SIAM J. SCI. COMPUT.
Vol. 43, No. 4, pp. C290–C311

COMPARISON OF ACCURACY AND SCALABILITY OF GAUSS–NEWTON AND ALTERNATING LEAST SQUARES FOR CANDECOMC/PARAFAC DECOMPOSITION*

NAVJOT SINGH†, LINJIAN MA‡, HONGRU YANG‡, AND EDGAR SOLOMONIK‡

Abstract. Alternating least squares is the most widely used algorithm for CANDECOMC/PARAFAC (CP) tensor decomposition. However, alternating least squares may exhibit slow or no convergence, especially when high accuracy is required. An alternative approach is to regard CP decomposition as a nonlinear least squares problem and employ Newton-like methods. Direct solution of linear systems involving an approximated Hessian is generally expensive. However, recent advancements have shown that use of an implicit representation of the linear system makes these methods competitive with alternating least squares (ALS). We provide the first parallel implementation of a Gauss–Newton method for CP decomposition, which iteratively solves linear least squares problems at each Gauss–Newton step. In particular, we leverage a formulation that employs tensor contractions for implicit matrix-vector products within the conjugate gradient method. The use of tensor contractions enables us to employ the Cyclops library for distributed-memory tensor computations to parallelize the Gauss–Newton approach with a high-level Python implementation. In addition, we propose a regularization scheme for the Gauss–Newton method to improve convergence properties without any additional cost. We study the convergence of variants of the Gauss–Newton method relative to ALS for finding exact CP decompositions as well as approximate decompositions of real-world tensors. We evaluate the performance of sequential and parallel versions of both approaches, and study the parallel scalability on the Stampede2 supercomputer.

Key words. tensor decomposition, CP decomposition, Gauss–Newton method, alternating least squares, Cyclops tensor framework

AMS subject classifications. 15A69, 15A72, 65K10, 65Y20, 65Y04, 65Y05, 68W25

DOI. 10.1137/20M1344561

1. Introduction.

The CP (canonical polyadic or CANDECOMC/PARAFAC) tensor decomposition is widely used for data analytics in different scientific fields [14, 18, 30, 34, 44], machine learning applications [2, 5, 26], and quantum chemistry [51]. CP decomposition of an input tensor can be computed via different optimization techniques, such as variants of gradient descent [1, 37], deflations [2, 3], and alternating least squares [26].

Nowadays, the alternating least squares (ALS) method, which solves quadratic optimization subproblems for each factor matrix in an alternating manner, is most commonly used and has become a target for parallelization [17, 21], performance optimization [29, 43], and acceleration by randomization [9]. A major advantage of ALS is its guaranteed monotonic decrease of the residual. However, there are many cases where ALS shows slow or no convergence when a solution with high

*Submitted to the journal's Software and High-Performance Computing section June 11, 2020; accepted for publication (in revised form) March 22, 2021; published electronically August 4, 2021.
https://doi.org/10.1137/20M1344561

Funding: The work of the first, second, and fourth authors was supported by the US NSF OAC SSI program, award 1931258. This work was supported by National Science Foundation grants ACI-1548562, OCI-0725070, and ACI-1238993, and by the Texas Advanced Computing Center (TACC) through allocation TG-CCR180006.

†Department of Mathematics, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (navjot2@illinois.edu).

‡Department of Computer Science, University of Illinois at Urbana Champaign, Urbana, IL 61801 USA (lma16@illinois.edu, hyang87@illinois.edu, solomonik@cs.illinois.edu).

resolution is required, which is also called the "swamp" phenomenon [31]. Swamps deteriorate both the running time and the convergence behavior of the ALS method. Consequently, researchers have been looking at different alternatives to ALS, including various regularization techniques [28, 35], line search [32, 36, 42], and gradient-based methods [1, 37, 41, 48, 53, 55].

Of the variants of gradient-based methods, one promising approach is to perform the CP decomposition by solving a nonlinear least squares problem using the Newton or Gauss–Newton methods [37, 54, 55]. These methods offer superlinear convergence and are better at avoiding the swamps inhibiting performance of ALS. Naive solution of linear equations arising in these methods is expensive to compute. For rank-R decomposition of an order-N tensor with all the dimension sizes equal to s, standard algorithms either perform Cholesky on the normal equations [37] or QR on the Jacobian matrix [55], yielding a complexity of $O(N^3s^3R^3)$. However, the matrices involved in this linear system are sparse and have much implicit structure. A recent advancement has shown that the cost of inverting the Hessian can be reduced to $O(N^3R^6)$ [41]. A successive study showed that the cost can be further reduced to $O(NR^6)$, albeit the approach can suffer from numerical instability [53].

