NESTEROV-BASED PARALLEL ALGORITHM FOR LARGE-SCALE NONNEGATIVE TENSOR FACTORIZATION

A. P. Liavas*, *G. Kostoulas**, *G. Lourakis**, *K. Huang*†, *N. D. Sidiropoulos*†

* Department of ECE Technical University of Crete

(liavas,kostoulas,lourakis)@telecom.tuc.gr

† Department of ECE University of Minnesota

(huang663,nikos)@ece.umn.edu

ABSTRACT

We consider the problem of nonnegative tensor factorization. Our aim is to derive an efficient algorithm that is also suitable for parallel implementation. We adopt the alternating optimization (AO) framework and solve each matrix nonnegative least-squares problem via a Nesterov-type algorithm for strongly convex problems. We describe a parallel implementation of the algorithm and measure the speedup attained by its Message Passing Interface implementation on a parallel computing environment. It turns out that the attained speedup is significant, rendering our algorithm a competitive candidate for the solution of very large-scale dense nonnegative tensor factorization problems.

Index Terms— Tensors, constrained optimization, CANDECOMP, PARAFAC, nonnegative factorization, parallel algorithms.

1. INTRODUCTION

Tensor factorizations (or decompositions) into latent factors are very important for numerous tasks, such as feature selection, dimensionality reduction, compression, data visualization and interpretation, and are usually computed as solutions of optimization problems [1], [2]. The Canonical Decomposition or Canonical Polyadic Decomposition (CANDECOMP or CPD), also known as Parallel Factor Analysis (PARAFAC), and the Tucker Decomposition are the two most widely used tensor factorization models.

Typical data sizes have been growing rapidly in recent years, and this is especially true for machine learning applications of tensor factorization models. This trend towards bigger and bigger data has made imperative the development of parallel algorithms for tensor factorization. In [3], two parallel algorithms for unconstrained tensor factorization/completion have been developed and results concerning the speedup attained by their Message Passing Interface (MPI) implementations have been reported. Work on parallel algorithms for sparse tensor decomposition appears in [4], [5].

In this work, we focus on (dense) nonnegative PARAFAC, which, for simplicity, we call Nonnegative Tensor Factorization (NTF).[1] Our aim is to derive an efficient NTF algorithm that is also suitable for parallel implementation. We adopt the Alternating Optimization (AO) framework and solve each matrix nonnegative least-squares (MNLS) subproblem via a Nesterov-type (accelerated gradient) algorithm for L-smooth and μ-strongly convex problems [7, Chapter 2]. This framework seems quite appropriate because it optimally exploits first-order, i.e., gradient, information and leads to an algorithm that is suitable for parallel implementation. We describe a parallel implementation of the algorithm and measure the speedup attained by its Message Passing Interface (MPI) implementation in a multi-core environment. We conclude that the proposed algorithm is a strong candidate for the solution of very large dense NTF problems.

Notation: Vectors, matrices, and tensors are denoted by small, capital, and calligraphic capital bold letters, respectively; for example, $\mathbf{x}$, $\mathbf{X}$, and $\boldsymbol{\mathcal{X}}$. $\mathbb{R}_+^{I\times J\times K}$ denotes the set of $(I\times J\times K)$ real nonnegative tensors, while $\mathbb{R}_+^{I\times J}$ denotes the set of $(I\times J)$ real nonnegative matrices. $\|\cdot\|_F$ denotes the matrix or tensor Frobenius norm, $\mathbf{I}$ denotes the identity matrix of appropriate dimensions, and $(\mathbf{A})_+$ denotes the projection of matrix $\mathbf{A}$ onto the set of elementwise nonnegative matrices. The outer product of vectors $\mathbf{a}\in\mathbb{R}^{I\times 1}$, $\mathbf{b}\in\mathbb{R}^{J\times 1}$, and $\mathbf{c}\in\mathbb{R}^{K\times 1}$ is the rank-one tensor $\mathbf{a}\circ\mathbf{b}\circ\mathbf{c}\in\mathbb{R}^{I\times J\times K}$ with elements $(\mathbf{a}\circ\mathbf{b}\circ\mathbf{c})(i,j,k)=\mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)$. The Khatri-Rao product of compatible matrices $\mathbf{A}$ and $\mathbf{B}$ is denoted as $\mathbf{A}\odot\mathbf{B}$ and the Hadamard product is denoted as $\mathbf{A}\circledast\mathbf{B}$. Inequality $\mathbf{A}\succeq\mathbf{B}$ means that matrix $\mathbf{A}-\mathbf{B}$ is positive semidefinite.

