and the two mainstream big data and industrial platforms, namely, Spark and Flink, performs well for stream-like computations [40]–[43]. Thus, Spark, Flink, and GPU can complement one another. However, current parallel and distributed models for the SNMF involve only the basic optimization approaches [44], such as gradient descent (GD), alternative least square (ALS) [45]–[47], coordinate cyclic descent (CCD) [48], [49], and MU [11], [50]. We observe the following:

1) in addition to the HiDS matrices, many acquired datasets from big data include other information, such as geographical and temporal-spatio attributes [51], [52];
2) many acceleration algorithms have been developed, such as alternative direction multiplier method (ADMM) and the Nesterov approach [53];
3) the massive volume of big data requires sufficient memory capacity on distributed nodes, and the emerging edge computing approach collects data on local nodes directly;
4) parallel and distributed models do not take into account the variety of big data.

However, current solution approaches for parallel models do not integrate with acceleration algorithms and online learning on an incremental manner and require frequent manipulations of HiDS matrices and low-rank matrices, which will result in intermediate data explosion problems; at the same time, frequent communication patterns, which are due to iterative nature of current algorithms, and quadratic communication overhead prevent the distributed approaches from obtaining high efficiency [54]–[59].

To overcome the aforementioned problems, a single-thread-based model for the generalized SNMF is proposed, which can transform whole factor matrix manipulations into the constituent feature element operations and is amenable to fine-grained parallelization. To the best of our knowledge, this is the first work that realizes linear scalability, online learning, and GPU and multi-GPU parallelization for the generalized SNMF. The main contributions of this work are as follows:

1) we present a theoretical derivation and proof of linear time complexity and space requirements for the single-thread-based model;
2) we demonstrate that the streamline-like style of the single-thread-based model gives the generalized SNMF online manner capability with linear scalability;
3) the fine-grained parallelizability enables the single-thread-based model to be extended to CUDA parallelization on GPU (CUSNMF) and multi-GPU (MCUSNMF), with linear time complexity and communication overhead.

The remainder of this article is organized as follows. Section II presents the preliminaries. Section III introduces the single-thread-based model for generalized SNMF, online learning, CUSNMF, and MCUSNMF. Section IV discusses experimental results. Finally, Section V concludes this article.

II. Problem Formulation and Preliminaries

A. Problem and Notations

In this section, the main notations include the matrices and vectors, along with their basic elements and basic operations, and the compressed format of sparse matrices (in Table I).

Definition 1 (SNMF): Given a HiDS matrix $\mathbf{V} \in \mathbb{R}_{+}^{m\times n}$ and a divergence function $\mathcal{D}(\mathcal{P}_\Omega(\mathbf{V})\|\mathcal{P}_\Omega(\widetilde{\mathbf{V}}))$, which evaluates the distance between the two specified matrices on the nonzero elements, where $\mathcal{P}_\Omega$ is the projection operator on the index set, which is denoted by Ω. The SNMF problem is to find $\mathbf{W} \in \mathbb{R}_{+}^{m\times r}$

TABLE I
TABLE OF SYMBOLS

Symbol	Definition
$\mathbf{V}$	input matrix ($\in \mathbb{R}_{+}^{m\times n}$, bold-faced uppercase letters)
$\Omega/\overline{\Omega}$	non-zero index set in row / column orientation
$\Omega_i/\overline{\Omega}_j$	column/row indices in the ith row/jth column
$\mathbf{W}/\mathbf{H}$	left factor matrix $\in \mathbb{R}_{+}^{m\times r}$ / right factor matrix $\in \mathbb{R}_{+}^{n\times r}$
w_i/h_j	ith row of $\mathbf{W}$ / the jth row of $\mathbf{H}$
$w_{i,r}/h_{j,r}$	rth element of w_i / the rth element of h_j
$\overline{w}_k/\overline{h}_k$	rth column of $\mathbf{W}$ / the rth column of $\mathbf{H}$
$\overline{w}_{k,i}/\overline{h}_{k,j}$	ith element of $\overline{w}_k$ / the jth element of $\overline{h}_k$
$\circ/(-,/)$	element-wise multiplication/element-wise division

and $\mathbf{H} \in \mathbb{R}_{+}^{n\times r}$ such that $\mathcal{D}(\mathcal{P}_{\Omega}(\mathbf{V})\|\mathcal{P}_{\Omega}(\widetilde{\mathbf{V}}))$ is minimized, where $\widetilde{\mathbf{V}} = \mathbf{W}\mathbf{H}^T$.

A probabilistic interpretation of the SNMF is to consider $v_{i,j}$ an observation from a distribution. When we take $v_{i,j} \sim \text{Gaussian}(\widetilde{v}_{i,j}, \sigma^2)$, $v_{i,j} \sim \text{Poisson}(\widetilde{v}_{i,j})$, and $v_{i,j} \sim \text{Exponential}(\widetilde{v}_{i,j})$, the three maximum-likelihood problems become minimization problems of the Euclidean distance ($\mathcal{D}_{\text{Eu}}$), KL-divergence ($\mathcal{D}_{\text{KL}}$), and IS-divergence ($\mathcal{D}_{\text{KL}}$), which are widely used [11]–[16], and defined as

$$\begin{cases} \arg\min_{\mathbf{W},\mathbf{H}} d_{\text{Eu}} = \|\mathcal{P}_{\Omega}(\mathbf{V}) - \mathcal{P}_{\Omega}(\widetilde{\mathbf{V}})\|^2 \\ \arg\min_{\mathbf{W},\mathbf{H}} d_{\text{KL}} = \sum_{\Omega}\left(\mathcal{P}_{\Omega}(\widetilde{\mathbf{V}}) - \mathcal{P}_{\Omega}(\mathbf{V}) \circ \log\left(\mathcal{P}_{\Omega}(\widetilde{\mathbf{V}})\right)\right) \\ \arg\min_{\mathbf{W},\mathbf{H}} d_{\text{IS}} = \sum_{\Omega}\left(\frac{\mathcal{P}_{\Omega}(\mathbf{V})}{\mathcal{P}_{\Omega}(\widetilde{\mathbf{V}})} + \log(\mathcal{P}_{\Omega}(\widetilde{\mathbf{V}}))\right) \end{cases} \tag{1}$$

respectively, with nonnegativity constraints $\mathbf{W}, \mathbf{H} \geq 0$, where $\widetilde{v}_{i,j} = \sum_{k=0}^{r-1} w_{i,k} h_{j,k}$.

B. MU for Generalized SNMF

The three optimization problems in (1) are nonconvex; however, when $\mathbf{W}$ is fixed, the Bregman divergence with a convex function can be minimized to update $\mathbf{H}$; this alternative optimization approach can be used to solve the three optimization problems and vice versa [6]–[16]. We observe that when $\mathbf{V} = \widetilde{\mathbf{V}}$, the value of the Bregman divergence function is 0. In the collaborative filtering (CF) problems, $\mathbf{V}$ is sparse and the distribution of nonzero entries is nonuniform. To adopt the sparse matrix factorization to the CF problem, an indicator matrix of the weighted NMF (WNMF) is introduced [11], [23], [24]. Applying GD and omitting the negative items with guaranteed convergence [6]–[16], the update rules for the generalized SNMF of $\mathcal{D}_{\text{Eu}}$, $\mathcal{D}_{\text{KL}}$, and $\mathcal{D}_{\text{IS}}$ are as follows:

$$\begin{cases} \mathbf{W} \leftarrow \mathbf{W} \circ \frac{\mathbf{V}\mathbf{H}}{(\mathbf{G}\circ\widetilde{\mathbf{V}})\mathbf{H}}, \mathbf{H} \leftarrow \mathbf{H} \circ \frac{\mathbf{V}^T\mathbf{W}}{(\mathbf{G}\circ\widetilde{\mathbf{V}})^T\mathbf{W}} \\ \mathbf{W} \leftarrow \mathbf{W} \circ \frac{(\mathbf{G}\circ\frac{\mathbf{V}}{\widetilde{\mathbf{V}}})\mathbf{H}}{\mathbf{G}\mathbf{H}}, \mathbf{H} \leftarrow \mathbf{H} \circ \frac{(\mathbf{G}\circ\frac{\mathbf{V}}{\widetilde{\mathbf{V}}})^T\mathbf{W}}{\mathbf{G}^T\mathbf{W}} \\ \mathbf{W} \leftarrow \mathbf{W} \circ \frac{(\mathbf{G}\circ\frac{\mathbf{V}}{\widetilde{\mathbf{V}}^2})\mathbf{H}}{(\mathbf{G}\circ\frac{1}{\widetilde{\mathbf{V}}})\mathbf{H}}, \mathbf{H} \leftarrow \mathbf{H} \circ \frac{(\mathbf{G}\circ\frac{\mathbf{V}}{\widetilde{\mathbf{V}}^2})^T\mathbf{W}}{(\mathbf{G}\circ\frac{1}{\widetilde{\mathbf{V}}})^T\mathbf{W}} \end{cases} \tag{2}$$

respectively, where $\mathbf{G} \in \mathbb{R}^{m\times n}$, $\mathbf{G}_{i,j} = 1$, if $(i,j) \in \Omega$, and $\mathbf{G}_{i,j} = 0$, if $(i,j) \notin \Omega$. This update rule can be applied to the SNMF; however the manipulation of the HiDS matrices and factor matrices will result in the following problems.

1) Intermediate data explosion: The time and space complexities for $\{\mathbf{G}\circ(\mathbf{W}\mathbf{H}^T), \mathbf{G}\circ\frac{\mathbf{V}}{\mathbf{W}\mathbf{H}^T}, \mathbf{G}\circ\frac{\mathbf{V}}{(\mathbf{W}\mathbf{H}^T)^2}, \mathbf{G}\circ\frac{1}{\mathbf{W}\mathbf{H}^T}\}$ are $O(|\Omega|r)$ and $O(|\Omega|)$, respectively.
2) Online learning obstruction: When new entries are added into the HiDS matrix, the current update rule (2) must be reconstructed. The number of new elements is smaller than the number of old ones [54]–[59]. When the update rules (2) are applied to online problems, the time complexity is not scalable with the number of new entries.

