PARALLEL HIGHER-ORDER SVD FOR TAG-RECOMMENDATIONS

Philipp Shah, Christoph Wieser, François Bry
University of Munich, Institute for Informatics
Oettingenstr. 67, Munich, Germany

ABSTRACT

Social Tagging has become a method of choice for enriching data (like pictures) with meta-data which in turn can be used for searching (like retrieving art pictures) or tag recommendations relying on Singular Value Decompositions (SVD) to reduce dimensionality. We observed that social tagging-based search or tag-recommendation is more successful, if several dimensions describing the tagging community (like age, interests, cultural background) are considered. In other words, social tagging-based search calls for higher-order, or tensor-based, SVD instead of standard, matrix-based, SVD. This article reports on a parallelization of higher-order SVD which achieves a significantly better time complexity by comparison to single processor approaches. This time complexity is possible thanks to a careful use of Hestenes' SVD. This article also reports on a first experimental evaluation of the approach for a tag recommendation system.

KEYWORDS

Search, Ranking, parallel HOSVD, social tagging

1. INTRODUCTION

Social tagging is a method of choice for enriching data with meta-data because it makes use of the interpretation skills of a human community thus dispensing with algorithms that in many fields are still far less effective at interpreting data than humans are. If it can be deployed, social tagging outperforms algorithms at recognising faces, interpreting artworks or the subtle marks of social, ethnic or regional origin in language. We applied social tagging for collecting meta-data on artworks so as to build a similarity search for a collection of pictures of paintings, sculptures, buildings, etc. We observed that similarity search is significantly improved if one considers triples (tag, tagger, picture) instead of pairs (tag, picture). For another application in Italian linguistics, we need to consider quadruples or even tuples with more than four dimensions: Speakers characterised by their social background and the city they live in interpret phrases in terms of the social background and city of living they expect from uses of these expressions.

Principal Component Analysis (PCA) [11] is a method of choice for the calculation of a lower-dimensional, noise-reduced approximation of a data matrix based on a Singular Value Decomposition (section 4.2). The “condensed” approximation serves as basis for analysing the similarities between objects. For example taggings can be analysed if represented as matrix with one object per row (feature vector) and the tag names as columns. The number of columns determines the number of dimensions for the corresponding vector space. An entry of the matrix records how often a tag was assigned to an object, e.g., a picture.

Many social tagging systems track the creators of tags as well, which allows a fine-grained representation of taggings for sharper analyses by comparison to the matrix approach described in the latter paragraph with aggregated and, hence, anonymous taggings. For an adequate representation of tags, taggers, and pictures a 3-dimensional matrix, called 3rd-order tensor is needed with taggers as 3rd mode spanning the 3rd matrix dimension and holding one dimension for each tagger. (Each mode of a tensor holds its own dimensions. For example, the 2nd mode of our tensor holds one dimension for each distinct tag name.) Each entry of the 3rd-order tensor records how often a tag was assigned to an object by a tagger. The application in Italian linguistics requires at least four-mode feature vectors describing speakers and their perceptions of phrases.

However, instead of the standard matrix-based SVD for the PCA, one needs a higher-order, or tensor-based, SVD. Note, that noise-reduction for this generalised approach means reducing the dimensions in each mode separately (pruning tuple in algorithm 1), whereas the number of modes remains the same.

Conceptually, higher-order SVD can be reduced to conventional, matrix-based, SVD because a tensor $\mathcal{A}$ of dth-order can be represented by its d unfoldings $A_{(1)}, \dots, A_{(d)}$ that are matrices, and the SVDs of the unfoldings can be composed into the SVD of tensor $\mathcal{A}$. From an algorithmic viewpoint, however, the approach is far from being simple: The difficulty is to achieve an acceptable time complexity. This article describes how this challenge could be met. More precisely, this article reports (1) on a parallelisation of a higher-order SVD as a basis for higher-order Tag-Recommendation [13] which achieves a time complexity of $O(n^{d+1} \cdot \log(\mathrm{n}) / p)$ or even $O(n^{d+1}/p)$ where n denotes data per d dimensions, and p the number of processors. This time complexity is possible thanks to a careful use of Hestenes' SVD [2, 9]; (2) on a first experimental evaluation of the approach for a social tagging-based tag-recommender (section 6).

While parallelising linear methods is often rather straightforward, parallelising higher-order SVD and achieving the time complexity reported about in the present article is not trivial at all. To the best of the authors' knowledge, there has been no former report on parallelising higher-order SVD and achieving efficiency comparable to that reported about below. This article is organised as follows. Section 1 is this introduction. Section 2 is devoted to related work. Section 3 motivates higher-order SVD by describing two search problems for art history on the one hand, and for Italian linguistics on the other hand. Section 4 describes how to parallel higher-order SVD. Section 5 establishes the time complexity of parallel higher-order SVD. Section 6 reports on a first experimental evaluation of the approach with a social tagging-based search engine for art pictures. Section 7 is a conclusion and points to perspectives of future work.

2. RELATED WORK

Higher-order SVD has been considered in few publications. Lathauwer et al. [4] has been one of the first systematic studies of higher-order SVD called there multilinear SVD. The article investigates what properties (such as uniqueness and first-order perturbation effects) and how symmetries in a tensor can be exploited for simplifying its SVD. Like all other work on higher-order SVD, the present article build upon the recognition by Lathauwer et al. that SVD can be generalised to tensors. The present article extends [4] with a tractable parallelisation of higher-order SVD.