2. NONNEGATIVE TENSOR FACTORIZATION

Let tensor $\boldsymbol{\mathcal{X}}^o\in\mathbb{R}_+^{I\times J\times K}$ admit a factorization of the form

$$\boldsymbol{\mathcal{X}}^o=[\![\mathbf{A}^o,\mathbf{B}^o,\mathbf{C}^o]\!]=\sum_{r=1}^{R}\mathbf{a}_r^o\circ\mathbf{b}_r^o\circ\mathbf{c}_r^o, \tag{1}$$

where $\mathbf{A}^o=[\mathbf{a}_1^o\ \cdots\ \mathbf{a}_R^o]\in\mathbb{R}_+^{I\times R}$, $\mathbf{B}^o=[\mathbf{b}_1^o\ \cdots\ \mathbf{b}_R^o]\in\mathbb{R}_+^{J\times R}$, and $\mathbf{C}^o=[\mathbf{c}_1^o\ \cdots\ \mathbf{c}_R^o]\in\mathbb{R}_+^{K\times R}$. We observe the noisy tensor $\boldsymbol{\mathcal{X}}=\boldsymbol{\mathcal{X}}^o+\boldsymbol{\mathcal{E}}$, where $\boldsymbol{\mathcal{E}}$ is the additive noise. Estimates of $\mathbf{A}^o$, $\mathbf{B}^o$, and $\mathbf{C}^o$ can be obtained by computing matrices $\mathbf{A}\in\mathbb{R}_+^{I\times R}$, $\mathbf{B}\in\mathbb{R}_+^{J\times R}$, and $\mathbf{C}\in\mathbb{R}_+^{K\times R}$ that solve the optimization problem

$$\min_{\mathbf{A}\geq\mathbf{0},\mathbf{B}\geq\mathbf{0},\mathbf{C}\geq\mathbf{0}} f_{\boldsymbol{\mathcal{X}}}(\mathbf{A},\mathbf{B},\mathbf{C}):=\frac{1}{2}\left\|\boldsymbol{\mathcal{X}}-[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]\right\|_F^2. \tag{2}$$

If $\boldsymbol{\mathcal{Y}}=[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]$, then its matrix unfoldings, with respect to the first, second, and third mode, are given by [8]

$$\mathbf{Y}_{\mathbf{A}}=\mathbf{A}\,(\mathbf{C}\odot\mathbf{B})^T,\ \mathbf{Y}_{\mathbf{B}}=\mathbf{B}\,(\mathbf{C}\odot\mathbf{A})^T,\ \mathbf{Y}_{\mathbf{C}}=\mathbf{C}\,(\mathbf{B}\odot\mathbf{A})^T.$$

Thus, $f_{\boldsymbol{\mathcal{X}}}$ can be expressed as

$$\begin{aligned} f_{\boldsymbol{\mathcal{X}}}(\mathbf{A},\mathbf{B},\mathbf{C}) &= \frac{1}{2}\left\|\mathbf{X}_{\mathbf{A}}-\mathbf{A}\,(\mathbf{C}\odot\mathbf{B})^T\right\|_F^2 \\ &= \frac{1}{2}\left\|\mathbf{X}_{\mathbf{B}}-\mathbf{B}\,(\mathbf{C}\odot\mathbf{A})^T\right\|_F^2 \\ &= \frac{1}{2}\left\|\mathbf{X}_{\mathbf{C}}-\mathbf{C}\,(\mathbf{B}\odot\mathbf{A})^T\right\|_F^2. \end{aligned} \tag{3}$$

† Supported in part by NSF IIS-1247632 and IIS-1447788.

[1] A more detailed presentation of the algorithm, its performance, and its parallel implementation appears in [6].

These expressions form the basis for the AO NTF in the sense that, if we fix two matrix factors, we can update the third by solving an MNLS problem. For reasons related with the conditioning of the MNLS problem, we propose to add a proximal term. More specifically, if $\mathbf{A}_k$, $\mathbf{B}_k$, and $\mathbf{C}_k$ are the estimates of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, respectively, after the k-th AO iteration, then $\mathbf{A}_{k+1}$ is given by

$$\mathbf{A}_{k+1} := \underset{\mathbf{A}\geq\mathbf{0}}{\operatorname{argmin}} \frac{1}{2}\left\|\mathbf{X}_{\mathbf{A}} - \mathbf{A}(\mathbf{C}_k \odot \mathbf{B}_k)^T\right\|_F^2 + \frac{\lambda_k^{\mathbf{A}}}{2}\left\|\mathbf{A}-\mathbf{A}_k\right\|_F^2, \tag{4}$$

where $\lambda_k^{\mathbf{A}} \geq 0$ determines the weight assigned to the proximal term. If $(\mathbf{C}_k \odot \mathbf{B}_k)$ is a well-conditioned matrix, then it is reasonable to put small weight on the proximal term and compute $\mathbf{A}_{k+1}$ that leads to a large decrease of $f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}, \mathbf{B}_k, \mathbf{C}_k)$. If, on the other hand, $(\mathbf{C}_k \odot \mathbf{B}_k)$ is an ill-conditioned matrix, then it is reasonable to put large weight on the proximal term, leading to a better conditioned problem and easy computation of $\mathbf{A}_{k+1}$ that improves the fit in $f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}, \mathbf{B}_k, \mathbf{C}_k)$ but is not very far from $\mathbf{A}_k$. This is the strategy we shall follow for the solution of problem (2) (see also [9], [10]).

