Alternating Optimization for Tensor Factorization with Orthogonality Constraints: Algorithm and Parallel Implementation

Paris A. Karakasis
School of Electrical and Computer Engineering
Technical University of Crete, Greece
Email: pkarakasis@isc.tuc.gr

Athanasios P. Liavas
School of Electrical and Computer Engineering
Technical University of Crete, Greece
Email: aliavas@isc.tuc.gr

***Abstract*—We consider the problem of tensor factorization in the cases where one of the factors is constrained to have orthonormal columns. We adopt the alternating optimization framework and derive an efficient algorithm that is also suitable for parallel implementation. We describe in detail a distributed memory implementation of the algorithm on a three-dimensional processor grid. The speedup attained by a message-passing implementation of the algorithm is significant, indicating that it is a competitive candidate for the solution of very large tensor factorization problems with orthogonality constraints.**

***Index Terms*—tensors, tensor factorization, PARAFAC, orthogonality constraints, algorithms, parallel algorithms.**

I. Introduction

Tensors have recently gained great popularity due to their ability to model multiway data dependencies [1], [2], [3], [4]. Tensor factorizations into latent factors are very important for numerous tasks, such as feature selection, dimensionality reduction, compression, data visualization, interpretation and completion, 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.

The PARAFAC model comes with theoretical background that guarantees essentially unique tensor factorizations under mild conditions. However, the problem of finding a best rank-R approximation of tensors of order 3 (three-way), in the unconstrained case, has no solution, in general [5]. Existence of an optimal solution is guaranteed if one of the factors is constrained to have orthonormal columns [6]. Also, the orthonormally constrained PARAFAC model can be unique under more relaxed conditions than the unconstrained model [7]. In this work, we focus on the PARAFAC model with unimodal orthogonality constraints.

Alternating Optimization (AO) and All-at-Once Optimization (AOO) are among the most commonly used techniques for tensor factorization [2], [8]. Recent work for constrained tensor factorization/completion includes, among others, [9], [10], [11], and [12]. In [12], the authors consider constrained matrix/tensor factorization/completion problems. They adopt the AO framework as outer loop and use the Alternating Direction Method of Multipliers (ADMM) for solving the inner constrained optimization problem for one matrix factor conditioned on the rest. The ADMM offers significant flexibility, due to its ability to efficiently handle a wide range of constraints.

In [13], two parallel algorithms for unconstrained tensor factorization/completion have been developed and results concerning the speedup attained by their Message Passing Interface (MPI) implementations on a multi-core system have been reported. Related work on parallel algorithms for sparse tensor decomposition includes [14] and [15].

A. Contribution

In this work, we focus on large tensor factorization problems with one of the factors constrained to have orthonormal columns. Our aim is to derive an efficient algorithm that is also suitable for parallel implementation. We adopt the AO framework, and develop an algorithm for the solution of the aforementioned problem. We describe in detail a parallel implementation of the algorithm on a three-dimensional processor grid [1] and measure the speedup attained by an MPI implementation of the algorithm. We observe that the proposed algorithm is very efficient, in practice.

B. 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 tensors, while $\mathbb{R}^{I\times J}$ denotes the set of $(I \times J)$ real matrices. $\mathbf{I}$ denotes the identity matrix of appropriate dimensions. $\mathbb{S}_{(I,J)} = \left\{\mathbf{X} \in \mathbb{R}^{I\times J} : \mathbf{X}^T\mathbf{X} = \mathbf{I}\right\}$ denotes the Stiefel manifold formed by all orthonormal J-frames in $\mathbb{R}^I$. $\|\cdot\|_F$ denotes the Frobenius norm of the tensor or matrix argument. 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

This work was supported by computational time granted from the Greek Research & Technology Network (GRNET) in the National HPC facility - ARIS - under project ID pa171201.

[1]The MPI C++ implementation of the algorithm, that we used in our experiments, is available upon request to the authors.

elements $(\mathbf{a} \circ \mathbf{b} \circ \mathbf{c})(i,j,k) = \mathbf{a}(i)\mathbf{b}(j)\mathbf{c}(k)$. The Khatri-Rao (columnwise Kronecker) product of compatible matrices $\mathbf{A}$ and $\mathbf{B}$ is denoted as $\mathbf{A} \odot \mathbf{B}$ and the Hadamard (elementwise) product is denoted as $\mathbf{A} \circledast \mathbf{B}$. Finally, inequality $\mathbf{A} \succeq \mathbf{B}$ means that matrix $\mathbf{A} - \mathbf{B}$ is positive semidefinite.

C. Structure

In Section II, we briefly describe the tensor factorization problem with uni-modal orthogonality constraints. In Section III, we briefly describe the Orthogonal Procrustes problem. In Section IV, we derive the AO algorithm and in Section V we describe in detail a parallel implementation of the algorithm. In Section VI, we test the efficiency of the proposed algorithm with numerical experiments in a parallel computing environment. Finally, in Section VII, we conclude the paper.