Another approach for performing Gauss–Newton with low cost is to leverage an implicit conjugate gradient (CG) method [48]. The structure of the approximated Hessian can be leveraged to perform fast matrix-vector multiplications for CG iterations with a cost of $O(N^2sR^2)$ per iteration. This approach that can also be augmented with preconditioning to accelerate CG convergence rate [48]. In comparison to the aforementioned direct methods, this iterative method is substantially more scalable with respect to the CP rank R. This advantage is critical in many applications of CP decomposition, as in many cases $R \geq s$ is needed (in general CP rank can be as high as s^{N-1} for an order N tensor). Moreover, to solve the problem of CP decomposition with rank $R < s$, Tucker decomposition (or simply HoSVD) [57] can be used to effectively compress the input tensor from dimensions of size s to a Tucker rank r ($r < s$), and CP decomposition of rank R can be efficiently performed on the "smaller" core tensor. This technique in effect leverages the CANDELINC decomposition, which imposes linear constraints on the factor matrices [13]. The authors in [13] prove that this acceleration finds an exact solution if both the Tucker and CP decomposition used above are exact.

In this paper, we investigate the behavior of Gauss–Newton optimization with preconditioned CG on CP decomposition in high rank scenarios (with $R \geq s$ or more generally when the rank is at least the smallest dimension size of the input tensor). We consider various approaches to regularization for Gauss–Newton with implicit CG and ALS. To understand their efficacy, we quantify their ability to converge to exact CP decompositions of synthetic tensors of various CP ranks, as well as to approximate tensors arising in applications in quantum chemistry. With the best regularization strategy, we find that Gauss–Newton is able to consistently find exact CP decompositions for problems where ALS generally does not converge. Further, the Gauss–Newton method obtains lower residuals in approximation. We present these results in section 5.

Our main contribution is the parallel implementation of Gauss–Newton with implicit CG, via a tensor-contraction-based formulation of the method. We develop a distributed-memory implementation of the method using the Cyclops library for parallel tensor algebra. Our implementation supports both NumPy and Cyclops as backends, enabling both sequential and parallel experimental studies. We detail our implementations in section 4. We evaluate the strong and weak scalability of the

method on the Stampede2 supercomputer, and compare its performance to ALS for a variety of test problems. Our results demonstrate that the Gauss–Newton method can converge faster both in sequential and parallel settings. These results are presented in section 5.

This paper makes the following contributions.

- We cast the large matrix-vector multiplication into several tensor contractions so that an existing library on parallel tensor contractions can be utilized. Our analysis achieves the same computational cost as previous work [48].
- We propose and evaluate a new regularization strategy, and demonstrate that it is well suited for CP decomposition with Gauss–Newton.
- We provide the first parallel implementation of Gauss–Newton for CP decomposition.
- We demonstrate that an implementation of parallel Gauss–Newton with preconditioned CG can both converge faster and achieve higher convergence probability for CP decompositions of both synthetic and application-based tensors with high CP rank.

2. Background.

We introduce the notation and definitions used in the forthcoming sections here along with a brief introduction to the ALS algorithm. We suggest [7, 24, 26, 29, 58] for a detailed review of the algorithm and its high performance formulation.

2.1. Notation and definitions.

We use tensor algebra notation in both elementwise form and specialized form for tensor operations [26]. For vectors, bold lowercase Roman letters are used, e.g., $\boldsymbol{x}$. For matrices, bold uppercase Roman letters are used, e.g., $\boldsymbol{X}$. For tensors, bold calligraphic fonts are used, e.g., $\boldsymbol{\mathcal{X}}$. An order N tensor corresponds to an N-dimensional array with dimensions $s_1 \times \cdots \times s_N$. Elements of vectors, matrices, and tensors are denoted in subscript, e.g., x_i for a vector $\boldsymbol{x}$, x_{ij} for a matrix $\boldsymbol{X}$, and x_{ijkl} for an order 4 tensor $\boldsymbol{\mathcal{X}}$. The ith column of a matrix $\boldsymbol{X}$ is denoted by $\boldsymbol{x}_i$. The mode-n matrix product of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{s_1 \times \cdots \times s_N}$ with a matrix $\boldsymbol{A} \in \mathbb{R}^{J \times s_n}$ is denoted by $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{A}$, with the result having dimensions $s_1 \times \cdots \times s_{n-1} \times J \times s_{n+1} \times \cdots \times s_N$. Matricization is the process of reshaping a tensor into a matrix. Given a tensor $\boldsymbol{\mathcal{X}}$ the mode-n matricized version is denoted by $\boldsymbol{X}_{(n)} \in \mathbb{R}^{s_n \times K}$, where $K = \prod_{m=1, m \neq n}^{N} s_m$. We use parenthesized superscripts as labels for different tensors and matrices, e.g., $\boldsymbol{A}^{(1)}$ and $\boldsymbol{A}^{(2)}$ are different matrices.