The computational efficiency of the AO NTF heavily depends on the algorithm we use for the solution of problem (4). In this work, we adopt the approach of Nesterov for the solution of smooth and strongly convex problems. The derived algorithm is optimal under the (worst-case) black-box first-order oracle framework [7, Chapter 2] and is very efficient in practice. Furthermore, it leads to an AO NTF algorithm that is suitable for parallel implementation.

3. NESTEROV-TYPE ALGORITHM FOR AO NTF

3.1. Optimal first-order methods for set-constrained L-smooth μ-strongly convex optimization problems

Let $0 < \mu \leq L < \infty$ and $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth convex function, with Hessian $\nabla^2 f(\mathbf{x})$, such that

$$\mu\mathbf{I} \preceq \nabla^2 f(\mathbf{x}) \preceq L\mathbf{I}, \ \ \forall\, \mathbf{x} \in \mathbb{R}^n. \tag{5}$$

Then, we say that f is an L-smooth μ-strongly convex function, denoted as $f \in \mathcal{S}_{\mu,L}^{1,1}$ [7, p. 63]. Let $\mathbb{Q}$ be a closed convex set. Our aim is to solve problem

$$\min_{\mathbf{x}\in\mathbb{Q}} f(\mathbf{x}), \tag{6}$$

within accuracy $\epsilon > 0$, using only first-order, i.e., gradient, information. The accuracy of the solution is defined as follows. Let $f_* := \min_{\mathbf{x}\in\mathbb{Q}} f(\mathbf{x})$. A point $\bar{\mathbf{x}} \in \mathbb{Q}$ solves problem (6) within accuracy $\epsilon > 0$ if $f(\bar{\mathbf{x}}) - f_* \leq \epsilon$.

We consider optimal first-order methods because they are strong and, in many cases, the only viable candidates for the efficient solution of very large optimization problems. It has been shown in [7, Chapter 2] that the first-order oracle complexity of this class of problems is $O\left(\sqrt{\frac{L}{\mu}}\log\frac{1}{\epsilon}\right)$. An algorithm that achieves this complexity and, thus, is first-order optimal is given in Algorithm 1 (see [7, pp. 80, 81, 90]). If the projection operation onto set $\mathbb{Q}$, $\Pi_{\mathbb{Q}}(\cdot)$, is easy to compute, then this algorithm is very efficient in practice. Constants μ and L are unknown and, thus, should be estimated.

In the sequel, we adopt Algorithm 1 for the solution of MNLS problems with proximal term. We note that [11] and [12] solved MNLS problems using a variation of Algorithm 1 which is equivalent to Algorithm 1 with $\mu = 0$. This algorithm is first-order optimal if the problem is L-smooth; in this case, the oracle complexity becomes $O\left(\frac{1}{\sqrt{\epsilon}}\right)$ (see [7, p. 90]). However, if the problem is L-smooth *and* μ-strongly convex, then this algorithm is *not* first-order optimal and usually performs much worse than the optimal. Thus, strong convexity is a very important property that should not be neglected.

Algorithm 1: Nesterov algorithm for set-constrained L-smooth and μ-strongly convex optimization problems

```
   Input: $\mathbf{x}_0 \in \mathbb{Q}$, $\mu$, $L$. Set $\mathbf{y}_0 = \mathbf{x}_0$, $\alpha_0 \in (0,1)$, $q = \frac{\mu}{L}$.
1  $k$-th iteration
2      $\mathbf{x}_{k+1} = \Pi_{\mathbb{Q}}\left(\mathbf{y}_k - \frac{1}{L}\nabla f(\mathbf{y}_k)\right)$
3      $\alpha_{k+1} \in (0,1)$ from $\alpha_{k+1}^2 = (1-\alpha_{k+1})\alpha_k^2 + q\alpha_{k+1}$
4      $\beta_{k+1} = \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2+\alpha_{k+1}}$
5      $\mathbf{y}_{k+1} = \mathbf{x}_{k+1} + \beta_{k+1}(\mathbf{x}_{k+1} - \mathbf{x}_k)$
```