C. Solution Approaches for the Euclidean Distance ($\mathcal{D}_{Eu}$)

The solution approaches for the minimization of $\mathcal{D}_{\text{Eu}}$ in (1) with L_2 regularization include GD, ALS, MU, and CCD [45]–[50]. The main differences among the four approaches are the choice of training step and the number of training parameters of a feature vector. $\mathcal{D}_{\text{Eu}}$ with L_2 regularization is given by

$$d_{Eu} = \sum_{(i,j)\in\Omega} (v_{i,j} - \widetilde{v}_{i,j})^2 + \lambda_{\mathbf{W}}\|\mathbf{W}\|_F^2 + \lambda_{\mathbf{H}}\|\mathbf{H}\|_F^2 \tag{3}$$

where $\|\bullet\|_F$ is the Frobenius norm. d_{Eu} can be split into many independent subproblems: $d_{\text{Eu}} = \sum_{i=0}^{m-1}(d_{\text{Eu}})_i$ and $d_{\text{Eu}} = \sum_{j=0}^{n-1}(\overline{d}_{\text{Eu}})_j$, where $(d_{\text{Eu}})_i = \sum_{j\in\Omega_i}(v_{i,j} - \widetilde{v}_{i,j})^2 + \lambda_{\mathbf{W}}\|w_i\|_F^2$, and $(\overline{d}_{\text{Eu}})_j = \sum_{i\in\overline{\Omega}_j}(v_{i,j} - \widetilde{v}_{i,j})^2 + \lambda_{\mathbf{H}}\|h_j\|_F^2$. The gradient is given by $\partial(d_{\text{Eu}})_i/\partial w_i = w_i\mathbf{B}_i - c_i$ and $(\overline{d}_{\text{Eu}})_j/\partial h_j = h_j\overline{\mathbf{B}}_j - \overline{c}_j$, where $\mathbf{B}_i = \sum_{j\in\Omega_i} h_j^T h_j + \lambda_{\mathbf{W}} I_r$ and $\overline{\mathbf{B}}_j = \sum_{i\in\overline{\Omega}_j} w_i^T w_i + \lambda_{\mathbf{H}} I_r$, and $c_i = \sum_{j\in\Omega_i} v_{i,j} h_j$ and $\overline{c}_j = \sum_{i\in\overline{\Omega}_j} v_{i,j} w_i$. I_r is the r by r identity matrix.

1) Alternative Least Square (ALS): When the gradients $\{\partial(d_{\text{Eu}})_i/\partial w_i, (\overline{d}_{\text{Eu}})_j/\partial h_j\}$ are set to 0, we can determine the optimal training parameters [45]–[47]. The update rule for the ALS is formulated as

$$w_i \leftarrow c_i\mathbf{B}_i^{-1}; h_j \leftarrow \overline{c}_j\overline{\mathbf{B}}_j^{-1}. \tag{4}$$

Updating a row w_i of the factor matrix $\mathbf{W}$, for example, using (4), involves taking $O(|\Omega_i|(r + r^2) + r^3)$, which consists of $O(|\Omega_i|r)$, to calculate $\sum_{j\in\Omega_i} v_{i,j}h_j$ for all the entries in Ω_i, $O(|\Omega_i|r^2)$ to build $\mathbf{B}_i$, and $O(r^3)$ to obtain $\mathbf{B}_i^{-1}$. Thus, updating every row of the two factor matrices, namely, $\{\mathbf{W}, \mathbf{H}\}$, which corresponds to a full ALS per training epoch, utilizes $O(2|\Omega|(r + r^2) + (m + n)r^3)$.

2) Multiplicative Update (MU): The update rule of MU is derived from GD [16]. With nonnegative initial $\mathbf{W}$ and $\mathbf{H}$ and autoadjusted training steps, the update rule can make the distance between $\mathbf{V}$ and $\widetilde{\mathbf{V}}$ monotonically decreasing and ensure that the nonnegativity constraints are satisfied for factor matrices $\{\mathbf{W}, \mathbf{H}\}$. There are two main approaches to utilizing MU for SNMF, which are as follows.

a) Autoadjusted training steps: The alternative approach of GD on $\{w_i, h_j\}$ are given by

$$\begin{cases} w_i \leftarrow w_i + \gamma_{\mathbf{W}} \circ \left(c_i - w_i\mathbf{B}_i\right) \\ h_j \leftarrow \mathbf{h}_j + \gamma_{\mathbf{H}} \circ \left(\overline{c}_j - h_j\overline{\mathbf{B}}_j\right). \end{cases} \tag{5}$$

respectively. If the training step is set as $\{\gamma_{\mathbf{W}} = h_j/(w_i\mathbf{B}_i), \gamma_{\mathbf{H}} = w_i/(h_j\overline{\mathbf{B}}_j)\}$, the negative items, namely, $\{w_i\mathbf{B}_i, h_j\overline{\mathbf{B}}_j\}$, in the update rule (5) of GD can be cancelled out.

b) Fitting with a quadratic function: The Taylor expansion of fitting functions $F_{\text{Eu}}(w_i, w_i^t)$ for approximating $d_{\text{Eu}}(w_i)$ is given by $F_{\text{Eu}}(w_i, w_i^t) = (d_{\text{Eu}})_i(w_i^t) + (w_i - w_i^t)(\partial(d_{\text{Eu}})_i/\partial w_i)^T + \frac{1}{2}(w_i - w_i^t)K(w_i)(w_i - w_i^t)^T$, where $K(w_i)$ are diagonal matrices with $K(w_i)_{k,k} = (w_i\mathbf{B}_i)_k/w_{i,k}$. $F_{\text{Eu}}(w_i^t, w_i^t) = (d_{\text{Eu}})_i(w_i^t)$ and $F_{\text{Eu}}(w_i, w_i^t) > (d_{\text{Eu}})_i(w_i)$ elsewhere. In addition, $F_{\text{Eu}}(w_i, w_i^t)$ is a strictly convex function. The optimal point, namely, w_i^o, for $F_{\text{Eu}}(w_i, w_i^t)$, which satisfy $\partial F(w_i^o, w_i^t)/\partial w_i^o = 0$, yields $F_{\text{Eu}}(w_i^o, w_i^t) \leq F_{\text{Eu}}(w_i^t, w_i^t)$. Thus, the monotonic decrease, which is expressed as $(d_{\text{Eu}})_i(w_i^o) \leq (d_{\text{Eu}})_i(w_i^t)$, and the nonnegativity constraints for w_i can be satisfied simultaneously [6]–[16]. By reversing the analysis of $\mathbf{W}$ and $\mathbf{H}$, using the approximate optimum, namely, h_j^o, which is obtained by the Taylor expansion of the fitting function $\overline{F}_{\text{Eu}}(h_j, h_j^t)$ to approximate $\overline{d}_{\text{Eu}}(h_j)$, the update rule for MU can be obtained as follows:

$$\begin{cases} w_i^o = w_i^t \circ \dfrac{\sum_{j\in\Omega_i} v_{i,j} h_j^t}{w_i^t \mathbf{B}_i} \\ h_j^o = h_j^t \circ \dfrac{\sum_{i\in\overline{\Omega}_j} v_{i,j} w_i^t}{h_j^t \overline{\mathbf{B}}_j}. \end{cases} \tag{6}$$

Updating row w_i for the factor matrix $\mathbf{W}$, for example, using (6), requires $O(|\Omega_i|(r + r^2) + r^2)$, which consists of $O(|\Omega_i|r)$, to calculate $\sum_{j\in\Omega_i} v_{i,j}h_j$ for all the entries in Ω_i; $O(|\Omega_i|r^2)$ to build $\mathbf{B}_i$; and $O(r^2)$ to obtain $w_i\mathbf{B}_i$. Thus, updating every row of the two factor matrices $\{\mathbf{W}, \mathbf{H}\}$, which corresponds to a full MU per training epoch, requires $O(2|\Omega|(r + r^2) + (m + n)r^2)$.

3) CCD++: Rewriting d_{Eu} in (3) in element-wise form, where $\|\mathbf{W}\|_F^2 = \sum_{i=0}^{m-1}\sum_{k=0}^{r-1} w_{i,k}^2$, $\|\mathbf{H}\|_F^2 = \sum_{j=0}^{n-1}\sum_{k=0}^{r-1} h_{j,k}^2$, and $\widetilde{v}_{i,j} = \sum_{k=0}^{r-1} w_{i,k}h_{j,k}$, and applying the alternative approach and GD on $\{w_{i,k}, h_{j,k}\}$, the update rules for CCD are obtained as

$$\begin{cases} w_{i,k} \leftarrow w_{i,k} - \gamma_{\mathbf{W}} \frac{\partial(d_{\text{Eu}})_i}{\partial w_{i,k}} \\ h_{j,k} \leftarrow h_{j,k} - \gamma_{\mathbf{H}} \frac{(\overline{d}_{\text{Eu}})_j}{\partial h_{j,k}} \end{cases} \tag{7}$$

respectively, where γ is the learning rate and the gradients $\{\partial(d_{\text{Eu}})_i/\partial w_{i,k}, \partial(\overline{d}_{\text{Eu}})_j/\partial h_{j,k}\}$ without constant 2 are given by $\{\sum_{j\in\Omega_i}(\widetilde{v}_{i,j} - v_{i,j})h_{j,k} + \lambda_{\mathbf{W}}w_{i,k}, \sum_{i\in\overline{\Omega}_j}(\widetilde{v}_{i,j} - v_{i,j})w_{i,k} + \lambda_{\mathbf{H}}h_{j,k}\}$, respectively. Based on the defining features of CCD, CCD++ updates $(\overline{w}_1, \overline{h}_1)$ until $(\overline{w}_r, \overline{h}_r)$ cyclically. Updating $\{\overline{w}_k, \overline{h}_k\}$ can be converted into updating $\{\overline{w}_{k,i}, \overline{h}_{k,j}\}$, respectively, in parallel [48]. The solutions for $\{\overline{w}_k, \overline{h}_k\}$ are reformulated as

$$\begin{cases} u \leftarrow \dfrac{\sum_{j\in\Omega_i}(v_{i,j} - \widehat{v}_{i,j})\overline{h}_{k,j}}{\lambda_{\mathbf{W}} + \sum_{j\in\Omega_i}\overline{h}_{k,j}^2}, \overline{w}_{k,i} \leftarrow u \\ v \leftarrow \dfrac{\sum_{j\in\Omega_i}(v_{i,j} - \widehat{v}_{i,j})\overline{w}_{k,i}}{\lambda_{\mathbf{W}} + \sum_{i\in\overline{\Omega}_j}\overline{w}_{k,i}^2}, \overline{h}_{k,j} \leftarrow v \end{cases} \tag{8}$$

respectively, where $\widehat{v}_{i,j} = (\widetilde{v}_{i,j} - \overline{w}_{k,i}\overline{h}_{k,j})$. CCD++ is a parallel approach on shared memory [48]. More recently, a parallelization extension of CCD++ was presented on a GPU [49]; however, no research on multi-GPU has been carried out.