The Hadamard product of two matrices $\boldsymbol{U}, \boldsymbol{V} \in \mathbb{R}^{I \times J}$ resulting in matrix $\boldsymbol{W} \in \mathbb{R}^{I \times J}$ is denoted by $\boldsymbol{W} = \boldsymbol{U} * \boldsymbol{V}$, where $w_{ij} = u_{ij} v_{ij}$. The inner product of matrices $\boldsymbol{U}, \boldsymbol{V}$ is denoted by $\langle \boldsymbol{U}, \boldsymbol{V} \rangle = \sum_{i,j} u_{ij} v_{ij}$. The outer product of K vectors $\boldsymbol{u}^{(1)}, \ldots, \boldsymbol{u}^{(K)}$ of corresponding sizes $s_1, \ldots, s_K$ is denoted by $\boldsymbol{\mathcal{X}} = \boldsymbol{u}^{(1)} \circ \cdots \circ \boldsymbol{u}^{(K)}$, where $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{s_1 \times \cdots \times s_K}$ is an order K tensor.

For matrices $\boldsymbol{A} \in \mathbb{R}^{I \times K} = \big[\boldsymbol{a}_1, \ldots, \boldsymbol{a}_K\big]$ and $\boldsymbol{B} \in \mathbb{R}^{J \times K} = \big[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_K\big]$, their Khatri–Rao product resulting in a matrix of size $(IJ) \times K$ defined by $\boldsymbol{A} \odot \boldsymbol{B} = [\boldsymbol{a}_1 \otimes \boldsymbol{b}_1, \ldots, \boldsymbol{a}_K \otimes \boldsymbol{b}_K]$, where $\boldsymbol{a} \otimes \boldsymbol{b}$ denotes the Kronecker product of the two vectors.

2.2. CP decomposition with ALS.

The CP tensor decomposition [16, 18] for an input tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{s_1 \times \cdots \times s_N}$ is denoted by

$$\boldsymbol{\mathcal{X}} \approx [\![\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}]\!], \quad \text{where} \quad \boldsymbol{A}^{(i)} = [\boldsymbol{a}_1^{(i)}, \ldots, \boldsymbol{a}_r^{(i)}],$$

and serves to approximate a tensor by a sum of R tensor products of vectors,

$$\boldsymbol{\mathcal{X}} \approx \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \cdots \circ \boldsymbol{a}_r^{(N)}.$$

The CP-ALS method alternates among quadratic optimization problems for each of the factor matrices $\boldsymbol{A}^{(n)}$, resulting in linear least squares problems for each row,

$$\boldsymbol{A}_{\text{new}}^{(n)} \boldsymbol{P}^{(n)T} \cong \boldsymbol{X}_{(n)},$$

where the matrix $\boldsymbol{P}^{(n)} \in \mathbb{R}^{I_n \times R}$, where $I_n = \prod_{i=1, i\neq n}^{N} s_i$ is formed by Khatri–Rao products of the other factor matrices,

$$\boldsymbol{P}^{(n)} = \boldsymbol{A}^{(N)} \odot \cdots \odot \boldsymbol{A}^{(n+1)} \odot \boldsymbol{A}^{(n-1)} \odot \cdots \odot \boldsymbol{A}^{(1)}. \tag{2.1}$$

These linear least squares problems are often solved via the normal equations [26],

$$\boldsymbol{A}_{\text{new}}^{(n)} \boldsymbol{\Gamma}^{(n)} \leftarrow \boldsymbol{X}_{(n)} \boldsymbol{P}^{(n)},$$

where $\boldsymbol{\Gamma} \in \mathbb{R}^{R \times R}$ can be computed via

$$\boldsymbol{\Gamma}^{(n)} = \boldsymbol{S}^{(1)} * \cdots * \boldsymbol{S}^{(n-1)} * \boldsymbol{S}^{(n+1)} * \cdots * \boldsymbol{S}^{(N)} \tag{2.2}$$

with each $\boldsymbol{S}^{(i)} = \boldsymbol{A}^{(i)T} \boldsymbol{A}^{(i)}$. These equations also give the nth component of the optimality conditions for the unconstrained minimization of the nonlinear objective function,

$$f(\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}) := \frac{1}{2} ||\boldsymbol{\mathcal{X}} - [\![\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}]\!]||_F^2. \tag{2.3}$$