Algorithm 2: Nesterov algorithm for MNLS with proximal term

```
   Input: $\mathbf{X} \in \mathbb{R}^{m\times k}$, $\mathbf{B} \in \mathbb{R}^{k\times n}$, $\mathbf{A}_* \in \mathbb{R}^{m\times n}$
1  $L' = \max(\mathrm{eig}(\mathbf{B}^T\mathbf{B}))$, $\mu' = \min(\mathrm{eig}(\mathbf{B}^T\mathbf{B}))$
2  $\lambda = g(\mu', L')$, $\mathbf{W} = -\mathbf{X}\mathbf{B} - \lambda\mathbf{A}_*$, $\mathbf{Z} = \mathbf{B}^T\mathbf{B} + \lambda\mathbf{I}$
3  $q = \frac{\mu'+\lambda}{L'+\lambda}$; $L = L' + \lambda$; $\mathbf{A}_0 = \mathbf{Y}_0 = \mathbf{A}_*$, $\alpha_0 = 1$, $k = 0$
4  while (1) do
5      $\nabla f(\mathbf{Y}_k) = \mathbf{W} + \mathbf{A}_k\mathbf{Z}$
6      if (terminating_condition is TRUE) then
7          break
8      else
9          $\mathbf{A}_{k+1} = \left(\mathbf{Y}_k - \frac{1}{L}\nabla f(\mathbf{Y}_k)\right)_+$
10         $\alpha_{k+1}^2 = (1-\alpha_{k+1})\alpha_k^2 + q\alpha_{k+1}$
11         $\beta_{k+1} = \frac{\alpha_k(1-\alpha_k)}{\alpha_k^2+\alpha_{k+1}}$
12         $\mathbf{Y}_{k+1} = \mathbf{A}_{k+1} + \beta_{k+1}(\mathbf{A}_{k+1} - \mathbf{A}_k)$
13         $k = k+1$
14 return $\mathbf{A}_k$.
```

3.2. Nesterov-type algorithm for MNLS with proximal term

Let $\mathbf{X} \in \mathbb{R}^{m\times k}$, $\mathbf{A}, \mathbf{A}_* \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{k\times n}$, and consider the problem

$$\min_{\mathbf{A}\geq\mathbf{0}} f(\mathbf{A}) := \frac{1}{2}\left\|\mathbf{X} - \mathbf{A}\mathbf{B}^T\right\|_F^2 + \frac{\lambda}{2}\|\mathbf{A} - \mathbf{A}_*\|_F^2. \tag{7}$$

We shall solve this problem using only gradient information. The gradient of f, at point $\mathbf{A}$, is

$$\nabla f(\mathbf{A}) = -\left(\mathbf{X} - \mathbf{A}\mathbf{B}^T\right)\mathbf{B} + \lambda(\mathbf{A} - \mathbf{A}_*). \tag{8}$$

As we mentioned in Section 2, the main reason we introduce the proximal term into the cost function is the improvement of the conditioning of the MNLS problem and guarantee of strong convexity. If we consider problem (7) under the framework of the AO NTF algorithm, then we may compute the proximal coefficient λ based on $L' := \max\left(\mathrm{eig}(\mathbf{B}^T\mathbf{B})\right)$ and $\mu' := \min\left(\mathrm{eig}(\mathbf{B}^T\mathbf{B})\right)$. For example, if $\frac{\mu'}{L'} \ll 1$, then we may set $\lambda \gtrapprox 10\mu'$, significantly improving the conditioning of the problem, by putting large weight on the proximal term; however, in this case, we expect that the optimal solution will be biased towards $\mathbf{A}_*$. Otherwise, we may set $\lambda \lessapprox \mu'$, putting small weight on the proximal term and permitting significant progress towards the computation of $\mathbf{A}$ that satisfies approximate