II. Tensor Factorization with Uni-modal Orthogonality Constraints

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

$$\min_{\mathbf{A}, \mathbf{B} \in \mathbb{S}_{(J,R)}, \mathbf{C}} \quad f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}, \mathbf{B}, \mathbf{C}), \tag{2}$$

where $f_{\boldsymbol{\mathcal{X}}}$ is a function measuring the quality of the factorization. A common choice for $f_{\boldsymbol{\mathcal{X}}}$ is

$$f_{\boldsymbol{\mathcal{X}}}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \frac{1}{2} \| \boldsymbol{\mathcal{X}} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] \|_F^2 . \tag{3}$$

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 [3]

$$\begin{gathered} \mathbf{Y}_{\mathbf{A}} = \mathbf{A}(\mathbf{C} \odot \mathbf{B})^T, \ \mathbf{Y}_{\mathbf{B}} = \mathbf{B}(\mathbf{C} \odot \mathbf{A})^T, \\ \mathbf{Y}_{\mathbf{C}} = \mathbf{C}(\mathbf{B} \odot \mathbf{A})^T. \end{gathered} \tag{4}$$

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

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

These expressions form the basis for the AO algorithm in the sense that, if we fix two matrix factors, we can update the third by solving a (potentially constrained) least squares problem. The update of matrix factor $\mathbf{B}$ requires the solution of a least squares problem with orthogonality constraints.

III. Orthogonal Procrustes

Given two matrices $\mathbf{Y} \in \mathbb{R}^{N \times M}$ and $\mathbf{X} \in \mathbb{R}^{M \times D}$, the optimization problem

$$\min_{\mathbf{G} \in \mathbb{S}_{(N,D)}} \quad \left\| \mathbf{Y} - \mathbf{G}\mathbf{X}^T \right\|_F^2 , \tag{5}$$

is known as Orthogonal Procrustes (OP) and has a closed form solution given by [16], [17]

$$\mathbf{G}_{\text{opt}} = \mathbf{U}\mathbf{V}^T = \mathbf{M}\left(\mathbf{M}^T\mathbf{M}\right)^{-\frac{1}{2}} , \tag{6}$$

where matrices $\mathbf{U} \in \mathbb{R}^{N \times D}$ and $\mathbf{V} \in \mathbb{R}^{D \times D}$ are given by the singular value decomposition of matrix $\mathbf{M} = \mathbf{Y}\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$.

A. Computational complexity of the OP problem

For later use, we notice that an efficient way of solving the OP problem, after calculating matrix $\mathbf{M}$ with computational complexity $\mathcal{O}(NMD)$ arithmetic operations and when $\min(N, M) > D$, is the following algorithm:

1) Calculate $\mathbf{M}^T\mathbf{M}$, with complexity $O\left(ND^2\right)$;
2) Calculate the eigen-decomposition of $\mathbf{M}^T\mathbf{M} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{V}^T$ with complexity $O\left(D^3\right)$;
3) Set $\mathbf{G} = \mathbf{M}\mathbf{V}\boldsymbol{\Sigma}^{-\frac{1}{2}}\mathbf{V}^T$ with complexity $O\left(ND^2\right)$.

Thus, the overall complexity is $\mathcal{O}\left(ND^2\right)$ in contrast to computing the singular value decomposition of matrix $\mathbf{M}$ in $O\left(N^2D\right)$. The most demanding computation of this approach is the computation of matrix $\mathbf{M}$.

IV. AO Uni-Modal Orthogonal Tensor Factorization

In Algorithm 1, we present the AO Uni-modal Orthogonal Tensor Factorization (AO UOTF) algorithm. We start from point $(\mathbf{A}_0, \mathbf{B}_0, \mathbf{C}_0)$ and solve, in a circular manner, a least squares problem (via function LS_Update) for updating factors $\mathbf{A}_k$ and $\mathbf{C}_k$, while we update factor $\mathbf{B}_k$ by solving the OP problem (via function OP_Update).

Updating factors $\mathbf{A}_k$ and $\mathbf{C}_k$ can be done as follows:

$$\begin{aligned} \mathbf{A}_{k+1} &= \mathbf{X}_{\mathbf{A}}(\mathbf{C}_k \odot \mathbf{B}_k)\left[\left(\mathbf{C}_k^T\mathbf{C}_k\right) \circledast \left(\mathbf{B}_k^T\mathbf{B}_k\right)\right]^{-1} \\ &= \mathbf{X}_{\mathbf{A}}(\mathbf{C}_k \odot \mathbf{B}_k)\left[\left(\mathbf{C}_k^T\mathbf{C}_k\right) \circledast \mathbf{I}\right]^{-1} \\ &= \mathbf{X}_{\mathbf{A}}(\mathbf{C}_k \odot \mathbf{B}_k)\mathbf{D}_{\mathbf{C}_k^T\mathbf{C}_k}^{-1} \end{aligned} \tag{7}$$

and

$$\begin{aligned} \mathbf{C}_{k+1} &= \mathbf{X}_{\mathbf{C}}(\mathbf{B}_{k+1} \odot \mathbf{A}_{k+1})\left[\mathbf{I} \circledast \left(\mathbf{A}_{k+1}^T\mathbf{A}_{k+1}\right)\right]^{-1} \\ &= \mathbf{X}_{\mathbf{C}}(\mathbf{B}_{k+1} \odot \mathbf{A}_{k+1})\mathbf{D}_{\mathbf{A}_{k+1}^T\mathbf{A}_{k+1}}^{-1}, \end{aligned} \tag{8}$$

that solve the corresponding least squares problems exploiting the orthonormality of factor $\mathbf{B}$. For a matrix $\mathbf{G}$, matrix $\mathbf{D}_{\mathbf{G}^T\mathbf{G}}^{-1}$ denotes the diagonal matrix with elements the inverses of the corresponding diagonal elements of matrix $\mathbf{G}^T\mathbf{G}$.