Symeonidis et al. [13] present an algorithm for the SVD of 3rd-order tensors, called (HO)SVD, based on the CubeSVD algorithm of Sun et al. [12] (correcting of a little mistake in that article). Symeonidis et al. [13] also explain how LSA [6] can be extended to 3rd-order tensors and, interestingly, for tag recommendation. The paper reports on significantly better tag recommendation with higher-order SVD than with standard SVD. In contrast to the results of the present paper, Symeonidis et al. [13] is limited to 3rd-order tensors and does not address parallelising their 3rd-order tensor SVD. Vasilescu et al. [14] explain how a higher-order SVD, called there N-mode SVD, can be used for the analysis of facial images with multifactor structure such as scene geometry, view point and illumination. Parallelisation is not addressed and the method's complexity is not discussed. Drineas et al. [7, 8] introduce randomised algorithms for SVD and higher-order SVD with aiming at a better runtime than non-randomised methods. We believe that some time complexity bounds given in this paper are too optimistic.

Brent et al. [2] state and explain a parallel algorithm for the SVD of matrices based on Hestenes' Method [9] making use of a variable number of processors. Chuang [3] extends Brent's approach to a cube connected SIMD for a fixed number of processors (that can be lower than the number of columns). We are critical about the optimistic runtime analysis of [2]. Instead we follow the analysis of [3]. Berry et al. [1] report on the state of the art of parallel (matrix) SVD computation (relaying on [2, 3]). In particular the algorithm for parallel basis orthogonalisation ROCVEC using the algorithm ROCDEC correspond to ORTH (algorithm 3) using Parallel Matrix SVD (algorithm 2) in the present paper. The Hestenes' Method is described as an “efficient way to compute the decomposition”.

None of the above mentioned articles report on parallelising higher-order SVD and achieving an efficiency comparable to that the present article reports about. With the exception of Sun et al. [12] and Symeonidis et al. [13] only considering 3rd-order tensors, there are no former reports on experimental evaluations pointing to the superiority of higher-order LSA for social tagging-based similarity search.

3. TAG-RECOMMENDATION

Social tagging has become a method of choice for enriching data (like pictures) with meta-data that, in turn, can be used for search purposes. Indeed, social tagging is a form of human computation, that is, a software-based harnessing of human skills (like face recognition or semantic interpretation) that, so far, algorithms can hardly mimic. Furthermore, social tagging has the potential to unveil the very meta-data a community is interested in and therefore is likely to use in search. For these reasons, and due to a lack of convincing alternatives, we decided to rely on social tagging-based search for the following two applications. Even though the two application fields are fully unrelated, they both greatly benefit from the very same approach to search – and both require the same knowledge discovery techniques

The first application is a search in an online image archive used by art historians, the Prometheus image archive http://www.prometheus-bildarchiv.de/ consisting of about 30.000 images of paintings, sculptures, drawings, and early photographs. So far, very few meta-data, if at all, are available. Using an ESP game or image labeler called ARTigo http://www.artigo.org/, one of the games with a purpose suggested by Luis von Ahn [15], we are collecting tags reflecting the users' perception, and interpretation, of the archive's images. ARTigo is rather successful. Within 4 years, it collected 4.03 million tags 0.855 million (21%) of which have been validated, that is, suggested by at least two players acting independently from each other. Furthermore, ARTigo succeeded in attracting an active community of 14.169 registered users and additional 26.428 unregistered users of which on every day at least 73 play on the platform during visits lasting in average 5.5 minutes. ARTigo users are both players and users of the art work search engine offered on the platform. The second application, still under development, is a Web-based field research in linguistics aiming at tracking the divergence of the languages spoken in large Italian (and Italian speaking Swiss) cities. In contrast to most other developed countries, Italy currently experiences such a divergence. Here, specific games with a purpose are conceived that aim at collecting both, language expressions and Italian speakers' perceptions of their metropolitan and social usage.

Even though tapping the “knowledge of the crowds” is promising an approach for both applications, we soon discovered that exploiting this knowledge requires for both applications the same rather sophisticated knowledge discovery techniques. The reason is mostly that the tags collected reflect semantics whose discovery and understanding is by no means obvious. We discovered for example that artworks primarily tagged with colours belong to abstract art. Clearly, for a social tagging-based search to unleash its potential, the contextual usage of tags must be discovered. While PCA is a convenient instrument for discovering the implicit meaning of tags, its standard two-dimensional realisation is insufficient for our applications. Indeed, the tags associated with art images span a three-dimensional data space: user – tag – image. The data and meta-data collected for the Italian linguistic application even span an at least four dimensional space: user – user's city – expression – expression's city. If more parameters are considered, assuming we succeed in faithfully collecting them, like age, education and/or income, then even more dimensions are relevant. Experiments we performed point to the significant increase in quality of the data analysis when high dimensional spaces are considered. Thus, a thesis of this article is that tagging-based search calls for a higher-order latent semantic analysis, that is, a latent semantic analysis from higher-order tensors instead of matrices (2nd-order tensors).

4. PARALLELISING HIGHER-ORDER SVD

This section reports on the parallelisation of higher-order SVD. Section 4.1 recalls concepts of multi-linear algebra used in the following sections. Section 4.2 reduces higher-order SVD to matrix SVD [4] resulting in a generalised HOSVD algorithm for tensors of more than 3 dimensions. Section 4.3 describes the parallelization of the HOSVD algorithm.