The *matricized tensor times Khatri–Rao product* or MTTKRP computation $\boldsymbol{M}^{(n)} = \boldsymbol{X}_{(n)} \boldsymbol{P}^{(n)}$ is the main computational bottleneck of CP-ALS [8]. A work efficient way to compute MTTKRP is to contract the factor matrices with the tensor successively. The bottleneck for this implementation is the contraction between the tensor and the first-contracted matrix. Algebraically, this contraction can be written as the tensor times matrix product, $\boldsymbol{\mathcal{X}} \times_i \boldsymbol{A}^{(i)T}$. For a rank-R CP decomposition, this computation has the cost $2s^N R$ if $s_n = s$ for all $n \in \{1, \ldots, N\}$.

The dimension-tree algorithm for ALS [7, 22, 23, 24, 40, 58] uses a fixed amortization scheme to update MTTKRP in each ALS sweep. This scheme only needs to perform two tensor times matrix contraction calculations for each ALS sweep, decreasing the leading order cost of a sweep from $O(Ns^N R)$ to $O(s^N R)$.

3. Gauss–Newton for CP decomposition. The Gauss–Newton (GN) method is a modification of Newton's method to solve nonlinear least squares problem for a quadratic objective function defined as

$$\phi(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x})\|^2,$$

where $\boldsymbol{y}$ is the given vector of points with respect to which we solve the least squares problem, $\boldsymbol{x}$ is the solution vector required, and $\boldsymbol{f}$ is the nonlinear function of $\boldsymbol{x}$ given in the problem. The gradient and the Hessian matrix of $\phi(\boldsymbol{x})$ can be expressed as

$$\nabla \phi(\boldsymbol{x}) = \boldsymbol{J}_r^T(\boldsymbol{x}) \boldsymbol{r}(\boldsymbol{x}),$$
$$\boldsymbol{H}_\phi(\boldsymbol{x}) = \boldsymbol{J}_r^T(\boldsymbol{x}) \boldsymbol{J}_r(\boldsymbol{x}) + \sum_i r_i(\boldsymbol{x}) \boldsymbol{H}_{r_i}(\boldsymbol{x}),$$

where $\boldsymbol{r}(\boldsymbol{x})$ is the residual function defined as $\boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x})$, $\boldsymbol{J}_r(\boldsymbol{x})$ is the Jacobian matrix of the residual function with respect to $\boldsymbol{x}$, and $\boldsymbol{H}_{r_i}(\boldsymbol{x})$ is the Hessian matrix of a component of the residual function r_i with respect to $\boldsymbol{x}$.

The GN method leverages the fact that $\boldsymbol{H}_{r_i}(\boldsymbol{x})$ is small in norm when the residual is small, to approximate the Hessian as $\boldsymbol{H}_\phi(\boldsymbol{x}) \approx \boldsymbol{J}_r^T(\boldsymbol{x})\boldsymbol{J}_r(\boldsymbol{x})$. Consequently, the GN iteration aims to perform the update,

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - (\boldsymbol{J}_r^T(\boldsymbol{x}^{(k)})\boldsymbol{J}_r(\boldsymbol{x}^{(k)}))^{-1}\boldsymbol{J}_r^T(\boldsymbol{x}^{(k)})\boldsymbol{r}(\boldsymbol{x}^{(k)}),$$

where $\boldsymbol{x}^{(k)}$ represents $\boldsymbol{x}$ at the kth iteration. This linear system corresponds to the normal equations for the linear least squares problem,

$$\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}^{(k)})(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}) \cong -\boldsymbol{r}(\boldsymbol{x}^{(k)}).$$

Approximation with CP decomposition (2.3) is a nonlinear least squares problem where the points are tensor entries and the unknowns are factor matrix entries.

We define the Jacobian tensor as

$$\boldsymbol{\mathcal{J}} = [\boldsymbol{\mathcal{J}}^{(1)}, \ldots, \boldsymbol{\mathcal{J}}^{(N)}]$$

for the N-dimensional CP decomposition, where $\boldsymbol{\mathcal{J}}^{(n)} \in \mathbb{R}^{s_1 \times \cdots \times s_N \times s_n \times R}$ is the Jacobian tensor for the residual tensor with respect to $\boldsymbol{A}^{(n)}$, and is expressed elementwise as

$$j^{(n)}_{i_1 \ldots i_N k r} = \Bigg(- \prod_{m=1, m \neq n}^{N} a^{(m)}_{i_m r} \Bigg) \delta_{i_n k}. \tag{3.1}$$

Another way to derive the Jacobian matrices is by unfolding the factor matrices and the residual function as suggested in [1]. Factorization of the Hessian to solve a linear system in GN has a cost of $O(N^3 s^3 R^3)$. More advanced approaches to solving the Hessian can achieve a cost of $O(NR^6)$ [53], but this reduction is not substantial when the CP rank is high, i.e., $R \geq s$.

