ICIC Express Letters ICIC International ©2016 ISSN 1881-803X
Volume **10**, Number **12**, December **2016** pp. **2905–2912**

PARALLEL MULTIPLE NONNEGATIVE MATRICES FACTORIZATION USING GRAPHICS PROCESSING UNIT

Xiaohui Huang[1], Xin Fu[2], Liyan Xiong[1], Yunming Ye[3]
Shaokai Wang[3] and Xiaolin Du[4]

[1]School of Information Engineering
East China Jiaotong University
No. 808, Shuanggang East Ave., Nanchang 330013, P. R. China
hxh016@gmail.com; xly_ecjtu@163.com

[2]Jiangxi College of Construction
No. 999, Huiren Ave., Nanchang 330200, P. R. China
76462714@qq.com

[3]Shenzhen Graduate School
Harbin Institute of Technology
HIT Campus at Xili University Town, Shenzhen 518055, P. R. China
yeyunming@hit.edu.cn; wangshaokai@gmail.com

[4]College of Computer Science
Beijing University of Technology
No. 100, Pingleyuan, Chaoyang Dist., Beijing 100124, P. R. China
du_xiaolin@bjut.edu.cn

Received April 2016; accepted July 2016

Abstract. *Multiple Nonnegative Matrices Factorization (MNMF) is a promising method to study and analyze a dataset which has different types of features or relationships. However, due to the high computational cost, MNMF cannot meet the needs of time response for large-scale datasets. In this paper, we introduce a Parallel Multiple Nonnegative Matrices Factorization (PMNMF) approach which is implemented on Graphics Processing Unit (GPU) under the Compute Unified Device Architecture (CUDA) framework. Experimental studies demonstrate that PMNMF approach using GPU is able to obtain* $100\times$ *speedup in comparison to the traditional multiple nonnegative matrices factorization under our experimental condition.*
Keywords: MNMF, CUDA, GPU, Parallelization

1. Introduction.

In nature, many real-world datasets include different types of features which are able to provide information complementary with each other. For example, an academic paper contains features: keywords, authors, citations and an object in Flicker (http://www.flickr.com/) includes tags and images. Multiple matrices are able to effectively represent this type of datasets. MNMF can be used to study and analyze this type of dataset, such as dimensions reduction [1], collaborative filter [2] and clustering analysis [3], to name just a few. However, due to the high computational cost, MNMF cannot meet the demand of the time response when it handles the high volumes of data. Therefore, we see a growing demand for accelerating MNMF to cater to different types of applications.

Since Lee and Seung proposed the algorithm of Nonnegative Matrix Factorization (NMF) [4] which is able to learn the parts of objects, different variants of NMF are proposed by changing similarity measures [5] or altering the iterative rules [6] or adding different constraints [7] to meet the requirements of different applications. In order to represent a dataset which has multiple types of features, a tensor [8] is used by many researchers in the process of discovering clusters or topics. However, the tensor is usually very sparse when it is used to represent a dataset in real-world application. In comparison to tensor, multiple matrices [3, 9, 10] have lower sparsity when we use them to represent a dataset having multiple types of features. Then, multiple nonnegative matrices factorization can be used as a tool for dimensions reduction [1], clustering analysis [3], community discover [10], etc.

The parallel matrix algorithms are also studied by many researchers [11, 12, 13, 14]. Parallel nonnegative matrix factorization is implemented in [11, 12], of which the most time-consuming step, matrix multiplication, is calculated with BLAS library (http://docs.nvidia.com/cuda/cublas/index.html) of nVIDIA CUDA. Antikainen et al. [15] proposed nonnegative tensor factorization implementation which targets analysis of high-dimensional spectral images, including dimensionality reduction, feature extraction, and other tasks related to spectral imaging. Inspired by these algorithms, we introduce a PMNMF algorithm for reducing the running time of MNMF.