D. Summary

Due to the limited space, the summary section has been removed to supplementary material.

III. Single-Thread-Based Generalized SNMF

In this section, a single-thread-based model for the generalized SNMF is proposed, which consists of the following parts:

1) *Update process* (see Section III-A): The update process for each feature vector $\{w_i, h_j\}$ moves along the index set, which is denoted as $\{\Omega_i, \overline{\Omega}_j\}$, and can cooperate with the related elements, namely, $\{\{v_{i,j}, h_j | j \in \Omega_i\}, \{v_{i,j}, w_i | i \in \overline{\Omega}_j\}\}$, respectively. The process follows a streamline-like style, namely, first-come-first-compute.

2) *Online approach* (see Section III-B): The streamline-like style gives SNMF online learning capability with linear scalability.

3) *Extension to big data platforms* (see Section III-C): The fine-grained parallelizability enables the single-thread-based model to be extended to GPU and multi-GPU with linear communication overhead.

A. Single-Thread-Based Model

1) Update Rule: The element-wise form of (1) with L_2 regularization can be written as

$$\begin{cases} \arg\min_{w_i,h_j} d_{\text{Eu}} = \sum_{(i,j)\in\Omega}(v_{i,j} - \widetilde{v}_{i,j})^2 \\ \qquad + \lambda_{\mathbf{W}}\sum_{k=0}^{r-1} w_{i,k}^2 + \lambda_{\mathbf{H}}\sum_{k=0}^{r-1} h_{j,k}^2 \\ \arg\min_{w_i,h_j} d_{\text{KL}} = \sum_{(i,j)\in\Omega}\left(\widetilde{v}_{i,j} - v_{i,j}\log(\widetilde{v}_{i,j})\right) \\ \qquad + \lambda_{\mathbf{W}}\sum_{k=0}^{r-1} w_{i,k}^2 + \lambda_{\mathbf{H}}\sum_{k=0}^{r-1} h_{j,k}^2 \\ \arg\min_{w_i,h_j} d_{\text{IS}} = \sum_{(i,j)\in\Omega}\left(\frac{v_{i,j}}{\widetilde{v}_{i,j}} + \log(\widetilde{v}_{i,j})\right) \\ \qquad + \lambda_{\mathbf{W}}\sum_{k=0}^{r-1} w_{i,k}^2 + \lambda_{\mathbf{H}}\sum_{k=0}^{r-1} h_{j,k}^2. \end{cases} \tag{9}$$

Similar to D_{Eu} in (3), the minimization problems for D_{KL} and D_{IS} can be split into many independent subproblems, $d_{\text{KL}} = \sum_{i=0}^{m-1}(d_{\text{KL}})_i$ and $\sum_{j=0}^{n-1}(\overline{d}_{\text{KL}})_j$, $d_{\text{IS}} = \sum_{i=0}^{m-1}(d_{\text{IS}})_i$ and $\sum_{j=0}^{n-1}(\overline{d}_{\text{IS}})_j$, where $(d_{\text{KL}})_i = \sum_{j\in\Omega_i}(v_{i,j} - \widetilde{v}_{i,j})^2 + \sum_{k=0}^{r-1} w_{i,k}^2$ and $(d_{\text{IS}})_i = \sum_{j\in\overline{\Omega}_i}\frac{v_{i,j}}{\widetilde{v}_{i,j}} + \log(\widetilde{v}_{i,j}) + \sum_{k=0}^{r-1} w_{i,k}^2$. Applying GD on $\partial(d_{\text{Eu}})_i/\partial w_{i,k}$, $\partial(d_{\text{KL}})_i/\partial w_{i,k}$, and $\partial(d_{\text{IS}})_i/\partial w_{i,k}$ without constant 2 are given by

$$\begin{cases} w_i \leftarrow w_i + \gamma\left(\sum_{j\in\Omega_i}(v_{i,j}h_{j,k} - \widetilde{v}_{i,j}h_{j,k}) - \lambda_{\mathbf{W}}w_{i,k}\right) \\ w_i \leftarrow w_i + \gamma\left(\sum_{j\in\Omega_i}(\frac{v_{i,j}}{\widetilde{v}_{i,j}}h_{j,k} - h_{j,k}) - \lambda_{\mathbf{W}}w_{i,k}\right) \\ w_i \leftarrow w_i + \gamma\left(\sum_{j\in\Omega_i}(\frac{v_{i,j}}{\widetilde{v}_{i,j}^2}h_{j,k} - \frac{h_{j,k}}{\widetilde{v}_{i,j}}) - \lambda_{\mathbf{W}}w_{i,k}\right). \end{cases} \tag{10}$$

When the learning rate γ for $\{d_{\text{Eu}}, d_{\text{KL}}, d_{\text{IS}}\}$ are set as $\{w_i/(\sum_{j\in\Omega_i}\widetilde{v}_{i,j}h_{j,k} + \lambda_{\mathbf{W}}w_{i,k}), w_i/(\sum_{j\in\Omega_i} h_{j,k} + \lambda_{\mathbf{W}}w_{i,k}), w_i/(\sum_{j\in\Omega_i}\frac{h_{j,k}}{\widetilde{v}_{i,j}} + \lambda_{\mathbf{W}}w_{i,k})\}$, respectively [14], the negative terms $\{-(\sum_{j\in\Omega_i}\widetilde{v}_{i,j}h_{j,k} + \lambda_{\mathbf{W}}w_{i,k}), -(\sum_{j\in\Omega_i} h_{j,k} + \lambda_{\mathbf{W}}w_{i,k}), -(\sum_{j\in\Omega_i}\frac{h_{j,k}}{\widetilde{v}_{i,j}} + \lambda_{\mathbf{W}}w_{i,k})\}$ can be cancelled out. By inverting $\mathbf{W}$ and $\mathbf{H}$, the similar conclusions are obtained. For simplification, let the intermediate values $\{c_{i,j,k}^{\text{Eu}}, c_{i,j,k}^{\text{KL}}, c_{i,j,k}^{\text{IS}}\}$ denote

Algorithm 1: Serial Version of the Single-thread-based Model for Updating $\mathbf{W}$ and $\mathbf{H}$ in each Training Epoch.

Input: Initial feature matrices $\mathbf{W}$ and $\mathbf{H}$, HiDS matrix $\mathbf{V}$, non-zeros indices set in row Ω, non-zeros indices set in column oriented $\overline{\Omega}$.

Output: $(\mathbf{W}, \mathbf{H})$

Set $Down \leftarrow 0$, $Up \leftarrow 0$;
for i *from* 0 *to* $m-1$ **do**
  for $j \in \Omega_i$ **do**
    $\widetilde{v}_{i,j} \leftarrow \sum_{k=0}^{r-1} w_{i,k} h_{j,k}$;
    for k *from* 0 *to* $r-1$ **do**
      Compute $c_{i,j,k}$ and $d_{i,j,k}$;
      $Up_{i,k} \leftarrow Up_{i,k} + c_{i,j,k}$;
      $Down_{i,k} \leftarrow Down_{i,k} + d_{i,j,k}$;
  for k *from* 0 *to* $r-1$ **do**
    $w_{i,k} \leftarrow w_{i,k}\big(Up_{i,k}/(Down_{i,k} + \lambda_{\mathbf{W}} w_{i,k})\big)$;
Set $Down \leftarrow 0$, $Up \leftarrow 0$;
for j *from* 0 *to* $n-1$ **do**
  for $i \in \overline{\Omega}_j$ **do**
    $\widetilde{v}_{i,j} \leftarrow \sum_{k=0}^{r-1} w_{i,k} h_{j,k}$;
    for k *from* 0 *to* $r-1$ **do**
      Compute $\overline{c}_{j,i,k}$ and $\overline{d}_{j,i,k}$;
      $Up_{j,k} \leftarrow Up_{j,k} + \overline{c}_{j,i,k}$;
      $Down_{j,k} \leftarrow Down_{j,k} + \overline{d}_{j,i,k}$;
  for k *from* 0 *to* $r-1$ **do**
    $h_{j,k} \leftarrow h_{j,k}\big(Up_{j,k}/(Down_{j,k} + \lambda_{\mathbf{H}} h_{j,k})\big)$;

$\{v_{i,j}h_{j,k}, (v_{i,j}/\widetilde{v}_{i,j})h_{j,k}, (v_{i,j}/\widetilde{v}_{i,j}^2)h_{j,k}\}$, respectively, and $\{d_{i,j,k}^{\text{Eu}}, d_{i,j,k}^{\text{KL}}, d_{i,j,k}^{\text{IS}}\}$ denote $\{\widetilde{v}_{i,j}h_{j,k}, h_{j,k}, (h_{j,k}/\widetilde{v}_{i,j})\}$, respectively. Let the intermediate values $\{\overline{c}_{j,i,k}^{\text{Eu}}, \overline{c}_{j,i,k}^{\text{KL}}, \overline{c}_{j,i,k}^{\text{IS}}\}$ denote $\{v_{i,j}w_{i,k}, (v_{i,j}/\widetilde{v}_{i,j})w_{i,k}, (v_{i,j}/\widetilde{v}_{i,j}^2)w_{i,k}\}$, respectively, and $\{\overline{d}_{j,i,k}^{\text{Eu}}, \overline{d}_{j,i,k}^{\text{KL}}, \overline{d}_{j,i,k}^{\text{IS}}\}$ denote $\{\widetilde{v}_{i,j}w_{i,k}, w_{i,k}, (w_{i,k}/\widetilde{v}_{i,j})\}$, respectively. Thus, the update rules for $\{d_{\text{Eu}}, d_{\text{KL}}, d_{\text{IS}}\}$ of the single-thread-based model can be expressed in a generalized form as