4.1 Multi-Linear Algebra Concepts

Recall that tensors generalise matrices to more than two dimensions: A matrix is a 2nd-order tensor. A dth-order tensor $\mathcal{A}$ is an element of $\mathbb{R}^{I_1 \times \ldots \times I_d}$ with tensor elements $a_{i_1,\ldots,i_d}$ where the I_i are positive integers.Thus,

a dth-order tensor is a multi-dimensional array with d dimensions. Collecting triples (tagger, tag, artwork) yields a 3rd-order tensor. Collecting 4-tuples (speaker, speaker's city, phrase, phrase's city) yields a 4th-order tensor. The tensors considered here are not the tensors, for example stress tensors, of physics and engineering (called in mathematics "tensor fields").

Mode-n vectors (or fibers) of a dth-order tensor are higher-order analogues of matrix rows and columns. They result from fixing every index of the tensor but one: A matrix column is a mode-1 vector, a matrix row is a mode-2 vector. 3rd-order tensors have column, or mode-1, row or mode-2 and tube or mode-3 vectors. A mode-1 vector of a tensor is a column vector within this tensor, a mode-2 vector a row vector within tensor, etc. The vectors of a tensor, whatever their modes, are represented as column vectors.

The n-mode product of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times \ldots \times I_d}$ with a matrix $M \in \mathbb{R}^{J \times I_n}$ is denoted by $\mathcal{A} \times_n M \in \mathbb{R}^{I_1 \times \ldots \times I_{n-1} \times J \times I_{n+1} \times \ldots \times I_d}$, and is defined by: $(\mathcal{A} \times_n M)_{i_1,\ldots,i_{n-1},j,i_{n+1},\ldots,i_d} = \sum_{i_n=1}^{I_n} a_{i_1,\ldots,i_d} \cdot m_{j,i_n}$

An unfolding (or flattening or matricisation) of a tensor $\mathcal{A}$ reorganises the tensor's entries into a matrix M. Folding this matrix M results in the original tensor $\mathcal{A}$. Tensor unfoldings are used to reduce tensor computations to matrix computations. The higher-order SVD of a tensor can be reduced to computing the SVD of each unfolding of that tensor [4]. The mode-n unfolding of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times \ldots \times I_d}$ is denoted by $A_{(n)} \in \mathbb{R}^{I_n \times (I_{n+1} \ldots I_d I_1 \ldots I_{n-1})}$. It re-arranges the mode-n vectors of A as the columns of a matrix, or as expressed in [4] "all the column (row, …) vectors are stacked one after the other". Formally, the tensor element $a_{i_1,\ldots,i_d}$ yields the matrix element $m_{i_n,j}$, where $j = 1 + \sum_{k=1;k\neq 1}^{d} (i_k - 1) \cdot J_k$ with $J_k = \prod_{h=1;h\neq n}^{k-1} I_h$.

In the case of a 3rd-order tensor $\mathcal{A} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$ there exist three different unfoldings:

$A_{(1)} \in \mathbb{R}^{J_1 \times J_2 J_3}$, $A_{(2)} \in \mathbb{R}^{J_2 \times J_1 J_3}$, $A_{(3)} \in \mathbb{R}^{J_3 \times J_1 J_2}$

The scalar product of the tensors $\mathcal{A}$ and $\mathcal{B}$ is $\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i_1} \ldots \sum_{i_d} a_{i_1,\ldots,i_d} \cdot b_{i_1,\ldots,i_d}$. $\mathcal{A}$ and $\mathcal{B}$ are orthogonal, if their scalar product is 0. The norm of a tensor is $\|\mathcal{A}\| = \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle}$.

The sub-tensors of a tensor $\mathcal{A}$ are denoted by $\mathcal{A}_{i_{n=a}}$ for $n = 1, \ldots, N$ where the nth index of $\mathcal{A}$ is fixed to a. All-orthogonal for a tensor means, that every pair of different sub-tensors is orthogonal. Ordered for a tensor means $\left\|\mathcal{A}_{i_n=1}\right\| \geq \cdots \geq \left\|\mathcal{A}_{i_n=I_n}\right\| \geq 0$ for all $n = 1, \ldots, N$.

[10] is a recent overview of tensor decomposition algorithms and software. The field is evolving fast since tensor decompositions are used more and more in many application areas such as psychometrics, chemometrics, computer vision, graph analysis, and neurosciences.

4.2 SVD and Higher-Order SVD

Algorithm 1: serial HOSVD, parallel HOSVD uses parallel for-loops and parallel SVD (Algorithm 2).

Input	dth-order tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times \ldots \times I_d}$, pruning tuple $(m_1, \ldots, m_d) \in [1, I_1] \times \ldots \times [1, I_d]$
Output	dth-order tensor $\mathcal{A}' \in \mathbb{R}^{I_1 \times \ldots \times I_d}$ as the best rank-$(m_1, \ldots, m_d)$ approximation to $\mathcal{A}$
Unfolding	**for** $i = 1, \ldots, d$ **Compute** the unfolding $A_{(i)}$ of $\mathcal{A}$ **end**
Matrix SVD	**for** $i = 1, \ldots, d$ **Compute** the SVD $A_{(i)} = U_i \Sigma_i V_i^{\mathsf{T}}$ **end**
Pruning	**for** $i = 1, \ldots, d$ $W_i := \left[u_{i,1}, \ldots, u_{i,m_i}\right]$ where $u_{i,1}$ column vectors of U_i **end**
Core Tensor	**Compute** $\mathcal{S} := \mathcal{A} \times_1 \mathcal{W}_1^{\mathsf{T}} \times_2 \mathcal{W}_2^{\mathsf{T}} \ldots \times_d \mathcal{W}_d^{\mathsf{T}}$
Approximation	**Compute** $\mathcal{A}' := \mathcal{S} \times_1 \mathcal{W}_1 \times_2 \mathcal{W}_2 \ldots \times_d \mathcal{W}_d$