On the basis of the works mentioned above, we introduce a parallel multiple nonnegative matrices factorization algorithm which is implemented on GPU under the CUDA framework. In our implementation, the operations can be transferred to the combination of some matrices' multiplications, matrices additions and matrices element-wise divisions. GPU has a significant speed advantage for those operations as opposed to Central Processing Unit (CPU). The experimental results on various datasets demonstrate that the speedup achieved by using a GPU is attractive, especially, with the increment of the number of objects, the dimensions of the objects and the value of the rank of the matrices after factorization.

The remaining sections of this paper are organized as follows. We introduce MNMF and give its parallel implementation in Section 2. Experiments on both synthetic and real datasets are presented in Section 3. Finally, we conclude this paper in Section 4.

2. The Model of Parallel Multiple Nonnegative Matrices Factorization.

2.1. The MNMF algorithm.

In this section, we briefly introduce multiple nonnegative matrices factorization. Let $X = \{X^1, X^2, \dots, X^p\}$ be a dataset which consists of p matrices. p is the number of types of features in a dataset. The number of attributes of the ith type of feature is I_i. $X^1, X^2, \dots, X^p$ are p feature matrices, the dimensions of which are $I_o \times I_1, \dots, I_o \times I_p$, respectively. All the values in X are nonnegative. Taking an academic paper as an example, there are three types of features: keywords, authors and citations. The value of p is three. Therefore, an academic paper dataset can be represented by paper-author matrix X^1, paper-keyword matrix X^2 and paper-citation matrix X^3, respectively.

The objective function of MNMF is written as follows:

$$P\left(O, U^1, \dots, U^p\right) = \frac{1}{2}\sum_{q=1}^{p} \left\| X^q - OU^{q^T} \right\|^2, \tag{1}$$

where O is the object facet and U^q is the qth feature facet. The above optimization problem can be solved iteratively by the following two steps:

$$u^q_{ij} = u^q_{ij} \frac{\left(X^{q^T} O\right)_{ij}}{\left(U^q O^T O\right)_{ij}}, \tag{2}$$

$$o_{ij} = o_{ij} \frac{\left(\sum\limits_{q=1}^{p} X^q U^q\right)_{ij}}{\left(\sum\limits_{q=1}^{p} OU^{q^T} U^q\right)_{ij}}. \tag{3}$$

The detailed proof can be referred to [4, 16].

2.2. **PMNMF on GPU.** The implementation of PMNMF also includes two parts, updating object facet and updating features facets, in which most of operations rely mainly on matrix multiplication, matrix addition and matrix element-wise division. For matrix multiplication, we can exploit the BLAS library which includes functions for Vector-Vector, Vector-Matrix and Matrix-Matrix operations. Moreover, we implement a matrix addition kernel function (1) and an element-wise matrix division kernel function (2) for PMNMF. The overall procedure of PMNMF can be described as Algorithm 1.

Algorithm 1 PMNMF

```
1: Input: $X^1$, $X^2$, ..., $X^p$, $k$.
2: Output: $O$, $U^1$, $U^2$, ..., $U^p$.
3: Initialize: Randomly choose an initial $O$, $U^1$, $U^2$, ..., $U^p$.
4: Initialize: Transfer the data to the GPU memory.
5: repeat
6:     calculate $O^TO$ with GPU;
7:     for $q = 1$ to $p$ do
8:         Fixed $O$, $U^1$, ..., $U^{q-1}$, $U^{q+1}$, ..., $U^p$, compute the following steps on GPU;
9:             (1) calculate ${X^q}^TO$ on GPU;
10:            (2) calculate $U^qO^TO$ on GPU;
11:            (3) calculate Equation (2) with kernel function in Figure 2 on GPU;
12:    end for
13:    Fixed $U^1$, ..., $U^p$, compute the following steps on GPU;
14:    Initialize temporary variables SumXU, SumUtU to zeros;
15:    for $q = 1$ to $p$ do
16:            (1) calculate $X^qU^q$ on GPU;
17:            (2) calculate ${U^q}^TU^q$ on GPU;
18:            (3) calculate SumUtU = SumUtU + ${U^q}^TU^q$ with kernel function in Figure 1 on GPU;
19:    end for
20:    calculate $O \times$ SumUtU on GPU;
21:    calculate Equation (3) with kernel function in Figure 2 on GPU;
22: until Convergence.
```

```
__global__ void matrixAdd(float* A, float* B, int element){
        int idx=blockIdx.x*blockDim.x+threadIdx.x;
        if(idx < element) A[idx]+=B[idx];
}
```