Algorithm 3: Nesterov-based AO NTF

```
Input: $\boldsymbol{\mathcal{X}}$, $\mathbf{A}_0 > \mathbf{0}$, $\mathbf{B}_0 > \mathbf{0}$, $\mathbf{C}_0 > \mathbf{0}$.
1  Set $k = 0$
2  while (terminating condition is FALSE) do
3      $\mathbf{W}_\mathbf{A} = -\mathbf{X}_\mathbf{A}(\mathbf{C}_k \odot \mathbf{B}_k) - \lambda_k^\mathbf{A}\mathbf{A}_k$, $\mathbf{Z}_\mathbf{A} = \mathbf{C}_k^T\mathbf{C}_k \circledast \mathbf{B}_k^T\mathbf{B}_k + \lambda_k^\mathbf{A}\mathbf{I}$
4      $\mathbf{A}_{k+1} = \text{Nesterov\_MNLS}(\mathbf{W}_\mathbf{A}, \mathbf{Z}_\mathbf{A}, \mathbf{A}_k)$
5      $\mathbf{W}_\mathbf{B} = -\mathbf{X}_\mathbf{B}(\mathbf{C}_k \odot \mathbf{A}_{k+1}) - \lambda_k^\mathbf{B}\mathbf{B}_k$, $\mathbf{Z}_\mathbf{B} = \mathbf{C}_k^T\mathbf{C}_k \circledast \mathbf{A}_{k+1}^T\mathbf{A}_{k+1} + \lambda_k^\mathbf{B}\mathbf{I}$
6      $\mathbf{B}_{k+1} = \text{Nesterov\_MNLS}(\mathbf{W}_\mathbf{B}, \mathbf{Z}_\mathbf{B}, \mathbf{B}_k)$
7      $\mathbf{W}_\mathbf{C} = -\mathbf{X}_\mathbf{C}(\mathbf{A}_{k+1} \odot \mathbf{B}_{k+1}) - \lambda_k^\mathbf{C}\mathbf{C}_k$, $\mathbf{Z}_\mathbf{C} = \mathbf{A}_{k+1}^T\mathbf{A}_{k+1} \circledast \mathbf{B}_{k+1}^T\mathbf{B}_{k+1} + \lambda_k^\mathbf{C}\mathbf{I}$
8      $\mathbf{C}_{k+1} = \text{Nesterov\_MNLS}(\mathbf{W}_\mathbf{C}, \mathbf{Z}_\mathbf{C}, \mathbf{C}_k)$
9      $k = k + 1$
10 return $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{C}_k$.
```

equality $\mathbf{X} \approx \mathbf{A}\mathbf{B}^T$ as accurately as possible. We denote this functional dependence as $\lambda = g(\mu', L')$.

Problem (7) is L-smooth and μ-strongly convex, with $L = L' + \lambda$ and $\mu = \mu' + \lambda$. We note that the values of L and μ are *necessary* for the development of the Nesterov-type algorithm, thus, computation of L' and μ' is imperative.

A Nesterov-type algorithm for the solution of the MNLS problem (7) is given in Algorithm 2, where $(\cdot)_+$ denotes projection onto the set of matrices with nonnegative elements. For notational convenience, we denote lines 3 to 13 of Algorithm 2 as

$$\mathbf{A}_{\text{opt}} = \text{Nesterov\_MNLS}(\mathbf{W}, \mathbf{Z}, \mathbf{A}_*).$$

The Karush-Kuhn-Tucker (KKT) conditions for problem (7) are [11]

$$\nabla f(\mathbf{A}) \geq \mathbf{0},\ \mathbf{A} \geq \mathbf{0},\ \nabla f(\mathbf{A}) \circledast \mathbf{A} = \mathbf{0}. \tag{9}$$

These expressions can be used in a terminating condition. For example, we may terminate the algorithm if

$$\min_{i,j}\left([\nabla f(\mathbf{Y}_k)]_{i,j}\right) > -\text{tol}_1, \max_{i,j}\left(\left|[\nabla f(\mathbf{Y}_k) \circledast \mathbf{Y}_k]_{i,j}\right|\right) < \text{tol}_2,$$

for small positive real numbers tol_1 and tol_2.

Computational complexity: Quantities $\mathbf{W}$ and $\mathbf{Z}$ are computed once per algorithm call and cost, respectively, $O(mkn)$ and $O(kn^2)$ arithmetic operations. Quantities L and μ are also computed once and cost at most $O(n^3)$ operations. $\nabla f(\mathbf{Y}_k)$, $\mathbf{A}_k$, and $\mathbf{Y}_k$ are updated in every iteration with cost $O(mn^2)$, $O(mn)$, and $O(mn)$ arithmetic operations, respectively.

3.3. Nesterov-based AO NTF

In Algorithm 3, we present the Nesterov-based AO NTF. We start from a random point $(\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0)$ and solve, in a circular manner, MNLS problems with proximal terms, based on the previous estimates. We can use various termination criteria, like, for example, the (relative) change of the cost function.

It has been shown in [10] that the AO NTF algorithm with proximal term falls under the block successive upper bound minimization (BSUM) framework, which ensures convergence to a stationary point of problem (2).

4. PARALLEL IMPLEMENTATION OF AO NTF

In this section, we assume that we have at our disposal $p = p_\mathbf{A} \times p_\mathbf{B} \times p_\mathbf{C}$ processing elements and present a parallel implementation of the Nesterov-based AO NTF algorithm following the lines of the medium-grained approach of [4]. The p processors form a three-dimensional cartesian grid and are denoted as $p_{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$, $i_\mathbf{B} = 1, \ldots, p_\mathbf{B}$, and $i_\mathbf{C} = 1, \ldots, p_\mathbf{C}$.