A matrix $A \in \mathbb{R}^{I_1 \times I_2}$ can be decomposed as $A = U\Sigma \mathrm{V}^{\mathsf{T}}$ where: Σ is a (pseudo) diagonal matrix (In a pseudo diagonal matrix nonzero elements can only occur in the diagonal of a (left) upper sub-matrix.) $\mathrm{diag}(\sigma_1, \ldots, \sigma_z) \in \mathbb{R}^{I_1 \times I_2}$ with $z = \min\{I_1, I_2\}$ and ordered $(\sigma_1 \geq \cdots \geq \sigma_z \geq 0)$. $U \in \mathbb{R}^{I_1 \times I_1}$ and $\mathrm{V} \in \mathbb{R}^{I_2 \times I_2}$ are orthogonal matrices. The σ_i are the singular values of A, the columns u_i of $U = \left[u_1, \ldots, u_{I_1}\right]$ (v_j of $\mathrm{V} = \left[v_1, \ldots, v_{I_2}\right]$ resp.) are the left (right, resp.) singular vectors of A.

A tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times \ldots \times I_d}$ can be decomposed [4] as $\mathcal{A} \coloneqq \mathcal{S} \times_1 U_1 \times_2 U_2 \ldots \times_d U_d$ where: The core tensor $\mathcal{S} \in \mathbb{R}^{I_1 \times \ldots \times I_d}$ is ordered and all-orthogonal. Every matrix $U_n = [u_{n,1}, \ldots, u_{n,I_n}] \in \mathbb{R}^{I_n \times I_n}$ (with $u_{n,i} \in \mathbb{R}^{I_n \times 1}$, $1 \leq n \leq d$) is orthogonal. $\sigma_{n,1} = \left\| \mathcal{S}_{i_n=1} \right\|$ are the n-mode singular values; the vectors $u_{n,i} \in \mathbb{R}^{I_n \times 1}$ are n-mode singular vectors of $\mathcal{A}$.

The n-mode singular vectors of a tensor generalise the left and right singular vectors of a matrix, the value of the Frobenius-norm $\|.\|_F = \sqrt{\sum_{i,j} |a_{i,j}|^2}$ of the $(d-1)$th-order sub-tensors of the core tensor $\mathcal{S}$ correspond to the singular values of a matrix.

The n-rank of $\mathcal{A}$, $\mathrm{rank_n}(\mathcal{A})$, is the dimension of the vector space spanned by the n-mode vectors.

Every tensor has a higher-order Singular Value Decomposition and its n-mode singular values are uniquely defined [4]. The following theorem [4] relates (matrix) SVD and (tensor) higher-order SVD: If $\mathcal{A} \coloneqq \mathcal{S} \times_1 U_1 \times_2 U_2 \ldots \times_d U_d$ is a higher-order SVD, then the SVD of the matrix unfolding $A_{(n)}$ is $U_n \Sigma_n \mathrm{V}_n^{\mathsf{T}}$ where $\Sigma_n = \mathrm{diag}(\sigma_{n,1}, \ldots, \sigma_{n,I_n}) \in \mathbb{R}^{I_n \times I_n}$.

It follows from this theorem that a higher-order SVD of a dth-order tensor can be computed from the SVD of its mode-n unfoldings $A_{(n)} = U_n \Sigma_n \mathrm{V}_n^{\mathsf{T}} = \Sigma_n \times_1 U_n \times_2 V_n$ for $n = 1, \ldots, d$ as follows: $\mathcal{A} \coloneqq \mathcal{S} \times_1 U_1 \times_2 U_2 \ldots \times_d U_d$ where U_n contains the nth left singular vectors of unfolding $A_{(n)}$. These equations result in the following algorithm which generalises the algorithm given in [12] and [13] for 3rd-order tensors to tensors of any order:

Note that, in principle, any SVD algorithm can be used at line 5. The parameters m_i aim at a reduction of the dimensionality of U_i (i$= 1, \ldots, d$) as follows: At line 8 the length of the column vectors of u_i is pruned according to m_i, that is, keeping the first $m_i \leq I_i$ rows of u_i for W_i, only. W_i (instead of full U_i) is used for the computation of $\mathcal{A}'$ in lines 10 and 11.

4.3 Parallel Higher-Order SVD

The parallelisation of the higher-order SVD algorithm of section 4.2 consists on one hand of a straightforward parallelisation of independent computations, and on a parallelisation of the matrix SVD on basis of the parallel SVD algorithm using the Hestenes' Method. The following steps of computing HOSVD (algorithm 1) are the subject of our parallelisation intent:

Unfolding: Computing the unfoldings $A_{(i)}$ is independent from each other they can be performed in parallel.

Matrix SVD: The SVDs of the unfoldings $A_{(i)}$ are independent from each other. They can be performed inparallel. The results of this step is necessary for the computation of the following steps. Below in this article the computation of an SVD itself will be subject to our parallelisation intent.