Figure 1. The kernel function of matrix addition

```
__global__ void MatrixElementWiseDivision
  (float* A, float* B,float* C, int element){
        int idx=blockIdx.x*blockDim.x+threadIdx.x;
        if(idx < element) C[idx]*=A[idx]/B[idx];
}
```

Figure 2. The kernel function of element-wise matrix division

2.3. **Computational complexity.** The MNMF algorithm mainly includes two computational steps: (1) updating feature facets; (2) updating object facet. The computational cost of both step (1) and step (2) is $\sum_{q=1}^{p} (I_q I_o k + I_o k^2 + I_q k^2)$. Therefore, the computational cost of MNMF is $\sum_{q=1}^{p} (I_q I_o k + I_o k^2 + I_q k^2)$ in overall.

In the implementation, we load the entire dataset to the memory. Thus, we need $\sum_{q=1}^{p} (I_o I_q)$ memory to save the original data. We need $\sum_{q=1}^{p} (I_q k) + I_o k$ memory to save the results. Moreover, another $k^2 + 2\max(I_q k + I_o k, 1 \leq q \leq p)$ memory space is required for saving temporary results. For CPU implementation, we need $\sum_{q=1}^{p} (I_o I_q) + \sum_{q=1}^{p} (I_q k) + I_o k + k^2 + 2\max(I_q k + I_o k, 1 \leq q \leq p)$ main memory space to run MNMF. Compared to the computational cost of MNMF, PMNMF must consider the extra time cost of data exchanges between main memory and GPU memory. Moreover, PMNMF also needs another $\sum_{q=1}^{p} (I_o I_q) + \sum_{q=1}^{p} (I_q k) + I_o k + k^2 + 2\max(I_q k + I_o k, 1 \leq q \leq p)$ GPU memory space to run PMNMF.

3. Experiments.

In our experiments, the algorithms run on a workstation machine with an NVIDIA Quadro FX580 GPU card. To evaluate the performance of PMNMF, we randomly generate several synthetic datasets and use two real datasets to compare the running speeds between MNMF and PMNMF. We implement both algorithms with C++.

3.1. **Synthetic dataset.** In this subsection, we test the performance of PMNMF from the increment of the following aspects: the number of iterations, the number of dimensions, the number of objects, the number of facets and the values of ranks. We randomly generate different sizes of datasets to compare the running time between MNMF and PMNMF. Tables 1-5 show the results produced by MNMF and PMNMF with different iterations, dimensions, number of objects, ranks and facets, respectively. From Tables 1 and 5, the running time produced by both MNMF and PMNMF increases linearly with the increment of iteration and the number of facets. From Figures 3(a) and 3(e), the speedup of the running time between MNMF and PMNMF keeps about 130 under our hardware configuration. That is because the number of threads that execute in parallel

Table 1. The comparative results of the computational time [Second] with different iterations

Iteration	10	50	100	200	500
MNMF (CPU)	369.31	1807.5	3595.3	7148.3	17836
PMNMF (GPU)	3.542	13.897	26.843	52.878	133.7

The dataset is comprised of two facets. Each facet is a $5,000 \times 1,000$ matrix. Rank = 50.