After the factor updates, we use two functions which have been proven very useful in our experiments, in the sense that they significantly reduce the number of outer iterations necessary to reach convergence. Function Normalize normalizes each column of $\mathbf{A}_{k+1}$ to unit Euclidean norm, putting all the power on the respective columns of $\mathbf{C}_{k+1}$. We denote its

Algorithm 1: AO UOTF

```
Input: $\boldsymbol{\mathcal{X}}$, $\mathbf{A}_0$, $\mathbf{B}_0$, $\mathbf{C}_0$.
1  Set $k = 0$
2  while $(1)$ do
3      $\mathbf{A}_{k+1} = \text{LS\_Update}(\mathbf{X}_{\mathbf{A}}, \mathbf{C}_k \odot \mathbf{B}_k,)$
4      $\mathbf{B}_{k+1} = \text{OP\_Update}(\mathbf{X}_{\mathbf{B}}, \mathbf{C}_k \odot \mathbf{A}_{k+1})$
5      $\mathbf{C}_{k+1} = \text{LS\_Update}(\mathbf{X}_{\mathbf{C}}, \mathbf{A}_{k+1} \odot \mathbf{B}_{k+1})$
6      $(\mathbf{A}^{\mathcal{N}}_{k+1}, \mathbf{C}^{\mathcal{N}}_{k+1}) = \text{Normalize}(\mathbf{A}_{k+1}, \mathbf{C}_{k+1})$
7      if (terminating_condition is TRUE) then break; endif
8      $(\mathbf{A}_{k+1}, \mathbf{B}_{k+1}, \mathbf{C}_{k+1}) = \text{Accelerate}(\mathbf{A}^{\mathcal{N}}_{k+1}, \mathbf{A}^{\mathcal{N}}_{k}, \mathbf{B}_{k+1}, \mathbf{B}_k, \mathbf{C}^{\mathcal{N}}_{k+1}, \mathbf{C}^{\mathcal{N}}_{k})$
9      $k = k + 1$
10 return $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{C}_k$.
```

$$\boldsymbol{\mathcal{X}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}} := \boldsymbol{\mathcal{X}}\left((i_{\mathbf{A}}-1)\frac{I}{p_{\mathbf{A}}}+1 : i_{\mathbf{A}}\frac{I}{p_{\mathbf{A}}}, (i_{\mathbf{B}}-1)\frac{J}{p_{\mathbf{B}}}+1 : i_{\mathbf{B}}\frac{J}{p_{\mathbf{B}}}, (i_{\mathbf{C}}-1)\frac{K}{p_{\mathbf{C}}}+1 : i_{\mathbf{C}}\frac{K}{p_{\mathbf{C}}}\right). \tag{9}$$

output as $\mathbf{A}^{\mathcal{N}}_{k+1}$ and $\mathbf{C}^{\mathcal{N}}_{k+1}$. Function Accelerate implements the acceleration technique used in the function `parafac` of n-way toolbox [18], briefly described in [19].

We can use various termination criteria for the AO UOTF algorithm based, for example, on the (relative) change of the cost function and/or the latent factors.

V. Parallel Implementation

In this section, we consider the implementation of the AO UOTF algorithm in a computing environment with $p = p_A \times p_B \times p_C$ processing elements. In general, the p processors form a three-dimensional Cartesian grid, with each processor 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 $i_{\mathbf{C}} = 1,\ldots,p_{\mathbf{C}}$.

A. Variable partitionings and data allocation

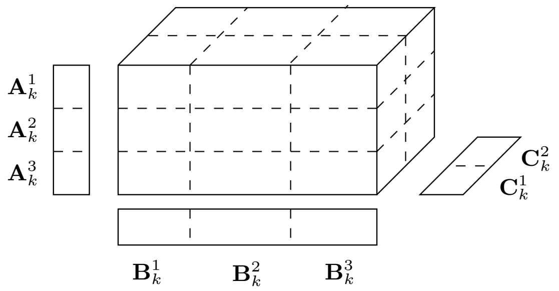

Fig. 1. Tensor $\boldsymbol{\mathcal{X}}$, factors $\mathbf{A}_k$, $\mathbf{B}_k$, and $\mathbf{C}_k$, and their partitioning for $p_{\mathbf{A}} = p_{\mathbf{B}} = 3$ and $p_{\mathbf{C}} = 2$.

In order to describe the parallel implementation, we introduce certain partitionings of the factor matrices and the tensor matricizations. We partition the factor matrix $\mathbf{A}_k$ into $p_{\mathbf{A}}$ block rows as

$$\mathbf{A}_k = \left[\begin{array}{ccc} \left(\mathbf{A}_k^1\right)^T & \cdots & \left(\mathbf{A}_k^{N_p}\right)^T \end{array}\right]^T, \tag{10}$$

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

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

with $\mathbf{X}_{\mathbf{A}}^{i_{\mathbf{A}}} \in \mathbb{R}^{\frac{I}{p_{\mathbf{A}}}\times JK}$, for $i_{\mathbf{A}} = 1,\ldots,p_{\mathbf{A}}$. In a similar manner, we partition $\mathbf{B}_k$ and $\mathbf{X}_{\mathbf{B}}$ into $p_{\mathbf{B}}$ block rows, each of size $\frac{J}{p_{\mathbf{B}}} \times R$ and $\frac{J}{p_{\mathbf{B}}} \times IK$, respectively, and $\mathbf{C}_k$ and $\mathbf{X}_{\mathbf{C}}$ into $p_{\mathbf{C}}$ block rows, each of size $\frac{K}{p_{\mathbf{C}}} \times R$ and $\frac{K}{p_{\mathbf{C}}} \times IJ$, respectively.