4.1. Variable partitionings and data allocation

We partition factor $\mathbf{A}_k$ into $p_\mathbf{A}$ block rows as

$$\mathbf{A}_k = \left[\begin{array}{ccc} (\mathbf{A}_k^1)^T & \cdots & (\mathbf{A}_k^{p_\mathbf{A}})^T \end{array}\right]^T, \tag{11}$$

with $\mathbf{A}_k^{i_\mathbf{A}} \in \mathbb{R}^{\tilde{I} \times R}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$, where $\tilde{I} := \frac{I}{p_\mathbf{A}}$. We partition accordingly the matricization $\mathbf{X}_\mathbf{A}$ and get

$$\mathbf{X}_\mathbf{A} = \left[\begin{array}{ccc} (\mathbf{X}_\mathbf{A}^1)^T & \cdots & (\mathbf{X}_\mathbf{A}^{p_\mathbf{A}})^T \end{array}\right]^T, \tag{12}$$

with $\mathbf{X}_\mathbf{A}^{i_\mathbf{A}} \in \mathbb{R}^{\tilde{I} \times JK}$. In an analogous manner, we partition $\mathbf{B}_k$ and $\mathbf{X}_\mathbf{B}$ into $p_\mathbf{B}$ block rows, each of size $\tilde{J} \times R$ and $\tilde{J} \times IK$, respectively, and $\mathbf{C}_k$ and $\mathbf{X}_\mathbf{C}$ into $p_\mathbf{C}$ block rows, each of size $\tilde{K} \times R$ and $\tilde{K} \times IJ$, respectively, where $\tilde{J} := \frac{J}{p_\mathbf{B}}$ and $\tilde{K} := \frac{K}{p_\mathbf{C}}$.

As we show in Fig. 1, we partition tensor $\boldsymbol{\mathcal{X}}$ into p subtensors, according to the partitionings of the factor matrices, and allocate its parts to the various processors, so that processor $p_{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$ receives subtensor $\boldsymbol{\mathcal{X}}^{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$, which is defined in equation (10) at the top of the next page. Further, we assume that processor $p_{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$ knows $\mathbf{A}_k^{i_\mathbf{A}}$, $\mathbf{B}_k^{i_\mathbf{B}}$, and $\mathbf{C}_k^{i_\mathbf{C}}$. Finally, we assume that *all* processors know $\mathbf{A}_k^T\mathbf{A}_k$, $\mathbf{B}_k^T\mathbf{B}_k$, and $\mathbf{C}_k^T\mathbf{C}_k$.

4.2. Communication groups

We define various communication groups over subsets of the p processors. At first, we define $p_\mathbf{A}$ two-dimensional groups, each involving the $p_\mathbf{B} \times p_\mathbf{C}$ processors $p_{i_\mathbf{A},:,:}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$, which are used for the collaborative update of $\mathbf{A}_k^{i_\mathbf{A}}$. In the same manner, we define the groups $p_{:,i_\mathbf{B},:}$, for $i_\mathbf{B} = 1, \ldots, p_\mathbf{B}$, and $p_{:,:,i_\mathbf{C}}$, for $i_\mathbf{C} = 1, \ldots, p_\mathbf{C}$, which are used for the updates of $\mathbf{B}_k^{i_\mathbf{B}}$ and $\mathbf{C}_k^{i_\mathbf{C}}$, respectively.

Finally, we define $p_\mathbf{B} \times p_\mathbf{C}$ single-dimensional communication groups, each involving the $p_\mathbf{A}$ processors $p_{:,i_\mathbf{B},i_\mathbf{C}}$. In a similar manner, we define groups $p_{i_\mathbf{A},i_\mathbf{B},:}$ and $p_{i_\mathbf{A},:,i_\mathbf{C}}$.

4.3. Algorithm implementation

In the sequel, we describe in detail the parallel update of $\mathbf{A}_k$, that is, lines 3 and 4 of Algorithm 3 (the updates of $\mathbf{B}_k$ and $\mathbf{C}_k$ can be implemented in an analogous manner). The update of $\mathbf{A}_k$ is achieved

$$\boldsymbol{\mathcal{X}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}} := \boldsymbol{\mathcal{X}}\left((i_{\mathbf{A}}-1)\tilde{I}+1 : i_{\mathbf{A}}\tilde{I}, (i_{\mathbf{B}}-1)\tilde{J}+1 : i_{\mathbf{B}}\tilde{J}, (i_{\mathbf{C}}-1)\tilde{K}+1 : i_{\mathbf{C}}\tilde{K}\right). \tag{10}$$