Pruning: The pruning of the (independent) column vectors of $U_{(i)}$ can be performed in parallel.

Core Tensor: The result of an n-mode product is a tensor. An entry of the resulting tensor does not depend on the other entries. Thus this step can be performed in parallel.

Approximation: As the approximation consists of n-mode products, it can be performed in parallel.

Obviously, the mentioned steps describe a straightforward parallelisation of the HOSVD algorithm. However, a parallel computation of the SVDs itself is needed for achieving the time complexity result established in section 5. The following Parallel Matrix SVD algorithm [2, 3] is based on the Hestenes' Method.

Parallel Matrix SVD (algorithm 2) computes the SVD of a matrix $A = U\Sigma V^{\mathsf{T}}$ using the Hestenes' Method [2, 9] as follows: An orthogonal matrix V is constructed such that the matrix $W = AV$ has orthogonal columns in several orthogonalisation steps. One step orthogonalises two columns with plane rotations (ORTH algorithm). These steps are performed in parallel (algorithm 2). Finally for the SVD, the matrix U is obtained by normalising the length of each nonzero column of the matrices $W = [w_1, \ldots, w_n]$ and with $W = U\Sigma$ we get $\Sigma = \mathrm{diag}(\|\mathrm{w}_1\|, \ldots, \|\mathrm{w}_\mathrm{n}\|)$. Thus $W = AV$ implies the SVD of $A = U\Sigma \mathrm{V}^{\mathsf{T}}$.

Parallel Matrix SVD (algorithm 2) uses the following communication scheme between the processors $P_1, \ldots, P_p$, where $n/2 = p$ is the total number of processors p in this scheme: The processors are arranged in logical groups $g_i$with $i = 1, \ldots, d$. Two adjacent processors P_k and P_{k+1} can communicate. Each processor P_k has registers R_k and L_k with output and input lines $outR_k$, $outL_k$ (and inR_k, inL_k, resp.). The output line $outR_k$ is connected to the input line inL_{k+1}. The scheme for 3 processors is described in figure 1.

ORTH (algorithm 3) is used in Parallel Matrix SVD (algorithm 2) aiming at approximating the orthogonalisation of two columns i and j of a matrix A_k up to a precision ε. ORTH yields the matrix A_{k+1} operating on a single processor. ORTH (algorithm 3) is the ORTH algorithm of Chuang [3] with a slight extension: The matrix V_k (of the SVD $A_k = U_k \Sigma_k V_k^{\mathsf{T}}$) is additionally computed. Computing the matrix V in ORTH is cheaper in terms of time complexity than reconstructing V out of the matrices W and A with $W = VA$.

Algorithm 2: Parallel Matrix SVD.

Input matrix $A \in \mathbb{R}^{m \times n}$, precission $\varepsilon \geq 0$
Output matrix W$\in \mathbb{R}^{m \times n}$, matrix V$\in \mathbb{R}^{n \times n}$
Assume $n/2$ processors are working in parallel

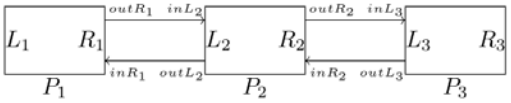
$A_1 \coloneqq A$, $V_1 \coloneqq I$, z:=0
for $n = 1, \dots, s$
  $L_k \coloneqq 2k - 1$ with $k \leq n/2$
  $R_k \coloneqq 2$ with $k \leq n/2$
  for i$= 1, \dots, n-1$
    $z \coloneqq z + 1$
    if $L_k < R_k$
      then ORTH$(A_z, V_z, \varepsilon, L_k, R_k)$
      else ORTH$(A_z, V_z, \varepsilon, R_k, L_k,)$
    if $k = 1$ **then** $outR_k \coloneqq R_k$
      else if $k < n/2$
        then $outR_k \coloneqq L_k$
    if $k > 1$
      then $outL_k \coloneqq R_k$
    {wait for outputs to propagate to
     inputs of adjacent processors}
    if $k < n/2$
      then $R_k \coloneqq inR_k$
      else $R_k \coloneqq L_k$
    if $k > 1$
      then $L_k \coloneqq inL_k$
  end
end
$W \coloneqq A_{z+1}$, $V \coloneqq V_{z+1}$

Algorithm 3: ORTH.

Input matrix $A \in \mathbb{R}^{m \times n}$, matrix $V_k \in \mathbb{R}^{n \times n}$
precission $\varepsilon \geq 0$
column indices $i, j \in \mathbb{N}$, $i, j \leq n$
Output matrix $A_{k+1} \in \mathbb{R}^{m \times n}$, matrix $V_{k+1} \in \mathbb{R}^{n \times n}$

$A_{k+1} \coloneqq A_k$
$V_{k+1} \coloneqq V_k$
$\gamma \coloneqq a_{k,i}^{\mathsf{T}} a_{k,j}$
  if $|\gamma| > \varepsilon$ **then**
    $\alpha \coloneqq \left\| a_{k,i} \right\|^2$
    $\beta \coloneqq \left\| a_{k,j} \right\|^2$
    $\xi \coloneqq \frac{\beta - \alpha}{2\gamma}$
    $t \coloneqq \frac{\operatorname{sign}(\xi)}{|\xi| + \sqrt{1+\xi^2}}$
    $\cos(\theta) \coloneqq \frac{1}{\sqrt{1+t^2}}$
    $\sin(\theta) \coloneqq t \cdot \cos(\theta)$
    $\left[a_{k+1,i}, a_{k+1,j}\right] \coloneqq \left[a_{k,i}, a_{k,j}\right] \cdot \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$
    $\left[v_{k+1,i}, v_{k+1,j}\right] \coloneqq \left[v_{k,i}, v_{k,j}\right] \cdot \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$
end

Figure 1: Communication Scheme for 3 processors.