We partition tensor $\boldsymbol{\mathcal{X}}$ into p subtensors, according to the partitioning of the factor matrices (see Fig. 1), and allocate its parts to the various processors. Thus, processor $p_{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$ receives subtensor $\boldsymbol{\mathcal{X}}^{i_{\mathbf{A}},i_{\mathbf{B}},i_{\mathbf{C}}}$, as defined in (9), at the top of this page.

We assume that, at the end of the k-th outer AO iteration,

a) 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 $\mathbf{C}_k^{i_{\mathbf{C}}}$;

b) *all* processors know $\mathbf{A}_k^T\mathbf{A}_k$ and $\mathbf{C}_k^T\mathbf{C}_k$ (note that, due the orthogonality constraints $\mathbf{B}_k^T\mathbf{B}_k = \mathbf{I}$).

B. Communication Groups

We start by defining certain communication groups, also known as communicators [20], over subsets of the p processors. The communicators are used for the efficient collaborative implementation of specific computational tasks, as explained in detail later.

We define $p_{\mathbf{A}}$ two-dimensional processor groups, each involving the $p_{\mathbf{B}} \times p_{\mathbf{C}}$ processors $p_{i_{\mathbf{A}},:,:}$, for $i_{\mathbf{A}} = 1,\ldots,p_{\mathbf{A}}$ (horizontal layers), with the $i_{\mathbf{A}}$-th processor group used for the collaborative update of $\mathbf{A}_k^{i_{\mathbf{A}}}$. Similarly, we define 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 collaborative update of $\mathbf{B}_k^{i_{\mathbf{B}}}$ and $\mathbf{C}_k^{i_{\mathbf{C}}}$, respectively.

We define $p_{\mathbf{B}} \times p_{\mathbf{C}}$ one-dimensional processor groups, each involving the $p_{\mathbf{A}}$ processors $p_{:,i_{\mathbf{B}},i_{\mathbf{C}}}$. Each of these groups is used for the collaborative computation of $\mathbf{A}_{k+1}^T\mathbf{A}_{k+1}$. Similarly, we define groups $p_{i_{\mathbf{A}},:,i_{\mathbf{C}}}$ and $p_{i_{\mathbf{A}},i_{\mathbf{B}},:}$, which are used for the collaborative update of $\mathbf{B}_k$ and computation of $\mathbf{C}_{k+1}^T\mathbf{C}_{k+1}$, respectively.

C. Factor Update Implementation

We describe in detail the updates of $\mathbf{A}_k$ and $\mathbf{B}_k$. The update of $\mathbf{C}_k$ is similar to the update of $\mathbf{A}_k$.

Collaborative Update of $\mathbf{A}_k$*:* The update of $\mathbf{A}_k$ is achieved via the parallel updates of $\mathbf{A}_k^{i_\mathbf{A}}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$, and consists of the following stages:

1) Processors $p_{i_\mathbf{A},:,:}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$, collaboratively compute the $\frac{I}{p_\mathbf{A}} \times R$ matrix
$$\widetilde{\mathbf{W}}_\mathbf{A}^{i_\mathbf{A}} := \mathbf{X}_\mathbf{A}^{i_\mathbf{A}} \left(\mathbf{C}_k \odot \mathbf{B}_k \right), \tag{12}$$
and the result is scattered among the processors in the group; thus, each processor in the group receives $\frac{I}{p_\mathbf{A} p_\mathbf{B} p_\mathbf{C}}$ successive rows of $\widetilde{\mathbf{W}}_\mathbf{A}^{i_\mathbf{A}}$. Term $\widetilde{\mathbf{W}}_\mathbf{A}^{i_\mathbf{A}}$ can be computed collaboratively because
$$\mathbf{X}_\mathbf{A}^{i_\mathbf{A}} \left(\mathbf{C}_k \odot \mathbf{B}_k \right) = \sum_{i_\mathbf{B}=1}^{p_\mathbf{B}} \sum_{i_\mathbf{C}=1}^{p_\mathbf{C}} \mathbf{X}_\mathbf{A}^{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}} \left(\mathbf{C}_k^{i_\mathbf{C}} \odot \mathbf{B}_k^{i_\mathbf{B}} \right), \tag{13}$$
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 $p_{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$ knows $\mathbf{X}_\mathbf{A}^{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$, $\mathbf{B}_k^{i_\mathbf{B}}$, and $\mathbf{C}_k^{i_\mathbf{C}}$, and computes the corresponding term of (13). The sum is computed and scattered among processors $p_{i_\mathbf{A},:,:}$ via a reduce-scatter operation.
2) Each processor in the group $p_{i_\mathbf{A},:,:}$ uses the scattered part of $\widetilde{\mathbf{W}}_\mathbf{A}^{i_\mathbf{A}}$ and $\widetilde{\mathbf{Z}}_\mathbf{A} = \mathbf{I} \circledast \mathbf{C}_k^T \mathbf{C}_k$, and computes the updated part of $\mathbf{A}_{k+1}^{i_\mathbf{A}}$, via a least-squares update. Then, the updated parts are all-gathered at the processors of the group $p_{i_\mathbf{A},:,:}$, so that *all* processors in the group learn the whole updated $\mathbf{A}_{k+1}^{i_\mathbf{A}}$.
3) By applying an all-reduce operation to $\left(\mathbf{A}_{k+1}^{i_\mathbf{A}}\right)^T \mathbf{A}_{k+1}^{i_\mathbf{A}}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$, on each of the single dimensional processor groups $p_{:, i_\mathbf{B}, i_\mathbf{C}}$, for $i_\mathbf{B} = 1, \ldots, p_\mathbf{B}$ and $i_\mathbf{C} = 1, \ldots, p_\mathbf{C}$, *all* p processors learn $\mathbf{A}_{k+1}^T \mathbf{A}_{k+1}$.