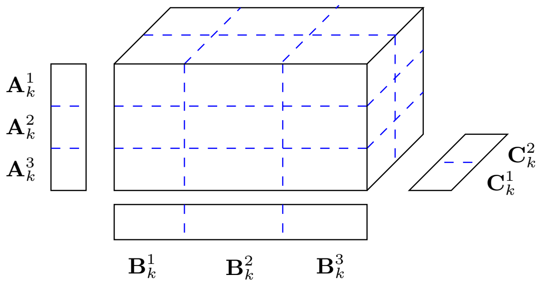

Fig. 1. Tensor $\boldsymbol{\mathcal{X}}$, factors $\mathbf{A}_k$, $\mathbf{B}_k$, and $\mathbf{C}_k$, and their partitioning for $p_{\mathbf{A}} = p_{\mathbf{B}} = 3$ and $p_{\mathbf{C}} = 2$.

via the updates of $\mathbf{A}_k^{i_{\mathbf{A}}}$, for $i_{\mathbf{A}} = 1, \ldots, p_{\mathbf{A}}$, and consists of the following stages:

1. For $i_{\mathbf{A}} = 1, \ldots, p_{\mathbf{A}}$, processors $p_{i_{\mathbf{A}},:,:}$ collaboratively compute $\mathbf{W}_{\mathbf{A}}^{i_{\mathbf{A}}} = \mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}}}(\mathbf{C}_k \odot \mathbf{B}_k)$. This can be done because

$$\mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}}}(\mathbf{C}_k \odot \mathbf{B}_k) = \sum_{i_{\mathbf{B}}=1}^{p_{\mathbf{B}}} \sum_{i_{\mathbf{C}}=1}^{p_{\mathbf{C}}} \mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}(\mathbf{C}_k^{i_{\mathbf{C}}} \odot \mathbf{B}_k^{i_{\mathbf{B}}}),$$

   where $\mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$ is the matricization of $\boldsymbol{\mathcal{X}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$, with respect to the first mode. Processor $p_{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$ computes the corresponding term of the above sum and the result is computed by an all-reduce operation over the group of processors $p_{i_{\mathbf{A}},:,:}$.

2. The computed $\mathbf{W}_{\mathbf{A}}^{i_{\mathbf{A}}}$ is scattered among the $p_{\mathbf{B}} \times p_{\mathbf{C}}$ processors $p_{i_{\mathbf{A}},:,:}$ (the scattering is based on the id of each processor in the group $p_{i_{\mathbf{A}},:,:}$). Each processor in the group $p_{i_{\mathbf{A}},:,:}$ uses the scattered part of $\mathbf{W}_{\mathbf{A}}^{i_{\mathbf{A}}}$, $\mathbf{Z}_{\mathbf{A}} = \mathbf{C}_k^T\mathbf{C}_k \circledast \mathbf{B}_k^T\mathbf{B}_k$, and $\lambda_k^{\mathbf{A}}$ and computes the updated part of $\mathbf{A}_{k+1}^{i_{\mathbf{A}}}$, via the Nesterov MNLS algorithm.

3. The updated parts of $\mathbf{A}_{k+1}^{i_{\mathbf{A}}}$ are all-gathered at the processors of the group $p_{i_{\mathbf{A}},:,:}$, so that all processors in the group know the updated $\mathbf{A}_{k+1}^{i_{\mathbf{A}}}$.

4. Using an all-reduce operation of $\left(\mathbf{A}_{k+1}^{i_{\mathbf{A}}}\right)^T \mathbf{A}_{k+1}^{i_{\mathbf{A}}}$ over processors $p_{:,i_{\mathbf{B}},i_{\mathbf{C}}}$, for $i_{\mathbf{B}} = 1, \ldots, p_{\mathbf{B}}$ and $i_{\mathbf{C}} = 1, \ldots, p_{\mathbf{C}}$, all processors compute the updated $\mathbf{A}_{k+1}^T\mathbf{A}_{k+1}$.

At this point, all processors are ready for the update of $\mathbf{B}_k$ and, then, of $\mathbf{C}_k$, finishing one outer iteration of the AO NTF algorithm.