5. TIME COMPLEXITY

Algorithm HOSVD (section 4.2) has time complexity $O\left(I \cdot \prod_{i=1}^{d} I_i\right)$, $I \coloneqq \max\{I_1, \dots, I_d\}$. This section gives a proof for the time complexity of the parallel higher-order LSA algorithm given in section 4.3, which is $O(n^{d+1} \cdot \log(n)/p)$ or even $O(n^{d+1}/p)$ where n denotes data per d dimensions, and p the number of processors.

5.1 Time Complexity of higher-order SVD (Algorithm 1, serial version)

The costs of the unfoldings $A_{(1)}, \dots, A_{(d)}$ are linear in the number of entries in the tensor.

Note: $A = U\Sigma V^{\mathsf{T}} \Leftrightarrow A^{\mathsf{T}} = V\Sigma^{\mathsf{T}}U^{\mathsf{T}}$ and the time complexity of computing the SVD of a matrix $A \in \mathbb{R}^{m \times n}$ is $O(mn^2)$, i.e. the time complexity of the SVD of the transposed matrix A^{T} is $O(m^2 n)$ and transposing costs at most $O(mn)$.

The computational costs of the SVD of $A_{(i)} \in \mathbb{R}^{I_i \times (I_{i+1} \cdot \dots \cdot I_d I_1 \cdot \dots \cdot I_{i-1})}$ can be minimised by computing the SVD of ${A_{(i)}}^{\mathsf{T}} \in \mathbb{R}^{(I_{i+1} \cdot \dots \cdot I_d I_1 \cdot \dots \cdot I_{i-1}) \times I_i}$ which is $O(I_i \cdot I_1 \cdot \dots \cdot I_d) = O\left(I_i \cdot \prod_{j=1}^{d} I_j\right)$, thus line 4–6 cost $\sum_{i=1}^{d} O\left(I_i \cdot \prod_{j=1}^{d} I_j\right) = O\left(\sum_{i=1}^{d} I_i \cdot \prod_{j=1}^{d} I_j\right)$.

Figure 2: Time Complexity of Serial higher-order SVD (Algorithm 1, serial version).

lines 1-3	d times
lines 2	$O(I_1 \cdot \ldots \cdot I_d)$
lines 4-6	$O\left(\sum_{i=1}^{d}\left(I_i \cdot \prod_{j=1}^{d} I_j\right)\right) = O\left(I \cdot \prod_{j=1}^{d} I_j\right)$
lines 7-9	$o\left(d \cdot \prod_{j=1}^{d} I_j\right) = o\left(\prod_{j=1}^{d} I_j\right)$
lines 10/11	$O\left(I \cdot \prod_{i=1}^{d} I_j\right)$

If $I \coloneqq \max\{I_1, \ldots, I_d\}$, then $O\left(\sum_{i=1}^{d}\left(I_i \cdot \prod_{j=1}^{d} I_j\right)\right) = O\left(d \cdot I \cdot \prod_{j=1}^{d} I_j\right) = O\left(I \cdot \prod_{j=1}^{d} I_j\right)$, as d is fixed.

Also line 7–9 can be implemented efficiently because pruning the last $I_i - m_i$ columns of U_i can be done in constant time. The slowest method would be to read every single entry and to store only the needed elements into a new matrix. This would cost $O\left(\prod_{j=1}^{d} I_j\right)$ for each matrix, thus $O\left(\prod_{j=1}^{d} I_j\right)$ is an upper bound. However, it would also be possible just to ignore the last columns of U_i which results in $O(1)$; but as $1 \ll I \cdot \prod_{j=1}^{d} I_j$ and $\prod_{j=1}^{d} I \ll I \cdot \prod_{j=1}^{d} I_j$ this won't change the total time complexity.

The n-mode product ($1 \leq d \leq n$) of a tensor $\mathcal{F} \in \mathbb{R}^{I_1 \times \ldots \times I_d}$ and a matrix $M \in \mathbb{R}^{J_n \times I_n}$ has a time complexity of: $O\left(\prod_{j=1}^{d} I_i \cdot J_n\right)$, e.g. for $n = 2$ the product has $\prod_{i=1, i\neq 2}^{d} I_i \cdot J_2 = I_1 \cdot J_2 \cdot I_3 \cdot \ldots \cdot I_d$ entries and to compute one entry we have to sum up I_2 numbers.