Collaborative Update of $\mathbf{B}_k$*:* The update of $\mathbf{B}_k$ is achieved via the parallel updates of $\mathbf{B}_k^{i_\mathbf{B}}$, for $i_\mathbf{B} = 1, \ldots, p_\mathbf{B}$, and consists of the following stages:

1) Processors $p_{:, i_\mathbf{B}, :}$, for $i_\mathbf{B} = 1, \ldots, p_\mathbf{B}$, collaboratively compute the $\frac{J}{p_\mathbf{B}} \times R$ matrix
$$\widetilde{\mathbf{W}}_\mathbf{B}^{i_\mathbf{B}} := \mathbf{X}_\mathbf{B}^{i_\mathbf{B}} \left(\mathbf{C}_k \odot \mathbf{A}_{k+1} \right), \tag{14}$$
by applying an all-reduce operation, since
$$\mathbf{X}_\mathbf{B}^{i_\mathbf{B}} \left(\mathbf{C}_k \odot \mathbf{A}_{k+1} \right) = \sum_{i_\mathbf{A}=1}^{p_\mathbf{A}} \sum_{i_\mathbf{C}=1}^{p_\mathbf{C}} \mathbf{X}_\mathbf{B}^{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}} \left(\mathbf{C}_k^{i_\mathbf{C}} \odot \mathbf{A}_{k+1}^{i_\mathbf{A}} \right),$$
where $\mathbf{X}_\mathbf{B}^{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$ is the matricization of $\boldsymbol{\mathcal{X}}^{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$, with respect to the second mode.
2) Processors $p_{i_\mathbf{A}, :, i_\mathbf{C}}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$ and $i_\mathbf{C} = 1, \ldots, p_\mathbf{C}$, collaboratively compute the $R \times R$ matrix
$$\widetilde{\mathbf{W}}_\mathbf{B}^T \widetilde{\mathbf{W}}_\mathbf{B} = \sum_{i_\mathbf{B}=1}^{p_\mathbf{B}} \left(\widetilde{\mathbf{W}}_\mathbf{B}^{i_\mathbf{B}} \right)^T \widetilde{\mathbf{W}}_\mathbf{B}^{i_\mathbf{B}} \tag{15}$$
by applying an all-reduce operation. We notice that at the end of this step, *all* p processors know matrix $\widetilde{\mathbf{W}}_\mathbf{B}^T \widetilde{\mathbf{W}}_\mathbf{B}$.
3) Each processor $p_{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$, $i_\mathbf{B} = 1, \ldots, p_\mathbf{B}$, and $i_\mathbf{C} = 1, \ldots, p_\mathbf{C}$, computes the updated partial factor $\mathbf{B}_{k+1}^{i_\mathbf{B}}$ as
$$\mathbf{B}_{k+1}^{i_\mathbf{B}} = \widetilde{\mathbf{W}}_\mathbf{B}^{i_\mathbf{B}} \left(\widetilde{\mathbf{W}}_\mathbf{B}^T \widetilde{\mathbf{W}}_\mathbf{B} \right)^{-\frac{1}{2}}. \tag{16}$$

We note that the Euclidean norms of the columns of $\mathbf{A}_{k+1}$ and $\mathbf{C}_{k+1}$ appear on the diagonals of $\mathbf{A}_{k+1}^T \mathbf{A}_{k+1}$ and $\mathbf{C}_{k+1}^T \mathbf{C}_{k+1}$, which are known to all processors. Thus, no additional communication is necessary for the normalization of the updated matrix factors.

After the normalization step of the $(k+1)$-st AO iteration, processor $p_{i_\mathbf{A}, i_\mathbf{B}, i_\mathbf{C}}$ knows the parts of the normalized factors, that is, $\mathbf{A}_{k+1}^{i_\mathbf{A} \mathcal{N}}$, $\mathbf{B}_{k+1}^{i_\mathbf{B}}$, $\mathbf{C}_{k+1}^{i_\mathbf{C} \mathcal{N}}$, as well as $\mathbf{A}_k^{i_\mathbf{A} \mathcal{N}}$, $\mathbf{B}_k^{i_\mathbf{B}}$, and $\mathbf{C}_k^{i_\mathbf{C} \mathcal{N}}$, and can collaboratively implement the acceleration mechanism as explained in detail in [19].