4.4. Speedup

We present results obtained from an MPI implementation of the Nesterov-based AO NTF algorithm. The program is executed on a CentOS system with four AMD Opteron(tm) Processor 6376 (in total, 64 cores at 2.3 GHz) and 500 GB RAM. The algorithm is implemented in C++, using the Eigen library for linear algebra operations [13] and compiled using the g++ compiler (version 6.1.0) with the -O3 option for compiler optimizations. We assume a noiseless tensor $\boldsymbol{\mathcal{X}}$, whose true latent factors have i.i.d elements, uniformly distributed in $[0, 1]$. We partition $\boldsymbol{\mathcal{X}}$ and appropriately allocate its parts to the p processing elements. The timer that counts the program execution duration starts counting *after* the data allocation to the processing elements. We use $\text{tol}_1 = \text{tol}_2 = 10^{-2}$ in the terminating conditions for MNLS. The AO terminates at iteration k if (recall that tensor $\boldsymbol{\mathcal{X}}$ is noiseless)

$$\frac{\|\boldsymbol{\mathcal{X}} - [\![\mathbf{A}_k, \mathbf{B}_k, \mathbf{C}_k]\!]\|_F}{\|\boldsymbol{\mathcal{X}}\|_F} < 10^{-3}. \tag{13}$$

In Fig. 2, we plot the speedup achieved with p processing elements, for $p = 2^3, 3^3, 4^3$, for a tensor with $I = J = K = 1008$ and ranks $R = 15, 30, 45, 60$. We observe that we attain significant speedup, irrespective of the factorization rank, indicating that our algorithm is a strong candidate for the solution of large dense NTF problems. For a more detailed exposition, we refer the reader to [6].

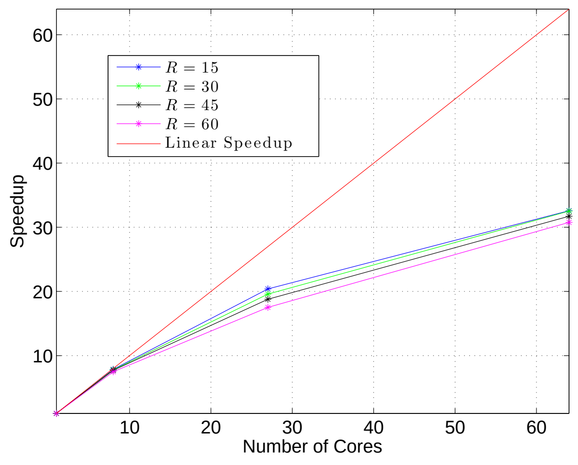

Fig. 2. Speedup diagram of the Nesterov-based AO NTF algorithm for $I = J = K = 1008$ and rank $R = 15, 30, 45, 60$.

5. CONCLUSION

We considered the NTF problem. We adopted the AO framework and solved each MNLS problem via a Nesterov-type algorithm for L-smooth and μ-strongly convex problems. We presented a parallel implementation of the algorithm and measured the attained speedup for a large-scale dense NTF problem. The proposed algorithm seems a strong alternative to state-of-the-art algorithms for the solution of very large-scale dense NTF problems.

6. REFERENCES

[1] P. M. Kroonenberg, *Applied Multiway Data Analysis*. Wiley-Interscience, 2008.

[2] A. Cichocki, R. Zdunek, A. H. Phan, and S. Amari, *Nonnegative Matrix and Tensor Factorizations*. Wiley, 2009.

[3] L. Karlsson, D. Kressner, and A. Uschmajew, "Parallel algorithms for tensor completion in the CP format," *Parallel Computing*, 2015.

[4] S. Smith and G. Karypis, "A medium-grained algorithm for distributed sparse tensor factorization," *30th IEEE International Parallel & Distributed Processing Symposium*, 2016.

[5] O. Kaya and B. Uçar, "Scalable sparse tensor decompositions in distributed memory systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2015, p. 77.

[6] A. P. Liavas, G. Kostoulas, G. Lourakis, K. Huang, and N. D. Sidiropoulos, "Nesterov-based alternating optimization for nonnegative tensor factorization: Algorithm and parallel implementations," *submitted to the IEEE Transactions on Signal Processing*.

[7] Y. Nesterov, *Introductory lectures on convex optimization*. Kluwer Academic Publishers, 2004.

[8] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, September 2009.

[9] Y. Xu and W. Yin, "A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion," *SIAM Journal on imaging sciences*, vol. 6, no. 3, pp. 1758–1789, 2013.

[10] M. Razaviyayn, M. Hong, and Z.-Q. Luo, "A unified convergence analysis of block successive minimization methods for nonsmooth optimization," *SIAM Journal on Optimization*, vol. 23, no. 2, pp. 1126–1153, 2013.

[11] N. Guan, D. Tao, Z. Luo, and B. Yuan, "Nenmf: An optimal gradient method for nonnegative matrix factorization," *IEEE Transactions on Signal Processing*, vol. 60, no. 6, pp. 2882–2898, 2012.

[12] G. Zhou, Q. Zhao, Y. Zhang, and A. Cichocki, "Fast nonnegative tensor factorization by using accelerated proximal gradient," in *Advances in Neural Networks – ISNN 2014*, Z. Zeng, Y. Li, and I. King, Eds. Cham: Springer International Publishing, 2014, pp. 459–468.

[13] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.