Table 2. The comparative results of computation time [Second] with different dimensions

Dimension	10	100	500	1000	5000	10,000
MNMF (CPU)	99.60	246.8	909.6	1807.5	10008	20349
PMNMF (GPU)	1.268	2.186	7.361	13.897	67.11	133.1

The dataset is comprised of two facets. The number of objects is $5,000$. We set the iterations and rank to 50 and 50, respectively.

TABLE 3. The comparative results of computation time [Second] with different number of objects

Object	100	500	1000	5000	10,000	50,000
MNMF (CPU)	52.44	196.0	357.3	1807.5	3904.8	19915
PMNMF (GPU)	1.138	2.034	3.284	13.897	24.313	128.2

The dataset is comprised of two facets. The number of dimensions is $1,000$. We set the iterations and rank to 50 and 50, respectively.

TABLE 4. The comparative results of computation time [Second] with different ranks

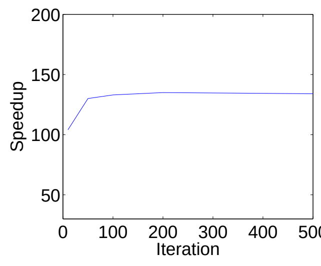
Rank	5	10	50	100	500
MNMF (CPU)	164.8	337.3	1807.6	3795.5	27894
PMNMF (GPU)	4.463	4.533	13.90	20.065	103.0

The dataset is comprised of two facets. Each facet is a $5,000 \times 1,000$ matrix. We set the iterations to 50.

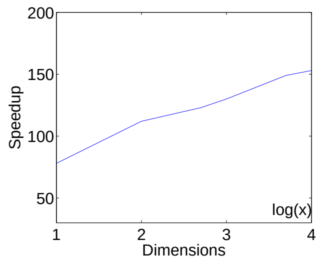
TABLE 5. The comparative results of computation time [Second] with different facets

Facet	2	5	10	20
MNMF (CPU)	1807.5	4455.9	9011.0	17543.6
PMNMF (GPU)	13.897	33.569	66.662	133.121

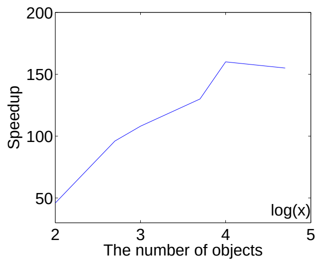
Each facet of this dataset is a $5,000 \times 1,000$ matrix. We set the iterations and rank to 50 and 50, respectively.



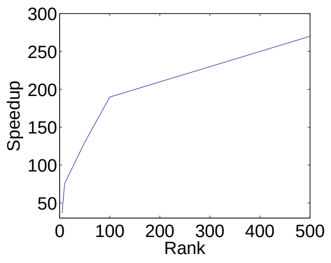
(a) Speedup with the increment of iterations



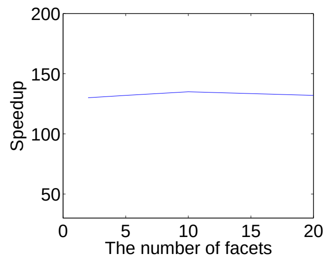
(b) Speedup with the increment of dimensions



(c) Speedup with the increment of the number of objects



(d) Speedup with the increment of the ranks



(e) Speedup with the increment of the number of facets

FIGURE 3. The speedup on synthetic datasets

keeps constant with increment of iteration and the number of facets. Therefore, the speedup is similar under the conditions of different iterations or the number of facets.

From Tables 2 and 3, the computational time produced by MNMF increases faster than that produced by PMNMF with the increment of the dimensions or the number of objects. Also, we can see from Figures 3(b) and 3(c) that the speedup increases with the increment of the dimensions or the number of objects. That is because the number of running threads in parallel will add gradually with the increment of the dimensions or the number of objects when updating the corresponding facet. From Table 4, we can observe that the running time of MNMF increases rapidly with the increment of the value of rank since the computational cost is directly proportional to the square of the value of rank. However, the running time of PMNMF is not as rapid as MNMF. That is because PMNMF is able to generate more running threads with increment of the values of rank when updating all the facets. Thus, the speedup increases rapidly with the increment of the values of the rank. In summary, the speedup between PMNMF and MNMF depends mainly on the values of rank, the number of objects and the dimensions. The bigger the calculating amount has, the more the number of running threads can be parallelized, and PMNMF can obtain a higher speedup in comparison to MNMF.