D. Communication Cost

We focus on the parallel updates of $\mathbf{A}_k^{i_\mathbf{A}}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$, and $\mathbf{B}_k^{i_\mathbf{B}}$, for $i_\mathbf{B} = 1, \ldots, p_\mathbf{B}$, and present results concerning the associated communication cost. The communication cost of the update of $\mathbf{C}_k^{i_\mathbf{C}}$, for $i_\mathbf{C} = 1, \ldots, p_\mathbf{C}$, can be computed by following analogous steps to those used for the computation of the communication cost of the update of $\mathbf{A}_k^{i_\mathbf{A}}$.

We assume that an m-word message is transferred from one process to another with communication cost $t_s + t_w m$, where t_s is the latency, or startup time for the data transfer, and t_w is the word transfer time [20].

Updating $\mathbf{A}_k$ *in parallel:* Communication occurs at three algorithm execution points.

1) The $\frac{I}{p_\mathbf{A}} \times R$ matrix $\widetilde{\mathbf{W}}_\mathbf{A}^{i_\mathbf{A}}$ is computed and scattered among the $p_\mathbf{B} \times p_\mathbf{C}$ processors of group $p_{i_\mathbf{A},:,:}$, using a reduce-scatter operation, with communication cost [20, §4.2]
$$\mathcal{C}_1^\mathbf{A} = t_s \left(p_\mathbf{B} + p_\mathbf{C} - 2 \right) + t_w \frac{IR}{p_\mathbf{A} p_\mathbf{B} p_\mathbf{C}} \left(p_\mathbf{B} p_\mathbf{C} - 1 \right).$$
2) Processors $p_{i_\mathbf{A},:,:}$ learn the updated $\mathbf{A}_{k+1}^{i_\mathbf{A}}$ through an all-gather operation on its updated parts, each of dimension $\frac{I}{p_\mathbf{A} p_\mathbf{B} p_\mathbf{C}} \times R$, with communication cost [20, §4.2]
$$\mathcal{C}_2^\mathbf{A} = t_s \left(p_\mathbf{B} + p_\mathbf{C} - 2 \right) + t_w \frac{IR}{p_\mathbf{A} p_\mathbf{B} p_\mathbf{C}} \left(p_\mathbf{B} p_\mathbf{C} - 1 \right).$$
3) Matrix $\mathbf{A}_{k+1}^T \mathbf{A}_{k+1}$ is computed by using an all-reduce operation on quantities $\left(\mathbf{A}_{k+1}^{i_\mathbf{A}}\right)^T \mathbf{A}_{k+1}^{i_\mathbf{A}}$, for $i_\mathbf{A} = 1, \ldots, p_\mathbf{A}$, on each single-dimensional processor group $p_{:, i_\mathbf{B}, i_\mathbf{C}}$, with communication cost [20, §4.3]
$$\mathcal{C}_3^\mathbf{A} = \left(t_s + t_w R^2 \right) \log_2 p_\mathbf{A}. \tag{17}$$

Updating $\mathbf{B}_k$ *in parallel:* Communication occurs at two algorithm execution points.

1) The $\frac{J}{p_\mathbf{B}} \times R$ matrix $\widetilde{\mathbf{W}}_\mathbf{B}^{i_\mathbf{B}}$ is computed among the $p_\mathbf{A} \times p_\mathbf{C}$ processors of group $p_{:, i_\mathbf{B}, :}$, using an all-reduce operation, with communication cost [20, §4.2]
$$\mathcal{C}_1^\mathbf{B} = \left(t_s + t_w \frac{J}{p_\mathbf{B}} R \right) \log_2 \left(p_\mathbf{A} p_\mathbf{C} \right).$$

2) Matrix $\widetilde{\mathbf{W}}_{\mathbf{B}}^T\widetilde{\mathbf{W}}_{\mathbf{B}}$ is computed by using an all-reduce operation on quantities $\left(\widetilde{\mathbf{W}}_{\mathbf{B}}^{i_{\mathbf{B}}}\right)^T \widetilde{\mathbf{W}}_{\mathbf{B}}^{i_{\mathbf{B}}}$ within each single-dimensional processor group $p_{i_{\mathbf{A}},:,i_{\mathbf{C}}}$, with communication cost [20, §4.3]

$$\mathcal{C}_2^{\mathbf{B}} = \left(t_s + t_w R^2\right) \log_2 p_{\mathbf{B}}. \tag{18}$$

The communication that takes place during the acceleration step involves scalar quantities and, thus, is ignored.

When we are dealing with large messages, the t_w terms dominate the communication cost. Thus, if we ignore the startup time, the total communication time, for updating $\mathbf{A}_k$, is

$$\begin{aligned}\mathcal{C}^{\mathbf{A}} &= t_w \left(\frac{2IR}{p_{\mathbf{A}}\, p_{\mathbf{B}}\, p_{\mathbf{C}}}\left(p_{\mathbf{B}} p_{\mathbf{C}} - 1\right) + R^2 \log_2 p_{\mathbf{A}}\right) \\ &\approx t_w \left(\frac{2IR}{p_{\mathbf{A}}} + R^2 \log_2 p_{\mathbf{A}}\right) \\ &\approx \frac{2IR\, t_w}{p_{\mathbf{A}}},\end{aligned} \tag{19}$$

with the second approximation being accurate for $R \ll \frac{I}{p_{\mathbf{A}}}$. The presence of $p_{\mathbf{A}}$ in the denominator of the last expression of (19) implies that our implementation is scalable in the sense that, if we double I, then we can have (approximately) the same communication cost per processor by doubling $p_{\mathbf{A}}$. Analogous results hold for the update of factor $\mathbf{C}_k$.