Remember: $(\mathcal{F} \times_n M)_{i_1 \ldots i_{n-1} j_n i_{n+1} \ldots i_d} = \sum_{i_n \leq I_n} f_{i_1 \ldots i_d} m_{j_n i_n}$. Thus

$\mathcal{A} \times_1 W_1$	needs $O\left(\prod_{i=1}^{d} I_i \cdot m_1\right) = O\left(\prod_{i=1}^{d} I_i \cdot I_1\right)$,
$(\mathcal{A} \times_1 W_1^{\intercal}) \times_2 W_2$	needs $O\left(\prod_{i=1}^{d} I_i \cdot m_1 + \prod_{i=2}^{d} I_i \cdot m_1 \cdot m_2\right)$, and finally
$\left(\ldots\left((\mathcal{A} \times_1 W_1^{\intercal}) \times_2 \ldots\right) \ldots \times_d W_d^{\intercal}\right)$	needs $O\left(\prod_{i=1}^{d} I_i \cdot m_1 + \cdots + I_d \cdot \prod_{i=1}^{d} m_i\right) = O\left(\prod_{i=1}^{d} I_i \cdot I\right)$ with $I \coloneqq \max\{I_1, \ldots, I_d\}$ $(m_i \leq I_i)$.

As $m_1, \ldots, m_d$ have switched roles with $I_1, \ldots, I_d$ in line 11, the time complexity will be the same. (line 10: $\mathcal{A} \in \mathbb{R}^{I_1 \times \ldots \times I_d}, W_1^{\intercal} \in \mathbb{R}^{m_1 \times I_1}, \ldots, W_d^{\intercal} \in \mathbb{R}^{m_d \times I_d}$, line 11: $\mathcal{S} \in \mathbb{R}^{m_1 \times \ldots \times m_d}, W_1 \in \mathbb{R}^{I_1 \times m_1}, \ldots, W_d \in \mathbb{R}^{I_d \times m_d}$)

The total time complexity of algorithm 1 in its serial version is

$$O\left(d \cdot \prod_{i=1}^{d} I_i + I \cdot \prod_{i=1}^{d} I_i + \prod_{i=1}^{d} I_i + 2I \cdot \prod_{i=1}^{d} I_i\right) = O\left((d + 3I + 1) \cdot \prod_{i=1}^{d} I_i\right) = O\left(I \cdot \prod_{i=1}^{d} I_i\right).$$

5.2 Time Complexity of Parallel SVD (Algorithm 2)

With p processors and n columns, the parallel SVD of a $m \times n$ matrix A has a time complexity of $O(mn^2 \cdot s/p)$ [3], if s is the number of needed sweeps (according to [2, 3] s could be $O(\log(n))$). However, as mentioned in [2] s can be chosen as a constant between 6 and 10 for practical purposes.

5.3 Time Complexity of Parallel higher-order SVD

Figure 3: Time Complexity of Parallel higher-order SVD (Algorithm 1, parallel version)

lines 1-3	$d \cdot O\left(1 \cdot \frac{I_1 \cdot \ldots \cdot I_d}{p_d}\right) = O\left(\frac{\prod_{i=1}^{d} I_i}{p}\right)$
lines 4-6	$d \cdot O\left(\frac{I \cdot \prod_{i=1}^{d} I_i}{p} s\right)$
lines 7-9	$o\left(\frac{d}{p} \cdot \prod_{j=1}^{d} I_j\right) = o\left(\frac{\prod_{j=1}^{d} I_j}{p}\right)$
lines 10/11	$2 \cdot O\left(\frac{I \cdot \prod_{i=1}^{d} I_i}{p}\right)$

The calculation of the d unfoldings $A_{(i)}$ could be implemented efficiently, like in the normal higher-order SVD algorithms. But this time we have p processors for our computation, which could work independently. The computational costs of the SVD of $A_{(i)} \in \mathbb{R}^{I_i \times (I_{i+1} \cdot \ldots \cdot I_d I_1 \cdot \ldots \cdot I_{i-1})}$ using the parallel SVD is $O\left(\frac{s_i}{p_i} \cdot I_i \cdot \left(\prod_{j=1; j\neq i}^{d} I_j\right)^2\right)$. (Remember that the time complexity of the parallel SVD using Hestenes' method is $O\left(s\frac{mn^2}{p}\right)$ if we have p processors and a $m \times n$ matrix.)

This time could be optimised by computing the SVD of the transposed ($A = U\Sigma V^{\intercal}$ iff $A^{\intercal} = V\Sigma^{\intercal}U^{\intercal}$) unfolding $A_{(i)}^{\intercal} \in \mathbb{R}^{(I_{i+1} \cdot \ldots \cdot I_d I_1 \cdot \ldots \cdot I_{i-1}) \times I_i}$ (if $I_{i+1} \cdot \ldots \cdot I_d I_1 \cdot \ldots \cdot I_{i-1} > I_i$), which has a time of

$O\left(\frac{s_i}{p_i} \cdot I_i \cdot \left(\prod_{j=1}^{d} I_j\right)\right)$. If $s := \max\{s_i : 1 \leq i \leq d\}$, then the maximal time complexity of all parallel SVDs together will be $d \cdot O\left(s \frac{I \cdot \prod_{i=1}^{d} I_i}{p}\right)$. Again, s is constant or logarithmically depends on products of the variables of $I_1, \ldots, I_d$.

Line 7–9 could be implemented efficiently, as well. As only the last $m_1, \ldots, m_d$ columns of U_i are pruned.

One processor needs $O\left(I \cdot \prod_{i=1}^{d} I_i\right)$ for line 10 and 11 as described in the paragraph about the serial higher-order SVD algorithm (Algorithm 1). And because we have p processors and each processor could compute one entry of a x-order tensor product we obtain $O\left(I \cdot \frac{1}{p} \cdot \prod_{j=1}^{d} I_j\right)$.