3.2. **Real-world dataset.** To further investigate the performance of PMNMF in a real-life dataset, we have evaluated PMNMF on another dataset NIPS which includes the full texts and the authors of the paper of the Proceedings from 2000 to 2012 Neural Information Processing Systems (NIPS) Conferences. NIPS includes 1752 documents and two facets. Two facets have 2257, 13359 features, respectively.

Since the real dataset has the fixed number of objects, facets and dimensions, we show the computational time and speedup produced by MNMF and PMNMF with increment of only the iteration and the values of the rank. Tables 6 and 7 show the computational time on NIPS with the increment of the iterations and the values of rank. From Table 6,

Table 6. The comparative results of computation time [Second] with different iterations on NIPS

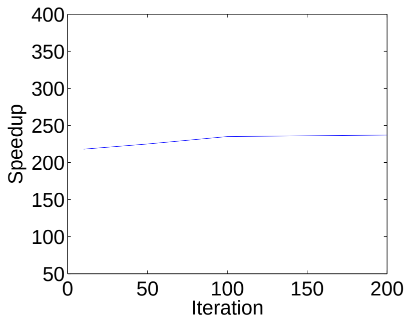
Iteration	10	50	100	200	500
MNMF (CPU)	1502.6	7357.5	15255	30656.1	–
PMNMF (GPU)	6.886	32.682	64.90	129.234	321.791

We set rank to 50. – represents that we do not obtain the computational time, because the computational time is too long.

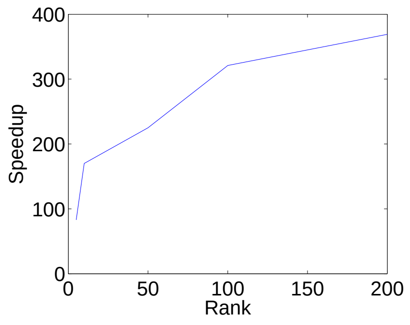
Table 7. The comparative results of computation time [Second] with different ranks on NIPS

Rank	5	10	50	100	200	500
MNMF (CPU)	732.96	1534.0	7357.5	14698	30898	–
PMNMF (GPU)	8.787	9.012	32.682	45.70	83.73	241.530

We set the iterations to 50. – represents that we donot obtain the computational time because the computational time is too long.



(a) Speedup with the increment of iterations



(b) Speedup with the increment of ranks

Figure 4. The speedup on NIPS

we can see that the proportion of time costs produced by PMNMF between two different iterations is approximately proportional to that produced by running MNMF. From Figure 4(a), we can also see that the speedups of different iterations are similar. However, from Table 7, we can see that the increment of the computational time produced by running MNMF is significantly faster than that produced by PMNMF. From Figure 4(b), we can see that the speedup increases from less than 100 to more than 350 with the increment of the values of the rank. This observation is similar to that produced by synthetic datasets.

4. **Conclusion and Future Work.** This paper introduces an efficient implementation of PMNMF algorithm on GPU platform with CUDA framework. From the results in Section 3, we can see that the speedups measured on synthetic datasets and a real dataset are around 100× compared to MNMF with a traditional C++ implementation. Our method is able to provide a solution to the problem of high computational complexity of MNMF. However, this implementation has to load the entire dataset to the memory of GPU in the first place, which may limit the scope of usage of our method. In the future work, we plan to develop a new version of PMNMF which can load a dataset to the GPU memory by batches for large datasets.