$$\begin{cases} w_{i,k} \leftarrow \dfrac{w_{i,k} \sum\limits_{j\in\Omega_i} c_{i,j,k}}{\sum\limits_{j\in\Omega_i} d_{i,j,k} + \lambda_{\mathbf{W}} w_{i,k}} \\ h_{j,k} \leftarrow \dfrac{h_{j,k} \sum\limits_{i\in\overline{\Omega}_j} \overline{c}_{j,i,k}}{\sum\limits_{i\in\overline{\Omega}_j} \overline{d}_{j,i,k} + \lambda_{\mathbf{H}} h_{j,k}} \end{cases} \tag{11}$$

where parameters $\{c_{i,j,k}, d_{i,j,k}\}$ are the generalized form of $\{\{c_{i,j,k}^{\text{Eu}}, c_{i,j,k}^{\text{KL}}, c_{i,j,k}^{\text{IS}}\}, \{d_{i,j,k}^{\text{Eu}}, d_{i,j,k}^{\text{KL}}, d_{i,j,k}^{\text{IS}}\}\}$, respectively, and $\{\overline{c}_{j,i,k}, \overline{d}_{j,i,k}\}$ are the generalized form of $\{\{\overline{c}_{j,i,k}^{\text{Eu}}, \overline{c}_{j,i,k}^{\text{KL}}, \overline{c}_{j,i,k}^{\text{IS}}\}, \{\overline{d}_{j,i,k}^{\text{Eu}}, \overline{d}_{j,i,k}^{\text{KL}}, \overline{d}_{j,i,k}^{\text{IS}}\}\}$, respectively. The serial version of the single-thread-based model is described in Algorithm 1. According to Algorithm 1, the single-thread-based model is transformed from the manipulations of the HiDS matrix and factor matrix to intermediate parameter operations and follows the order in Ω_i and $\overline{\Omega}_j$, which is the stream-like computation style.

2) Complexity Analysis: Theorem 1 (Time complexity of Algorithm 1 per training epoch): The time complexity of the single-thread-based model (Algorithm 1) is $O\big(6|\Omega|r + mr + nr\big)$.

Proof: The proof is omitted due to space limitation.

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

Proof: The proof is omitted due to space limitation.

Theorem 3 (Space complexity of CUSNMF): The space complexity of CUSNMF is $O\big(2|\Omega| + mr + nr\big)$.

Proof: The space complexity of the row and column compressed formats for the HiDS matrix $\mathbf{V}$ is $O(2|\Omega|)$. The space complexities of the factor matrices $\mathbf{W}$ and $\mathbf{H}$ are $O(mr)$ and $O(nr)$, respectively. In CUSNMF, the update tasks of the m feature vectors, namely, $\{w_i | i \in \{0,\ldots,m-1\}\}$, and the n feature vectors, namely, $\{h_j | j \in \{0,\ldots,n-1\}\}$, are allocated to the Tb CUDA thread blocks and the kth CUDA thread within a thread block updates $\{w_{i,k}, h_{j,k}\}$. $\{Down_{j,k}, Up_{j,k}\}$ are stored in local memory in the kth CUDA thread and $\sum_{k=0}^{r-1} w_{i,k}h_{j,k}$ can be solved by shared memory. More details will be presented in Section III-C. $\mathbf{W}$, $\mathbf{H}$, and the compressed format of the HiDS matrix $\mathbf{V}$ lie in global memory in CUSNMF. Thus, the space complexity of CUSNMF is $O\big(2|\Omega| + mr + nr\big)$.

3) Model Comparison: The comparisons are as follows.

a) Comparison with the model in Section II-B: The single-thread-based model can avoid forming intermediate matrices $\{\mathbf{G} \circ (\mathbf{W}\mathbf{H}^T), \mathbf{G} \circ \frac{\mathbf{V}}{\mathbf{W}\mathbf{H}^T}, \mathbf{G} \circ \frac{\mathbf{V}}{(\mathbf{W}\mathbf{H}^T)^2}, \mathbf{G} \circ \frac{1}{\mathbf{W}\mathbf{H}^T}\}$. Thus, the space complexity can be reduced from $O(6|\Omega| + mr + nr + 2\max(m,n)r)$ to $O(2|\Omega| + mr + nr + 2\max(m,n)r)$.

b) Comparison with MU in Section II-C2: The single-thread-based model for D_{Eu} can avoid forming intermediate Hessian matrices $\{\{\mathbf{B}_i | i \in \{0,\ldots,m-1\}\}, \{\overline{\mathbf{B}}_j | j \in \{0,\ldots,n-1\}\}\}$ and has the same function as in the original approach of MU. The time complexity can be reduced from $O(2|\Omega|(r + r^2) + (m+n)r^2)$ to $O\big(6|\Omega|r + mr + nr\big)$. Thus, the single-thread-based model has linear time complexity.

c) Comparison with CCD++ in Section II-C3: The single-thread-based model and CCD++ are derived from GD and the basic update element of the update rule only involves a feature element with in a feature vector. In addition, the single-thread-based model is a revised version of MU, which takes advantage of the convexity of the Bregman divergence function, e.g., d_{Eu}, d_{KL}, and d_{IS} [11]–[16], [23], [24]. Thus, the update process of each feature element within a feature vector takes the other feature elements information into consideration for the single-thread-based model. However, CCD++ is based on CCD and separate the correlations among those feature elements within a feature vector.

B. Online Learning

In this section, online learning in an incremental manner for the single-thread-based model is presented. The model of online learning in the DNMF [57]–[59] involves the reconstruction of Hessian matrices in an incremental manner. The extension from the static single-thread-based model to the dynamic and online learning model obeys the basic principle of the DNMF [57]–[59] to ensure the accuracy; meanwhile, it can avoid the reconstruction and inverse operations of Hessian matrices, which can extend the idea of the online learning in the DNMF [57]–[59] to SNMF with linear scalability. Without loss of generality, we follow the assumption [51],[56]–[59] that the row length (the number of users) is growing over time in CF problems, while the column size (the number of items) is constant over time. We briefly introduce new notations for online learning. An HiDS online matrix $\mathbf{V} \in \mathbb{R}_{+}^{(m^t+m^{t+1})\times n}$ is used for time from t to $t+1$, where $\mathbf{V}$ is expanded from $\mathbf{V}^t \in \mathbb{R}_{+}^{(m^t)\times n}$ by appending a new block of data $\mathbf{V}^{t+1} \in \mathbb{R}_{+}^{(m^{t+1})\times n}$. $\{\Omega^t, \overline{\Omega}^t\}$ and $\{\Omega_i^t, \overline{\Omega}_j^t\}$ are the corresponding index sets for $\mathbf{V}^t$. $\{\Omega^{t+1}, \overline{\Omega}^{t+1}\}$ and $\{\Omega_i^{t+1}, \overline{\Omega}_j^{t+1}\}$ are the corresponding index sets for $\mathbf{V}^{t+1}$. Considering that in most online systems, the size of the incoming data at time $t+1$ is usually much smaller than that of the current data before time t. Thus, we assume $m^{t+1} \ll m^t$. The online SNMF finds the temporal factor matrices $\mathbf{W}^{t+1}$ for time $t+1$, which are based on the matrices $\{\mathbf{V}^{t+1}, \mathbf{W}^{t+1}, \mathbf{H}\}$, where $\mathbf{W}^t \in \mathbb{R}_{+}^{m^t\times r}$ and $\mathbf{W}^{t+1} \in \mathbb{R}_{+}^{m^{t+1}\times r}$, to update the nontemporal $\mathbf{H}$ by $\{\mathbf{V}^t, \mathbf{V}^{t+1}, \mathbf{W}^t, \mathbf{W}^{t+1}, \mathbf{H}\}$ in a linearly scalable manner. The approximation function of the online problem is given by $\mathcal{D}(\mathbf{V}\|\widetilde{\mathbf{V}}) = \mathcal{D}(\mathbf{V}^t\|\widetilde{\mathbf{V}}^t) + \eta\mathcal{D}(\mathbf{V}^{t+1}\|\widetilde{\mathbf{V}}^{t+1})$, where η is the influence factor, which can measure the influence of the incoming elements at time $t+1$.

1) Update Temporal Factor Matrix $\mathbf{W}$*:* By fixing $\mathbf{H}$, we assume that the divergence $\mathcal{D}(\mathbf{V}^t\|\widetilde{\mathbf{V}}^t)$ is minimized and the factor matrices $\{\mathbf{W}^t, \mathbf{H}\}$ are updated. Thus, updating $\mathbf{W}^{t+1}$ is equivalent to minimizing $\mathcal{D}(\mathbf{V}^{t+1}\|\widetilde{\mathbf{V}}^{t+1})$. As a result, $\mathbf{W}$ is updated by appending the projection $\mathbf{W}^{t+1}$ of $\mathbf{V}^{t+1}$ via loading the factor matrices $\mathbf{H}$ of the previous time step. The update rule for $\mathbf{W}^{t+1}$ in element-wise form for online learning is given by

$$w_{i,k}^{t+1} \leftarrow \frac{w_{i,k}^{t+1}\sum_{j\in\Omega_i^{t+1}} c_{i,j,k}^{t+1}}{\sum_{j\in\Omega_i^{t+1}} d_{i,j,k}^{t+1} + \lambda_{\mathbf{W}} w_{i,k}^{t+1}} \tag{12}$$

where $(i,j) \in \Omega^{t+1}$ and $\widetilde{v}_{i,j}^{t+1} = \sum_{k=0}^{r-1} w_{i,k}^{t+1} h_{j,k}$. The intermediate parameters $\{c_{i,j,k}^{t+1}, d_{i,j,k}^{t+1}\}$ are chosen according to the divergence type. For $\{\mathcal{D}_{\text{Eu}}, \mathcal{D}_{\text{KL}}, \mathcal{D}_{\text{IS}}\}$, $c_{i,j,k}^{t+1}$ are denoted as $\{v_{i,j}^{t+1}h_{j,k}, (v_{i,j}^{t+1}/\widetilde{v}_{i,j}^{t+1})h_{j,k}, (v_{i,j}^{t+1}/(\widetilde{v}_{i,j}^{t+1})^2)h_{j,k}\}$, respectively, and $d_{i,j,k}^{t+1}$ for $\{\mathcal{D}_{\text{Eu}}, \mathcal{D}_{\text{KL}}, \mathcal{D}_{\text{IS}}\}$ are denoted as $\{\widetilde{v}_{i,j}^{t+1}h_{j,k}, h_{j,k}, (h_{j,k}/\widetilde{v}_{i,j}^{t+1})\}$, respectively.