As for the update of factor $\mathbf{B}_k$, if we ignore the startup time, the total communication time is

$$\begin{aligned}\mathcal{C}^{\mathbf{B}} &= t_w \left(\frac{JR}{p_{\mathbf{B}}} \log_2\left(p_{\mathbf{A}} p_{\mathbf{C}}\right) + R^2 \log_2 p_{\mathbf{B}}\right) \\ &\approx t_w \left(\frac{JR}{p_{\mathbf{B}}} \log_2\left(p_{\mathbf{A}} p_{\mathbf{C}}\right)\right)\end{aligned} \tag{20}$$

with the approximation being accurate for $R \ll \frac{J}{p_{\mathbf{B}}}$. We again observe that our implementation is scalable in the above mentioned sense.

VI. Numerical experiments

In this section, we present results obtained from the MPI implementation described in detail in Section V. The program is executed on a DELL PowerEdge R820 system with SandyBridge - Intel(R) Xeon(R) CPU E5 $-$ 4650v2 (in total, 16 nodes with 40 cores each at 2.4 Gz) and 512 GB RAM per node. The matrix operations are implemented using routines of the C++ library Eigen [21]. We assume a noiseless tensor $\boldsymbol{\mathcal{X}}$, whose true latent factors $\mathbf{A}^o$ and $\mathbf{C}^o$ have i.i.d elements, uniformly distributed in $[0,1]$, while true latent factor $\mathbf{B}^o$ was produced from the left singular vectors of a matrix with i.i.d elements, uniformly distributed in $[0,1]$.

The AO terminates at iteration k if

$$\texttt{RFE}(\mathbf{A}_k, \mathbf{B}_k, \mathbf{C}_k) < 10^{-3},$$

where

$$\text{RFE}\left(\mathbf{A}, \mathbf{B}, \mathbf{C}\right) := \frac{\|\boldsymbol{\mathcal{X}} - [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!]\|_F}{\|\boldsymbol{\mathcal{X}}\|_F}. \tag{21}$$

We test the behavior of our implementation for various tensor sizes and rank $R = 15, 50, 100$. The performance metric we compute is the speedup attained using $p = p_{\mathbf{A}} \times p_{\mathbf{B}} \times p_{\mathbf{C}}$ processors.

In Figures 2–4, we plot the speedup for the following cases[2]:

1) Cubic tensor: we set $I = J = K = 2000$ and implement the algorithm on a grid with $p_{\mathbf{A}} = p_{\mathbf{B}} = p_{\mathbf{C}} = \sqrt[3]{p}$, for $p = 1, 8, 27, 64, 125, 216, 343$.
2) Two large dimensions: we set $I = 5000$, $J = 320$, $K = 5000$ and implement the algorithm on a grid with $p_{\mathbf{A}} = p_{\mathbf{C}} = \sqrt{p}$, $p_{\mathbf{B}} = 1$, for $p = 1, 4, 9, 36, 64, 121, 225, 361$.
3) One large dimension: we set $I = 400$, $J = 50000$, $K = 400$ and implement the algorithm on a grid with $p_{\mathbf{A}} = p_{\mathbf{C}} = 1$, $p_{\mathbf{B}} = p$, for $p = 1, 8, 27, 64, 125, 216, 343$.

In order to highlight the need of parallel processing for the decomposition of very large tensors, we quote the serial execution times ($p = 1$) in Table I. We observe that, in all cases, we attain significant speedup, which is rather insensitive to the tensor shape and rank.

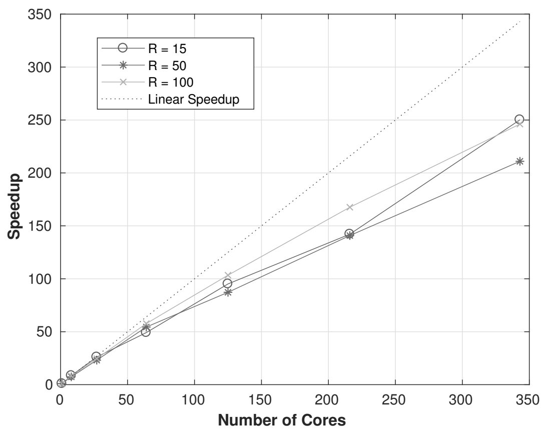

Fig. 2. Speedup achieved for a $2000 \times 2000 \times 2000$ tensor with p cores, for $p = 1, 8, 27, 64, 125, 216, 343$.

VII. Conclusion

We considered the UOTF problem. We adopted the AO framework and described in detail a parallel implementation of the AO UOTF algorithm on a three-dimensional processor grid. The speedup attained by the MPI implementation of the algorithm was significant in all the cases we considered, rendering our algorithm a strong candidate for the solution of very large-scale dense UOTF problems.