Acknowledgement. The authors are very grateful to the editor and anonymous referees for their helpful comments. This work was supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61562027, No. 61363072, the Natural Science Foundation of Jiangxi Province under Grant No. 20161BAB212050 and No. 20161BBI90032, Social Science Planning Project of Jiangxi Province under Grant No. 15XW12 and Education Department of Jiangxi Province under Grant No. GJJ150494.

REFERENCES

[1] D. W. Scott, The curse of dimensionality and dimension reduction, in *Multivariate Density Estimation: Theory, Practice, and Visualization*, John Wiley & Sons, Inc., 2015.
[2] R. Gemulla, E. Nijkamp, P. J. Haas and Y. Sismanis, Large-scale matrix factorization with distributed stochastic gradient descent, *Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.69-77, 2011.
[3] J. Liu, C. Wang, J. Gao and J. Han, Multi-view clustering via joint nonnegative matrix factorization, *Proc. of SIAM Data Mining Conf. (SDM'13)*, 2013.
[4] D. D. Lee and H. S. Seung, Learning the parts of objects by non-negative matrix factorization, *Nature*, vol.401, no.6755, pp.788-791, 1999.
[5] R. Zdunek, Trust-region algorithm for nonnegative matrix factorization with alpha- and beta-divergences, *Proc. of Joint the 34th DAGM and the 36th OAGM Symposium on Pattern Recognition (DAGM/OAGM)*, Graz, Austria, vol.7476, pp.226-235, 2012.
[6] Ş. M. Şoltuz and B. Rhoades, A mixed iteration for nonnegative matrix factorizations, *Applied Mathematics and Computation*, 2013.
[7] N. Guan, D. Tao, Z. Luo and B. Yuan, Nenmf: An optimal gradient method for nonnegative matrix factorization, *IEEE Trans. Signal Processing*, vol.60, no.6, pp.2882-2898, 2012.
[8] Y. Matsubara, Y. Sakurai, C. Faloutsos, T. Iwata and M. Yoshikawa, Fast mining and forecasting of complex time-stamped events, *Proc. of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp.271-279, 2012.
[9] S. Gupta, D. Phung, B. Adams and S. Venkatesh, A matrix factorization framework for jointly analyzing multiple nonnegative data, *Proc. of the 11th SIAM International Conference on Data Mining*, 2011.
[10] N. Du, X. Jia, J. Gao, V. Gopalakrishnan and A. Zhang, Tracking temporal community strength in dynamic networks, *IEEE Trans. Knowledge and Data Engineering*, vol.27, no.11, pp.3125-3137, 2015.
[11] J. Platoš, P. Gajdoš, P. Krömer and V. Snášel, Non-negative matrix factorization on GPU, *Networked Digital Technologies*, Springer, pp.21-30, 2010.
[12] N. Lopes and B. Ribeiro, Non-negative matrix factorization implementation using graphic processing units, *Intelligent Data Engineering and Automated Learning*, Springer, pp.275-283, 2010.

[13] W. Tan, L. Cao and L. Fong, Faster and cheaper: Parallelizing large-scale matrix factorization on GPUs, *Proc. of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, pp.219-230, 2016.

[14] R. Aihara, T. Takiguchi and Y. Ariki, Multiple non-negative matrix factorization for many-to-many voice conversion, *IEEE/ACM Trans. Audio, Speech, and Language Processing*, vol.24, no.7, pp.1175-1184, 2016.

[15] J. Antikainen, J. Havel, R. Josth, A. Herout, P. Zemcik and M. Hauta-Kasari, Nonnegative tensor factorization accelerated using GPGPU, *IEEE Trans. Parallel and Distributed Systems*, vol.22, no.7, pp.1135-1141, 2011.

[16] D. Seung and L. Lee, Algorithms for non-negative matrix factorization, *Advances in Neural Information Processing Systems*, vol.13, pp.556-562, 2001.