2) Update Nontemporal Factor Matrix $\mathbf{H}$*:* We rewrite the online problem as

$$\begin{aligned}\arg\min_{h_j} \overline{d}_j(h_j) = &\sum_{i\in\overline{\Omega}_j^t} \mathcal{D}(v_{i,j}^t\|\widetilde{v}_{i,j}^t) + \eta \sum_{i\in\overline{\Omega}_j^{t+1}} \mathcal{D}(v_{i,j}^{t+1}\|\widetilde{v}_{i,j}^{t+1})\\ &+ \lambda_{\mathbf{H}}\|h_j\|_F^2\end{aligned} \tag{13}$$

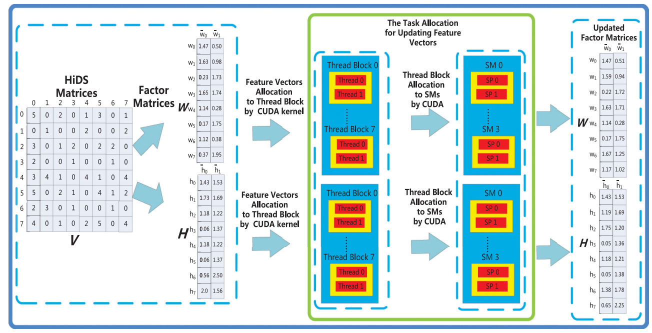


Fig. 1. Toy example of CUSNMF.

where $\mathcal{D}(v_{i,j}\|\widetilde{v}_{i,j})$ for $\{\mathcal{D}_{\text{Eu}}, \mathcal{D}_{\text{KL}}, \mathcal{D}_{\text{IS}}\}$ are denoted as $\{(v_{i,j} - \widetilde{v}_{i,j})^2, \widetilde{v}_{i,j} - v_{i,j}\log(\widetilde{v}_{i,j}), \frac{v_{i,j}}{\widetilde{v}_{i,j}} + \log(\widetilde{v}_{i,j})\}$, respectively. Applying GD to minimize the error $\overline{d}_j(h_j)$ can be written as

$$\begin{aligned}h_j \leftarrow h_j + \gamma_{\mathbf{W}}\Bigg(&\sum_{i\in\overline{\Omega}_j^t} \overline{c}_{j,i,k}^t + \eta \sum_{i\in\overline{\Omega}_j^{t+1}} \overline{c}_{j,i,k}^{t+1}\\ &- \Big(\sum_{i\in\overline{\Omega}_j^t} \overline{d}_{j,i,k}^t + \eta \sum_{i\in\overline{\Omega}_j^{t+1}} \overline{d}_{j,i,k}^{t+1} + \lambda_{\mathbf{H}} h_{j,k}\Big)\Bigg)\end{aligned} \tag{14}$$

where $\overline{c}_{j,i,k}^{t+1}$ are denoted as $\{v_{i,j}^{t+1}w_{i,k}^{t+1}, (v_{i,j}^{t+1}/\widetilde{v}_{i,j}^{t+1})w_{i,k}^{t+1}, (v_{i,j}^{t+1}/(\widetilde{v}_{i,j}^{t+1})^2)w_{i,k}^{t+1}\}$, respectively, for $\{\mathcal{D}_{\text{Eu}}, \mathcal{D}_{\text{KL}}, \mathcal{D}_{\text{IS}}\}$; $\overline{d}_{j,i,k}^{t+1}$ are denoted as $\{\widetilde{v}_{i,j}^{t+1}w_{i,k}^{t+1}, w_{i,k}^{t+1}, (w_{i,k}^{t+1}/\widetilde{v}_{i,j}^{t+1})\}$, respectively, for $\{\mathcal{D}_{\text{Eu}}, \mathcal{D}_{\text{KL}}, \mathcal{D}_{\text{IS}}\}$. When the learning rate $\gamma_{\mathbf{H}}$ is set as $h_j/(\sum_{i\in\overline{\Omega}_j^t} \overline{d}_{j,i,k}^t + \eta\sum_{i\in\overline{\Omega}_j^{t+1}} \overline{d}_{j,i,k}^{t+1} + \lambda_{\mathbf{H}}h_{j,k})$, the negative terms $-(\sum_{i\in\overline{\Omega}_j^t} \overline{d}_{j,i,k}^t + \eta\sum_{i\in\overline{\Omega}_j^{t+1}} \overline{d}_{j,i,k}^{t+1} + \lambda_{\mathbf{H}}h_{j,k})$ can be cancelled out. The update rule of $\mathbf{H}$ in element-wise form for online learning for the single-thread-based model is given by

$$h_{j,k} \leftarrow h_{j,k}\frac{\sum_{i\in\overline{\Omega}_j^t} \overline{c}_{j,i,k}^t + \eta\sum_{i\in\overline{\Omega}_j^{t+1}} \overline{c}_{j,i,k}^{t+1}}{\sum\limits_{i\in\overline{\Omega}_j^t} \overline{d}_{j,i,k}^t + \eta \sum\limits_{i\in\overline{\Omega}_j^{t+1}} \overline{d}_{j,i,k}^{t+1} + \lambda_{\mathbf{H}}h_{j,k}}. \tag{15}$$

C. CUSNMF and MCUSNMF

1) CUDA Parallelization: The single-thread-based model gives the generalized SNMF fine-grained parallelizability. First, according to Algorithm 1, the outer loop for updating $\mathbf{W}$ (Lines 2–10) can be divided into m independent parts and the outer loop for updating $\mathbf{H}$ (Lines 12–20) can be divided into n independent parts. Second, the precomputation of $\widetilde{v}_{i,j} = \sum_{k=0}^{r-1} w_{i,k}h_{j,k}$ can be accomplished via the cooperation of the shared memory within a thread block and the r threads within the thread block. With the precomputed $\widetilde{v}_{i,j}$, the inner loop for updating $\mathbf{W}$ (Lines 5–8) can be separated into r independent parts, and the inner loop for updating $\mathbf{H}$ (Lines 15–18) can be separated into r independent parts.

A toy example of CUSNMF is shown in Fig. 1. As Fig. 1 show, the tasks for factorizing a sparse matrix $\mathbf{V} \in \mathbb{R}_{+}^{8\times 8}$ into two factor matrices, namely, $\mathbf{W} \in \mathbb{R}_{+}^{8\times 2}$ and $\mathbf{H} \in \mathbb{R}_{+}^{8\times 2}$, are allocated to eight thread blocks and a GPU has four SMs with two SPs per

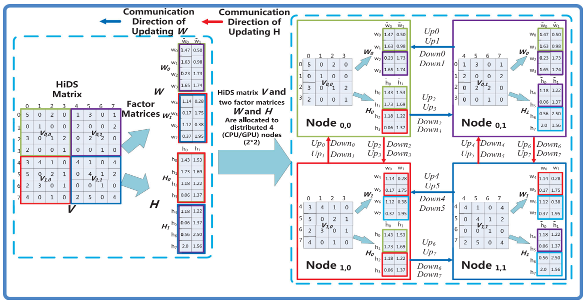


Fig. 2. Toy example of MCUSNMF on four (CPU/GPU) nodes.

SM. Considering the updating of $\mathbf{W}$ as an example, a GPU updates $\mathbf{W}$, the eight feature vectors $\{w_0, \ldots, w_7\}$ are allocated to the eight thread blocks $\{0, \ldots, 7\}$, and the thread k within thread block i updates $w_{i,k}$. The eight thread blocks are allocated to four SMs automatically by CUDA. Thus, the more SMs a GPU has, the higher the computing speed of the GPU.

2) Multi-GPU Model: We extend CUSNMF to MCUSNMF when size a dataset may exceed the memory capacity of a single GPU. The data partition on multi-GPU utilizes the maximal locality of the HiDS matrix $\mathbf{V}$, which can minimize the communication overhead, which is the same as that of HPC-NMF [50]. However, the communication overhead of MCUSNMF is much lower than of HPC-NMF. Given $p \times q$ GPUs, we divide $\mathbf{V}$ into p parts, i.e., $\{\mathbf{V}_0, \ldots, \mathbf{V}_{p-1}\}$ by row, and the lth part of $\mathbf{V}_l$ is divided into q subparts, i.e., $\{\mathbf{V}_{l,0}, \ldots, \mathbf{V}_{l,q-1}\}$. We divide $\mathbf{W}$ into p parts, i.e., $\{\mathbf{W}_0, \ldots, \mathbf{W}_{p-1}\}$, and the lth part of $\mathbf{W}_l$ is divided into q parts, i.e., $\{\mathbf{W}_{l,0}, \ldots, \mathbf{W}_{l,q-1}\}$. We divide $\mathbf{H}$ into q parts, i.e., $\{\mathbf{H}_0, \ldots, \mathbf{H}_{q-1}\}$, and the oth part of $\mathbf{H}_o$ is divided into p parts, i.e., $\{\mathbf{H}_{o,0}, \ldots, \mathbf{H}_{o,p-1}\}$. Then, we load $\{\mathbf{W}_l, H_o, \mathbf{V}_{l,o}\}$ to GPU $_{l,o}$ in the initial step. Taking updating $\mathbf{W}$ as an example, each GPU $_{l,o}$ computes local intermediate parameters $\{Up_l, Down_l\}$ and sends the local intermediate parameters $\{Up_{l,o}, Down_{l,o}\}$ to GPU $_{l,o}$, where $o \in \{0, \ldots, q-1\}$. However, HPC-NMF and cuMF [45] must compute and send each local Hessian $\mathbf{B}_i$ to other nodes; thus, MCUSNMF can reduce the communication overhead from $O(m\frac{q-1}{q}r^2 + n\frac{p-1}{p}r^2)$ to $O(m\frac{q-1}{q}r + n\frac{p-1}{p}r)$.