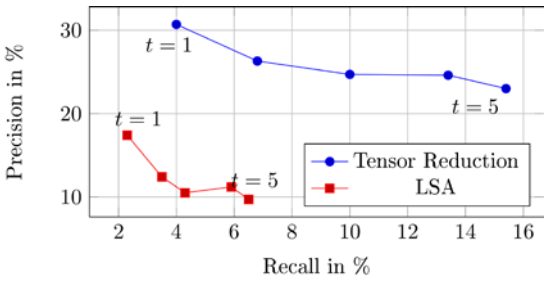
To sum it up: the parallel higher-order SVD algorithm has a time complexity of $O\left(s \cdot I \cdot \frac{1}{p} \cdot \prod_{j=1}^{d} I_j\right)$. In other words, using the parallel higher-order SVD algorithm is p-times faster than a single-threaded higher-order SVD algorithm, if p is the number of processors on the same system. Hence, a parallel higher-order SVD of a dth-order tensor $\mathcal{A} \in \mathbb{R}^{n \times \ldots \times n}$ has a time complexity of $O(n^{d+1} \cdot s/p)$. As mentioned above s, is either a constant or depends logarithmically on n.

6. EVALUATION

The correctness of the parallel HOSVD algorithm proposed in this paper and in particular the correctness of Hestenes' parallel SVD were tested using the tag-recommendation approach proposed in [13]. The explored data set was gathered by a Game with a Purpose [15] called ARTigo mentioned in section 3.

Our tag-recommendation results (see figure 4) based on the ARTigo data set confirm the results published in [13] qualitatively. Search strings consisting of 1–5 tags t yielded better average precision results from 12.2% (matrix reduction) to 25.86% (tensor reduction) and better average recall results from 4.5% (matrix reduction) to 9.9% (tensor reduction).

Figure 4: Social tagging-based search (t query tags).



7. CONCLUSION AND PERSPECTIVES

In this article we presented an approach of a generalised parallel Higher-Order Singular Value Decomposition algorithm. To the best of our knowledge, this is the first proposed parallel HOSVD approach. We extended the serial CubeSVD on 3rd-order tensors to a HOSVD on dth-order tensors, combined it with the parallel Hestenes' SVD on matrices and optimised it. The careful analysis of this new parallel algorithm for Higher-Order Singular Value Decomposition yields a better runtime complexity than the standard Higher-Order Singular Value Decomposition algorithms. The better runtime complexity mainly results from engaging many processors in parallel. The runtime complexity of the parallel Higher-Order Singular Value Decomposition is the runtime complexity of a standard Higher-Order Singular Value Decomposition divided by the number of processors. Thus it is a first optimal parallelisation for a Higher-Order Singular Value Decomposition. This is important because standard or non-parallel Higher-Order Singular Value Decompositions suffer from high runtime complexities. The correctness of the approach was qualitatively proven using an established approach for higher-order tag-recommendation. The achieved runtime results based on a prototypical implementation in Java and limited to only 8 processors are just about ten times slower then efficient serial implementations.

We plan to improve the implementation of the proposed parallel HOSVD approach using sparse matrix storage and a low-level implementation. A larger number of processors is targeted the MapReduce [5] approach.

REFERENCES

Journal

[1] M. W. Berry et al, July 2005. Parallel Algorithms for the Singular Value Decomposition. In Handbook on Parallel Computing and Statistics.

[4] L. De Lathauwer et al. A Multilinear Singular Value Decomposition, 2000. SIAM J. Matrix Anal. Appl, 21:1253–1278.

[6] S. C. Deerwester et al, 1990. Indexing by Latent Semantic Analysis. JASIS, 41(6):391–407.

[9] M. R. Hestenes, 1958. Inversion of Matrices by Biorthogonalization and Related Results. Journal of the Society for Industrial and Applied Mathematics, 6(1).

[10] T. G. Kolda and B. W. Bader, 2009. Tensor Decompositions and Applications. SIAM Review, 51(3).

[11] K. Pearson, 1901. On lines and planes of closest fit to systems of points in space. Philosophical Magazine, 2(6):559–572.

[15] L. von Ahn. Games With a Purpose, 2006. Computer, 29(6):92–94.

Conference paper or contributed volume

[2] R. Brent and F. Luk, 1985. The Solution of Singular-Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays. SIAM Journal on Scientific and Statistical Computing, 6:69–84.

[3] H. Y. J. Chuang and L. Chen, 1989. Efficient computation of the singular value decomposition on cube connected SIMD machine. In Supercomputing '89, pages 276–282, New York, NY, USA, ACM.

[5] J. Dean and S. Ghemawat, 2004. MapReduce: Simplified Data Processing on Large Clusters. In OSDI.

[7] P. Drineas et al, 2001. An Experimental Evaluation of a Monte-Carlo Algorithm for Singular Value Decomposition. In Panhellenic Conference on Informatics, pages 279–296.

[8] P. Drineas and M. W. Mahoney, 2005. A Randomized Algorithm for a Tensor-Based Generalization of the Singular Value Decomposition. Technical report, Yale University.

[12] J.-T. Sun et al, 2005. CubeSVD: a novel approach to personalized Web search. In WWW '05, pages 382–390, New York, NY, USA, ACM Press.

[13] P. Symeonidis, et al, 2008. Tag Recommendations Based on Tensor Dimensionality Reduction. In RecSys '08, New York, NY, USA, ACM.

[14] M. A. O. Vasilescu and D. Terzopoulos. Multilinear Image Analysis for Facial Recognition, 2002. In ICPR (2), pages 511–514.