Future work includes the development and implementation of efficient algorithms for UOTF with further constraints, like nonnegativity and sparsity.

[2]To the best of our knowledge, there is no other parallel algorithm solving the UOTF problem, thus, we cannot compare with any competing state-of-the art algorithm.

TABLE I
EXECUTION TIMES FOR $p = 1$ OVER DIFFERENT TENSOR SIZES AND RANKS

Size	R	Time of Execution (sec)
$2000 \times 2000 \times 2000$	15	$6,272.35$
	50	14744.09
	100	34381.45
$5000 \times 320 \times 5000$	15	$7,545.13$
	50	17908.56
	100	42628.99
$400 \times 5000 \times 400$	15	$6,434.69$
	50	$17,790.75$
	100	$36,599.60$

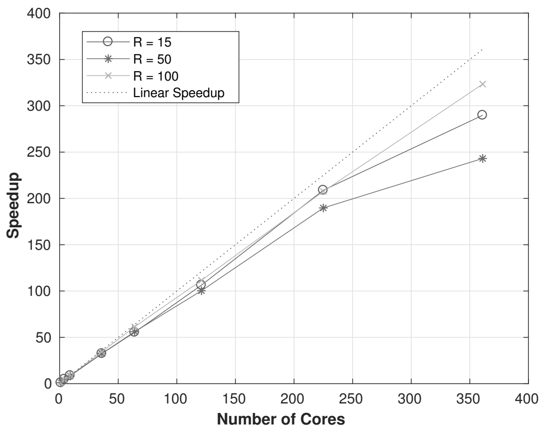

Fig. 3. Speedup achieved for a $5000 \times 320 \times 5000$ tensor with p cores, for $p = 1, 4, 9, 36, 64, 121, 225, 361$.

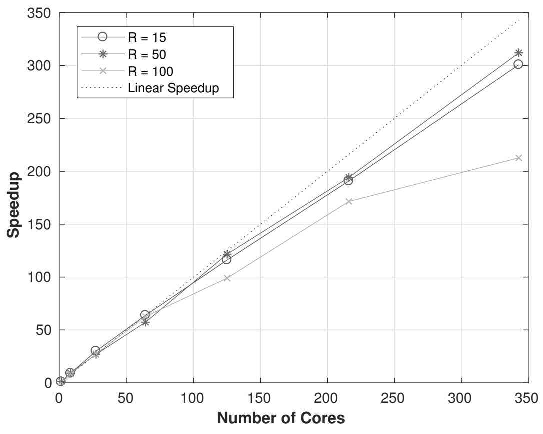

Fig. 4. Speedup achieved for a $400 \times 50000 \times 400$ tensor with p cores, for $p = 1, 8, 27, 64, 125, 216, 343$.

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

[4] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.

[5] L.-H. Lim and P. Comon, "Nonnegative approximations of nonnegative tensors," *Journal of chemometrics*, vol. 23, no. 7-8, pp. 432–441, 2009.

[6] W. P. Krijnen, T. K. Dijkstra, and A. Stegeman, "On the non-existence of optimal solutions and the occurrence of degeneracy in the candecomp/parafac model," *Psychometrika*, vol. 73, no. 3, pp. 431–439, 2008.

[7] M. Sørensen, L. D. Lathauwer, P. Comon, S. Icart, and L. Deneire, "Canonical polyadic decomposition with a columnwise orthonormal factor matrix," *SIAM Journal on Matrix Analysis and Applications*, vol. 33, no. 4, pp. 1190–1213, 2012.

[8] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *Signal Processing Magazine, IEEE*, vol. 32, no. 2, pp. 145–163, 2015.

[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] L. Sorber, M. Van Barel, and L. De Lathauwer, "Structured data fusion." *IEEE Journal on Selected Topics in Signal Processing*, vol. 9, no. 4, pp. 586–600, 2015.

[11] A. P. Liavas and N. D. Sidiropoulos, "Parallel algorithms for constrained tensor factorization via alternating direction method of multipliers," *IEEE Transactions on Signal Processing*, vol. 63, no. 20, pp. 5450–5463, 2015.

[12] K. Huang, N. D. Sidiropoulos, and A. P. Liavas, "A flexible and efficient framework for constrained matrix and tensor factorization," *IEEE Transactions on Signal Processing*, accepted for publication, May 2016.

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

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

[15] 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.

[16] L. Eldén and H. Park, "A procrustes problem on the stiefel manifold," *Numerische Mathematik*, vol. 82, no. 4, pp. 599–619, 1999.

[17] R. A. Harshman and M. E. Lundy, "Parafac: Parallel factor analysis," *Computational Statistics & Data Analysis*, vol. 18, no. 1, pp. 39–72, 1994.

[18] C. A. Andersson and R. Bro, "The n-way toolbox for matlab," *Chemometrics and intelligent laboratory systems*, vol. 52, no. 1, pp. 1–4, 2000.

[19] A. P. Liavas, G. Kostoulas, G. Lourakis, K. Huang, and N. D. Sidiropoulos, "Nesterov-based alternating optimization for nonnegative tensor factorization: Algorithm and parallel implementation," *IEEE Transactions on Signal Processing*, 2017.

[20] A. Grama, G. Karypis, V. Kumar, and A. Gupta, *Introduction to Parallel Computing (2nd Edition)*. Pearson, 2003.

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