A toy example of MCUSNMF is shown in Fig. 2. According to Fig. 2, the local matrices $\{\{\mathbf{V}_{0,0}, \mathbf{W}_0, \mathbf{H}_0\}, \{\mathbf{V}_{0,1}, \mathbf{W}_0, \mathbf{H}_1\}, \{\mathbf{V}_{1,0}, \mathbf{W}_1, \mathbf{H}_0\}, \{\mathbf{V}_{1,1}, \mathbf{W}_1, \mathbf{H}_1\}\}$ are allocated to $\{GPU_{0,0}, GPU_{0,1}, GPU_{1,0}, GPU_{1,1}\}$, respectively. We take updating $\mathbf{W}$ as an example. $\{GPU_{0,1}, GPU_{0,0}\}$ send intermediate parameters $\{\{Up_0, Up_1\}, \{Down_0, Down_1\}\}$ and $\{\{Up_2, Up_3\}, \{Down_2, Down_3\}\}$ to $\{GPU_{0,0}, GPU_{0,1}\}$, respectively, to update $\mathbf{W}_0$. $\{GPU_{1,1}, GPU_{1,0}\}$ send intermediate parameters $\{\{Up_4, Up_5\}, \{Down_4, Down_5\}\}$ and $\{\{Up_6, Up_7\}, \{Down_6, Down_7\}\}$ to $\{GPU_{1,0}, GPU_{1,1}\}$, respectively, to update $\mathbf{W}_1$.

IV. EXPERIMENTS

The experimental settings are presented in supplementary material. We performed experiments with the objective answering the following questions.

TABLE II
OCCUPANCY OF CUSNMF ON P100 GPU WITH DOUBLE-PRECISION FLOATING-POINT ARITHMETIC

Rank r	32	64	128	256	512	1024
Registers per thread	21	21	21	21	21	21
Shared memory per block(bytes)	256	512	1024	2048	4096	8192
Active thread blocks per SM	32	32	16	8	4	2
Occupancy per SM	50%	100%	100%	100%	100%	100%

1) *Q1: Scalability (see Section IV-A).* How does CUSNMF scale with regard to various conditions, e.g., the rank of the feature matrix or different input HiDS matrices?
2) *Q2: Convergence (see Section IV-B).* How quickly and accurately do CUSNMF, cuMF, HPC-NMF, and CCD++ factorize the real-world HiDS matrices and how much does MCUSNMF improve the speed of CUSNMF?
3) *Q3: Online learning (see Section IV-C).* What levels of correctness and efficiency are achieved with online learning?

A. Scalability

We denote $\mathcal{D}_{\text{Eu}}$, $\mathcal{D}_{\text{KL}}$, and $\mathcal{D}_{\text{IS}}$ of generalized MU as GMU_{Eu}, GMU_{KL}, and GMU_{IS}; $\mathcal{D}_{\text{Eu}}$ is denoted as MU_{Eu}; $\mathcal{D}_{\text{Eu}}$, $\mathcal{D}_{\text{KL}}$, and $\mathcal{D}_{\text{IS}}$ of the single-thread-based model for the SNMF are denoted as SNMF_{Eu}, SNMF_{KL}, and SNMF_{IS}; $\mathcal{D}_{\text{Eu}}$, $\mathcal{D}_{\text{KL}}$, and $\mathcal{D}_{\text{IS}}$ of CUSNMF are denoted as $\text{CUSNMF}_{\text{Eu}}$, $\text{CUSNMF}_{\text{KL}}$, and $\text{CUSNMF}_{\text{IS}}$; and similar for MCUSNMF.

The process of obtaining scalability on CPU and GPU includes the following:

1) the process of parameter selection for CUSNMF, which refer to the GPU *occupancy*;
2) space requirements and running time of SNMF_{Eu}, SNMF_{KL}, and SNMF_{IS}.

Choosing an appropriate value of r for the feature matrices can guarantee the reasonable accuracy and reduce the training time that is spent in the CF MF problems [25]. In our work, because we focus on the GPU acceleration performance for various value of r for feature matrices rather than on choosing an appropriate value of r, we conduct five sets of experiments with various values of r, e.g., $r \in \{32, 64, 128, 256, 512, 1024\}$ or $\log_2(r) \in \{5, 6, 7, 8, 9, 10\}$. GPU *occupancy* is the ratio of the number of active threads to the total number of threads; high *occupancy* means that the GPU is working with high efficiency. In CUSNMF, a thread block has r threads and the number of thread blocks is tunable, through which the *occupancy* can be controlled. Table II lists the *occupancy* under various value of r, namely, $\log_2(r) \in \{5, 6, 7, 8, 9, 10\}$, and we set the optimal number of thread blocks as $\{1792, 1792, 896448, 224112\}$, which is the number of active thread block per SM multiplying the number of SMs ($\{32, 32, 16, 8, 4, 2\} \cdot 56$).

We measure the scalability of SNMF_{Eu}, SNMF_{KL}, and SNMF_{IS} in terms of the rank of the feature matrices and the scale of the input sparse matrix and we test the scalability with regard to double-precision floating-point arithmetic. As shown in Fig. 3, the memory requirements of SNMF_{Eu}, SNMF_{KL}, and

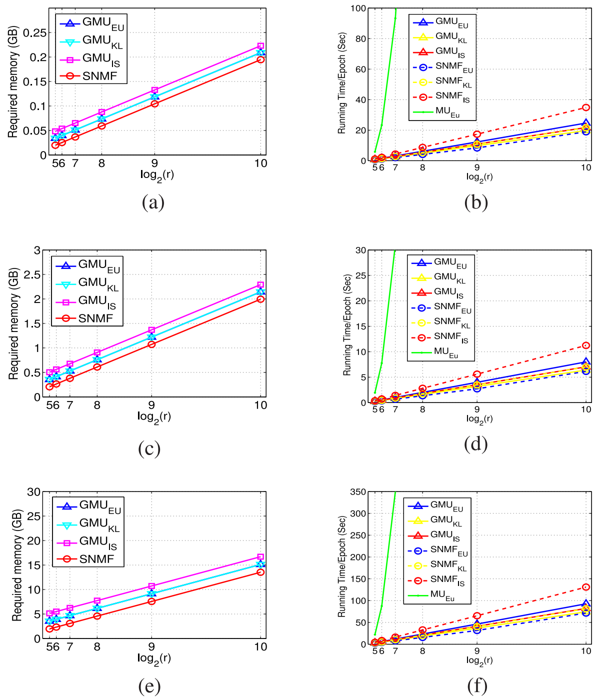


Fig. 3. Memory scalability: (a) (MovieLens-1 M), (c) (MovieLens-10 M), and (e) (Netflix-100 M). Running time scalability: (b) (MovieLens-1 M), (d) (MovieLens-10 M), and (f) (Netflix-100 M) with respect to the rank of the feature matrices r and the scale of the dataset.

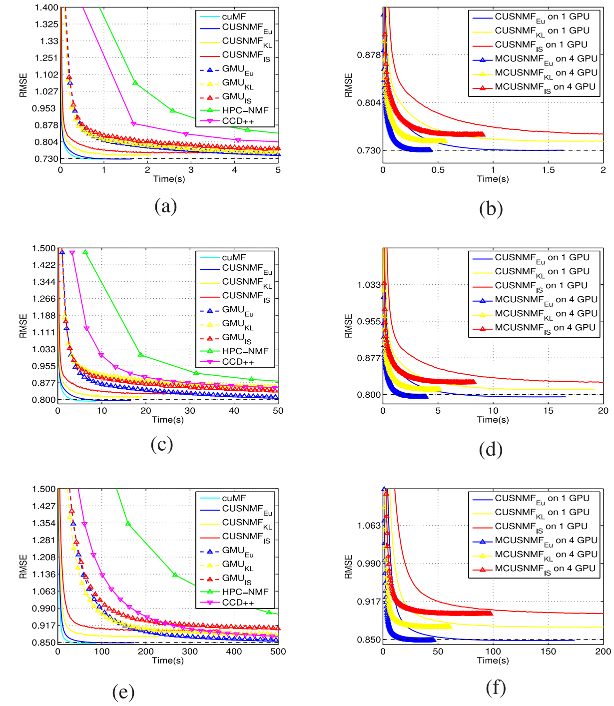


Fig. 4. Running time versus RMSE: (a) (MovieLens-1 M), (c) (MovieLens-10 M), and (e) (Netflix-100 M). Multi-GPU performance: (b) (MovieLens-1 M), (d) (MovieLens-10 M), and (f) (Netflix-100 M) for rank $r = 100$.

SNMF_{IS} [see Fig. 3(a), (c), and (e)] scale with r and the volume of the datasets. With the same space requirement, Fig. 3(b), (d), and (f) illustrates that the computational overhead increase linearly with both r and the volume of the input sparse matrix. Furthermore, SNMF_{Eu} adopts a convex optimization strategy and transforms the Moore Inverses $\left(\sum_{j\in\Omega_i} h_j^T h_j + \lambda_{\mathbf{W}} I\right)^{-1}$ and $\left(\sum_{i\in\overline{\Omega}_j} w_i^T w_i + \lambda_{\mathbf{H}} I\right)^{-1}$ into the diagonal matrix inverse operations $K(w_i)^{-1}$ and $\overline{K}(h_j)^{-1}$, respectively (see Sections II-C1 and II-C2). SNMF_{Eu} has the same function as MU_{Eu} and linear computational complexity (see Section III-A1). Thus, as Fig. 3(b), (d), and (f) shows, SNMF_{Eu} can reduce the cubic overhead of the ALS and quadratic computational overhead of MU_{Eu} to linear cost.

B. Convergence and Multi-GPU

We compare how quickly and accurately each method factorizes real-world HiDS matrices. Fig. 4(a), (c), and (e) illustrates the running time versus RMSE. As Fig. 4(a), (c), and (e) shows, in all datasets, the ALS converges fastest to the baseline accuracy, followed by SNMF_{Eu}; CCD++ converges slowest. Because the maximum value for the rank r of cuMF is $r = 100$, cuMF accelerates the ALS on GPU, and runs 2X faster than CUSNMF with rank $r = 100$; however, cuMF of ALS requires cubic time complexity, and CUSNMF has much better linear scalability than the ALS does, as discussed in Sections II-C1, II-C2, and III-A. SNMF_{KL} and SNMF_{IS} cannot obtain comparable accuracy to that of SNMF_{Eu}; we deduce that the probabilistic distributions of the three datasets are apt to $\mathcal{G}$ aussian distributions. In image clustering communities, NMF in $\mathcal{D}_{\text{KL}}$ outperforms NMF in $\mathcal{D}_{\text{Eu}}$ [17].

We evaluate the performance of MCUSNMF on 4 P100 GPUs. Multi-GPU can solve the problem that large-scale datasets cannot be loaded onto a single GPU. MCUSNMF can maintain the maximal locality of a submatrix, which can minimize the communication overhead. According to Fig. 4(b), (d), and (f), we conclude the following.

1) The reduced parallelism and unbalanced load of each SMs, which is due to the irregular distribution of nonzero entries in the rating matrix $\mathbf{V}$, may cause some SMs to be idle during the update process, which may lead to the sublinear speedup.
2) The speedup of four GPUs versus one GPU increases with the scale of the dataset, e.g., 3.7X on Movielen-1 M, 3.74X on Movielen-10 M and 3.76X on Netflix-100 M.

We conclude that the reason is because the ratio of the communication overhead to the scale of the HiDS matrix is inversely proportional to the scale of the HiDS matrix.

C. Online Learning

In online learning of a generalized SNMF, new data can be projected into the low-rank space that has already been determined, and slight adjustments can be made. The computational overhead is far less than that of the process of retraining on the combined data. The Movielen and Netflix datasets contain spatio-temporal information, and they give online learning for

TABLE III
INFLUENCE OF η ON SNMF_{Eu} IN OBTAINING A BASELINE RMSE

Data sets	CUSNMF_{Eu}	SNMF_{Eu}	Online $\eta = 0.5$	Online $\eta = 1.0$	Online $\eta = 1.5$	Online $\eta = 2.0$
Movielen-1M	1.15	7.87	0.00371	0.00368	0.00371	0.00364
Movielen-10M	7.48	43.96	0.02450	0.02425	0.02439	0.02400
Netflix-100M	75.69	567.34	0.2812	0.2770	0.2820	0.2806

generalized SNMF real significance. Table III presents performance comparisons between online learning and retraining when new data are added into HiDS matrices. We observe that with the trained low-rank factor matrices $\{\mathbf{W}^t, \mathbf{H}\}$, the update process for $\mathbf{W}^{t+1}$ and slight adjustment for $\mathbf{H}$ require much less time than that for $\text{CUSNMF}_{\text{Eu}}$ on P100 GPU and SNMF_{Eu} on OpenMP and obtain the same baseline accuracy value, which demonstrates the efficiency and correctness of online learning.

V. CONCLUSION

A. Summary of the Single-Thread-Based Model

This article focused on designing a single-thread-based model for generalized SNMF, which could decompose manipulations of whole feature matrices into the involved feature element operations and has fine-grained parallelizability inherence. Meanwhile, the model had a streamline-like computing style, which could cater to the computing characteristics of the mainstream big data and industrial platforms, e.g., GPU, Spark, and Flink. Furthermore, the streamline-like computing style could realize the online learning and fine-grained parallelization with CUDA parallelization ability on GPU (CUSNMF) and multi-GPU (MCUSNMF). CUSNMF achieved at least 7X speedup on P100 GPU compared to that of SNMF, CCD++, and HPC-NMF on OpenMP. cuMF, which accelerates the ALS on GPU, runs 2X faster than CUSNMF does with rank $r = 100$; however, CUSNMF had the advantages of linear computational scalability compared to the cubic complexity of cuMF.

B. Industrial Usage

Industrial informatization depends on the techniques maturity of mathematical application and the promotion of industrial platforms for big data analysis. Generalized SNMF is a useful dimension reduction technique and plays an important role in large-scale data analysis due to the identity for some styles of data, i.e., low-rank, nonnegativity, sparsity, and various styles of probabilistic distribution, over the past few decades [6]–[16]. The generalized SNMF takes only the combination of the two factor matrices to represent the original matrix for clustering, missing value prediction, anomaly detection and has been widely used in industrial informatics applications, e.g., bioinformatics, recommender systems, network traffic analysis, social network analysis, etc., [18]–[24],[26]–[31]. The single-thread-based model is an optimized generalized SNMF, and the training process of it depends only on the involved feature elements operations and has fine-grained parallelizing inherence and streamline-like computing style, rather than the whole large-scale matrices manipulations. Thus, it has the following contributions for industrial informatics communities:

1) low space requirements, time complexity, and simplification on programming;
2) the more convenience for implementation on big data and industrial platforms, e.g., GPU, Spark, and Flink;
3) the potential on online learning for incremental data.

C. Future Work

We observe that, on the one hand, the real-world data in industrial applications, e.g., bioinformatics, recommender systems, and social networks, may include not only the HiDS matrix but also other information, e.g., geographical and spatio-temporal attributes or disease-associated information, [26], [27], [51], [52]. Thus, to describe those data more accurately, regularization items must be added into the optimization formulae, e.g., graph regularization on manifold learning or weight matrices for implicit and explicit information [9], [15], [17], [26], [27], [51], [52]. On the other hand, in optimization communities, in addition to L_2 regularization, other norm, e.g., spectral and kernel norms, are commonly encountered [19], [31], [51]. However, the aforementioned operations will increase the memory and computational overheads substantially. Thus, decreasing the memory requirement and computational complexity can simplify the computational process, and then, it can promote the rapid development of NMF. More recently, deep learning is a rapid emerging technique for CF problems, which can extract the features of HiDS matrices in a nonlinear and deeper manner; thus, it can obtain a higher accuracy than SNMF [21], [22]; however, it runs much longer than SNMF. Thus, we want to explore the acceleration approach from the views of optimization and parallel and distributed computing. Furthermore, the implantation of MCUSNMF on Spark and Flink will be developed, which can complement each other.

ACKNOWLEDGMENT

The authors would like to thank the three anonymous reviewers to improve the quality of this work.

REFERENCES

[1] I. Stoica, "Trends and challenges in big data processing," *Proc. VLDB Endowment*, vol. 9, no. 13, pp. 1619–1619, 2016.
[2] Z. Lv, H. Song, P. Basanta-Val, A. Steed, and M. Jo, "Next-generation big data analytics: State of the art, challenges, and future research topics," *IEEE Trans. Ind. Informat.*, vol. 13, no. 4, pp. 1891–1899, Aug. 2017.
[3] P. Basanta-Val, "An efficient industrial big-data engine," *IEEE Trans. Ind. Informat.*, vol. 14, no. 4, pp. 1361–1369, Apr. 2018.
[4] J. Zhu, Z. Ge, and Z. Song, "Distributed parallel PCA for modeling and monitoring of large-scale plant-wide processes with big data," *IEEE Trans. Ind. Informat.*, vol. 13, no. 4, pp. 1877–1885, Aug. 2017.

[5] L. Kuang, F. Hao, L. T. Yang, M. Lin, C. Luo, and G. Min, "A tensor-based approach for big data representation and dimensionality reduction," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 3, pp. 280–291, Sep. 2014.

[6] J. Li, J. M. Bioucas-Dias, A. Plaza, and L. Liu, "Robust collaborative nonnegative matrix factorization for hyperspectral unmixing," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 10, pp. 6076–6090, Oct. 2016.

[7] W. He, H. Zhang, and L. Zhang, "Sparsity-regularized robust non-negative matrix factorization for hyperspectral unmixing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 9, pp. 4267–4279, Sep. 2016.

[8] K. Huang, N. D. Sidiropoulos, and A. P. Liavas, "A flexible and efficient algorithmic framework for constrained matrix and tensor factorization," *IEEE Trans. Signal Process.*, vol. 64, no. 19, pp. 5052–5065, Oct. 2016.

[9] X. Li, G. Cui, and Y. Dong, "Graph regularized non-negative low-rank matrix factorization for image clustering," *IEEE Trans. Cybern.*, vol. 47, no. 11, pp. 3840–3853, Nov. 2017.

[10] H. Li, K. Li, J. An, W. Zheng, and K. Li, "An efficient manifold regularized sparse non-negative matrix factorization model for large-scale recommender systems on GPUs," *Inf. Sci.*, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025518305875

[11] C. Liu, H.-c. Yang, J. Fan, L.-W. He, and Y.-M. Wang, "Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 681–690.

[12] S. Sra and I. S. Dhillon, "Generalized nonnegative matrix approximations with Bregman divergences," in *Proc. Adv. Neural Inf. Process. Syst.*, 2005, pp. 283–290.

[13] I. S. Dhillon and J. A. Tropp, "Matrix nearness problems with Bregman divergences," *SIAM J. Matrix Anal. Appl.*, vol. 29, no. 4, pp. 1120–1146, 2008.

[14] Y. Yuan and X. Luo, "Performance of nonnegative latent factor models with β-distance functions in recommender systems," in *Proc. IEEE 15th Int. Conf. Netw., Sens. Control*, 2018, pp. 1–7.

[15] H. Li, K. Li, J. An, and K. Li, "Cusntf: A scalable sparse non-negative tensor factorization model for large-scale industrial applications on multi-GPU," in *Proc. 27th ACM Int. Conf. Inf. Knowl. Manage*, 2018, pp. 1113–1122. [Online]. Available: http://doi.acm.org/10.1145/3269206.3271749

[16] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2001, pp. 556–562.

[17] N. Guan, D. Tao, Z. Luo, and B. Yuan, "Manifold regularized discriminative nonnegative matrix factorization with fast gradient descent," *IEEE Trans. Image Process.*, vol. 20, no. 7, pp. 2030–2048, Jul. 2011.

[18] X. Luo *et al.*, "An efficient second-order approach to factorize sparse matrices in recommender systems," *IEEE Trans. Ind. Informat.*, vol. 11, no. 4, pp. 946–956, Aug. 2015.

[19] K. Xie *et al.*, "Recover corrupted data in sensor networks: A matrix completion solution," *IEEE Trans. Mobile Comput.*, vol. 16, no. 5, pp. 1434–1448, May 2017.

[20] E. Baccarelli, M. Scarpiniti, P. G. V. Naranjo, and L. Vaca-Cardenas, "Fog of social IoT: When the fog becomes social," *IEEE Netw.*, vol. 32, no. 4, pp. 68–80, Jul./Aug. 2018.

[21] H. Wang and D. Y. Yeung, "Towards Bayesian deep learning: A framework and some existing methods," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 12, pp. 3395–3408, Dec. 2016.

[22] X. Luo, M. Zhou, S. Li, and M. Shang, "An inherently nonnegative latent factor model for high-dimensional and sparse matrices from industrial applications," *IEEE Trans. Ind. Informat.*, vol. 14, no. 5, pp. 2011–2022, May 2018.

[23] S. Zhang, W. Wang, J. Ford, and F. Makdon, "Learning from incomplete ratings using non-negative matrix factorization," in *Proc. SIAM Int. Conf. Data Mining*, 2006, pp. 43–47.

[24] Y.-D. Kim and S. Choi, "Weighted nonnegative matrix factorization," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2009, pp. 1541–1544.

[25] A. Paterek, "Improving regularized singular value decomposition for collaborative filtering," in *Proc. KDD Cup Workshop*, 2007, pp. 5–8.

[26] A. Ezzat, P. Zhao, M. Wu, X.-L. Li, and C.-K. Kwoh, "Drug-target interaction prediction with graph regularized matrix factorization," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 14, no. 3, pp. 646–656, May–Jun. 2017.

[27] Y. Zhong *et al.*, "A non-negative matrix factorization based method for predicting disease-associated miRNAs in miRNA-disease bilayer network," *Bioinformatics*, vol. 34, no. 2, pp. 267–277, 2018.

[28] X. Luo, J. Sun, Z. Wang, S. Li, and M. Shang, "Symmetric and nonnegative latent factor models for undirected, high dimensional and sparse networks in industrial applications," *IEEE Trans. Ind. Informat.*, vol. 13, no. 6, pp. pp. 3098–3107, Dec. 2017.

[29] M. Congosto, P. Basanta-Val, and L. Sanchez-Fernandez, "T-hoarder: A framework to process twitter data streams," *J. Netw. Comput. Appl.*, vol. 83, pp. 28–39, 2017.

[30] R. Kannan, H. Woo, C. C. Aggarwal, and H. Park, "Outlier detection for text data," in *Proc. SIAM Int. Conf. Data Mining*, 2017, pp. 489–497.

[31] G. Xie, K. Xie, J. Huang, X. Wang, Y. Chen, and J. Wen, "Fast low-rank matrix approximation with locality sensitive hashing for quick anomaly detection," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[32] D. Schneider, "Us supercomputing strikes back," *IEEE Spectr.*, vol. 55, no. 1, pp. 52–53, Jan. 2018.

[33] P. Basanta-Val, N. C. Audsley, A. J. Wellings, I. Gray, and N. Fernandezgarcia, "Architecting time-critical big-data systems," *IEEE Trans. Big Data*, vol. 2, no. 4, pp. 310–324, Dec. 2016.

[34] M. Shojafar, N. Cordeschi, and E. Baccarelli, "Energy-efficient adaptive resource management for real-time vehicular cloud services," *IEEE Trans. Cloud Comput.*, vol. 7, no. 1, pp. pp. 196–209, Jan.–Mar. 2019.

[35] M. T. Higuera-Toledano, "Java technologies for cyber-physical systems," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 680–687, Apr. 2017.

[36] T. Higuera, J. L. R. Martin, P. Arroba, and J. L. Ayala, "Green adaptation of real-time web services for industrial CPS within a cloud environment," *IEEE Trans. Ind. Informat.*, vol. 13, no. 3, pp. 1249–1256, Jun. 2017.

[37] N. Cordeschi, D. Amendola, and E. Baccarelli, "Reliable adaptive resource management for cognitive cloud vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 64, no. 6, pp. 2528–2537, Jun. 2015.

[38] J. Wu, L. Deng, and G. Jeon, "Image autoregressive interpolation model using GPU-parallel optimization," *IEEE Trans. Ind. Informat.*, vol. 14, no. 2, pp. 426–436, Feb. 2018.

[39] Z.-H. Liu, X.-H. Li, L.-H. Wu, S.-W. Zhou, and K. Liu, "Gpu-accelerated parallel coevolutionary algorithm for parameters identification and temperature monitoring in permanent magnet synchronous machines," *IEEE Trans. Ind. Informat.*, vol. 11, no. 5, pp. 1220–1230, Oct. 2015.

[40] P. Li, Y. Luo, N. Zhang, and Y. Cao, "Heterospark: A heterogeneous CPU/GPU spark platform for machine learning algorithms," in *Proc. IEEE Int. Conf. Netw.*, Architecture Storage, 2015, pp. 347–348.

[41] C. Chen, K. Li, A. Ouyang, and K. Li, "FlinkCL: An openCL-based in-memory computing architecture on heterogeneous CPU-GPU clusters for big data," *IEEE Trans. Comput.*, vol. 67, no. 12, pp. 1765–1779, Dec. 2018.

[42] M. Zaharia *et al.*, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[43] E. Baccarelli, N. Cordeschi, A. Mei, M. Panella, M. Shojafar, and J. Stefa, "Energy-efficient dynamic traffic offloading and reconfiguration of networked data centers for big data stream mobile computing: Review, challenges, and a case study," *IEEE Netw.*, vol. 30, no. 2, pp. 54–61, Mar./Apr. 2016.

[44] E. Karydi and K. Margaritis, "Parallel and distributed collaborative filtering: A survey," *ACM Comput. Surv.*, vol. 49, no. 2, p. 37, 2016.

[45] W. Tan, L. Cao, and L. Fong, "Faster and cheaper: Parallelizing large-scale matrix factorization on GPUs," in *Proc. 25th ACM Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2016, pp. 219–230.

[46] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, "Large-scale parallel collaborative filtering for the netflix prize," in *Algorithmic Aspects in Information and Management*. Berlin, Germany: Springer, 2008, pp. 337–348.

[47] K. Shin, L. Sael, and U. Kang, "Fully scalable methods for distributed tensor factorization," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 1, pp. 100–113, Jan. 2017.

[48] H.-F. Yu, C.-J. Hsieh, S. Si, and I. Dhillon, "Scalable coordinate descent approaches to parallel matrix factorization for recommender systems," in *Proc. IEEE 12th Int. Conf. Data Mining*, 2012, pp. 765–774.

[49] I. Nisa, A. Sukumaran-Rajam, R. Kunchum, and P. Sadayappan, "Parallel CCD on GPU for matrix factorization," in *Proc. Gen. Purpose GPUs*, 2017, pp. 73–83.

[50] R. Kannan, G. Ballard, and H. Park, "MPI-FAUN: An MPI-based framework for alternating-updating nonnegative matrix factorization," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 3, pp. 544–558, Mar. 2018.

[51] L. Xu and M. Davenport, "Dynamic matrix recovery from incomplete observations under an exact low-rank constraint," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3585–3593.

[52] D. Lian, C. Zhao, X. Xie, G. Sun, E. Chen, and Y. Rui, "GeoMF: Joint geographical modeling and matrix factorization for point-of-interest recommendation," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2014, pp. 831–840.

[53] N. Guan, D. Tao, Z. Luo, and B. Yuan, "NeNMF: An optimal gradient method for nonnegative matrix factorization," *IEEE Trans. Signal Process.*, vol. 60, no. 6, pp. 2882–2898, Jun. 2012.

[54] C. Chen *et al.*, "Online inductor parameters identification by small-signal injection for sensorless predictive current controlled boost converter," *IEEE Trans. Ind. Informat.*, vol. 13, no. 4, pp. 1554–1564, Aug. 2017.

[55] K. Xie *et al.*, "On-line anomaly detection with high accuracy," *IEEE/ACM Trans. Netw.*, vol. 26, no. 3, pp. 1222–1235, Jun. 2018.

[56] S. Rendle and L. Schmidt-Thieme, "Online-updating regularized kernel matrix factorization models for large-scale recommender systems," in *Proc. ACM Conf. Recommender Syst.*, Lausanne, Switzerland, Oct. 2008, pp. 251–258.

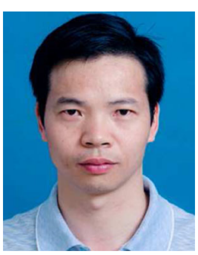
[57] S. Zhou, N. X. Vinh, J. Bailey, Y. Jia, and I. Davidson, "Accelerating online CP decompositions for higher order tensors," in *Proc. ACM SIGKDD Int. Conf.*, 2016, pp. 1375–1384.

[58] R. Zhao and V. Y. F. Tan, "Online nonnegative matrix factorization with outliers," *IEEE Trans. Signal Process.*, vol. 65, no. 3, pp. 555–570, Feb. 2017.

[59] X. Zhao *et al.*, "Scalable linear visual feature learning via online parallel nonnegative matrix factorization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 12, pp. 2628–2642, Dec. 2016.

Hao Li is currently working toward the Ph.D. degree with Hunan University, Changsha, China.

His research interests include large-scale sparse matrix and tensor factorization, recommender systems, social network, data mining, machine learning, and graphics processing unit (GPU) and multi-GPU computing. He has authored and coauthored six journal and conference papers in the IEEE Transactions on Parallel and Distributed Systems, *InforSci*, IEEE Transactions on Industrial Informatics, Conference on Information and Knowledge Management, and ISPA.



Kenli Li (Senior Member, IEEE) received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2003.

He was a Visiting Scholar with the University of Illinois at Urbana-Champaign from 2004 to 2005. He is currently a Full Professor of Computer Science and Technology with Hunan University, Changsha, China, and the Deputy Director with National Supercomputing Center, Changsha. His main research interests include parallel computing, high-performance computing, and grid and cloud computing. He has authored and coauthored more than 130 research papers in international conferences and journals such as the IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Signal Processing, *Journal of Parallel and Distributed Computing*, International Conference on Parallel Processing, and IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing.

Dr. Li is an Outstanding Member of the China Computer Federation. He serves on the editorial board of IEEE Transactions on Computers.

Jiyao An (Member, IEEE) received the Ph.D. degree in mechanical engineering from Hunan University, Changsha, China, in 2012.

He was a Visiting Scholar with the Department of Applied Mathematics, University of Waterloo, Waterloo, ON, Canada. He is currently a Full Professor with the College of Computer Science and Electronic Engineering, Hunan University. His research interests include cyber-physical systems, Takagi–Sugeno fuzzy systems, parallel and distributed computing, and computational intelligence. He has authored and coauthored more than 50 papers in international and domestic journals and refereed conference papers.

Dr. An is a Member of the Association for Computing Machinery, and a Senior Member of the China Computer Federation. He is an Active Reviewer for many international journals.

Keqin Li (Fellow, IEEE) is a Distinguished Professor of Computer Science with the State University of New York, New Paltz, NY, USA. He has authored and coauthored more than 480 journal articles, book chapters, and refereed conference papers. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU–GPU hybrid and cooperative computing, multicore computing, and storage and file systems.

Dr. Li was the recipient of several best paper awards. He is has been on the editorial boards for the IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Cloud Computing, IEEE Transactions on Services Computing, and IEEE Transactions on Sustainable Computing.