Alternatively, CG with implicit matrix products can be used to solve the linear least squares problems in this GN method [48]. Instead of performing a factorization or inversion of the approximate Hessian matrix, this approach only needs to perform matrix vector products $\boldsymbol{J}^T\boldsymbol{J}\boldsymbol{v}$ at each iteration (henceforth we drop the subscript r from $\boldsymbol{J}_r$ and simply refer to $\boldsymbol{J}$ for the matrix form of the Jacobian and $\boldsymbol{\mathcal{J}}$ for its tensor form). We derive the matrix vector product in terms of tensor contractions in the following section.

3.1. GN with implicit CG. With the Jacobian tensors defined in (3.1), the matrix-matrix product $\boldsymbol{H} = \boldsymbol{J}^T\boldsymbol{J}$ can be expressed as an operator with the following form,

$$h^{(n,p)}_{krlz} = \sum_{i_1 \ldots i_N} j^{(n)}_{i_1 \ldots i_N k r} j^{(p)}_{i_1 \ldots i_N l z},$$

which can be simplified to

$$h^{(n,p)}_{krlz} = \begin{cases} \delta_{kl}\Gamma^{(n,n)}_{rz} & \text{if } n = p, \\ a^{(n)}_{kz} a^{(p)}_{lr} \Gamma^{(n,p)}_{rz} & \text{otherwise,} \end{cases} \tag{3.2}$$

$$\text{where} \quad \Gamma^{(n,p)}_{rz} = \prod_{m=1, m \neq n,p}^{N} \Bigg(\sum_{i_m} a^{(m)}_{i_m r} a^{(m)}_{i_m z} \Bigg). \tag{3.3}$$

Note that $\mathbf{\Gamma}^{(n,n)} = \mathbf{\Gamma}^{(n)}$ as defined in (2.2). The matrix-vector product $\boldsymbol{Hw}$ can be written as

$$\boldsymbol{Hw} = \sum_{n=1}^{N}\sum_{p=1}^{N}\sum_{l=1}^{s_p}\sum_{z=1}^{R} h_{krlz}^{(n,p)} w_{lz}^{(p)}.$$

The contractions in the innermost summation have the form

$$\sum_{l,z} h_{krlz}^{(n,p)} w_{lz}^{(p)} = \begin{cases} \sum_{z} \Gamma_{rz}^{(n,n)} w_{kz}^{(n)} & \text{if } n = p, \\ \sum_{l,z} a_{kz}^{(n)} a_{lr}^{(p)} \Gamma_{rz}^{(n,p)} w_{lz}^{(p)} & \text{otherwise.} \end{cases} \tag{3.4}$$

Computation of

$$\sum_{n=1}^{N}\sum_{p=1}^{N}\sum_{l,z} h_{krlz}^{(n,p)} w_{lz}^{(p)}$$

requires N^2 contractions of the form

$$\sum_{l,z} h_{krlz}^{(n,p)} w_{lz}^{(p)}$$

for a total cost of $O(N^2 s R^2)$ when each mode of the input tensor has size s and is

$$O\Big(N(\sum_{m=1}^{N} s_m)R^2\Big)$$

in the general case. The matrix representation as well as the detailed contraction order of (3.4) is shown in Algorithm 3.3. Our GN algorithm for CP decomposition is summarized in Algorithm 3.1. This algorithm uses preconditioned CG for matrices (Algorithm 3.2). Note that we avoid reshaping matrices into vectors in this algorithm as it would be a nonnegligible overhead in the distributed setting. Algorithm 3.2 uses implicit matrix vector products, which are described in (3.4) and in Algorithm 3.3.

3.2. Regularization for GN. Since the approximated Hessian is inherently rank deficient [55], we incorporate Tikhonov regularization when solving the linear system, $\boldsymbol{J}^T\boldsymbol{J} + \lambda\boldsymbol{I}$, at each iteration, which corresponds to the Levenberg–Marquardt algorithm [33]. The convergence behavior of the GN method for CP decomposition as well as the CG method used within each GN iteration is sensitive to the choice of regularization parameter.

A common approach to resolve the scaling indeterminacy for the linear least squares problem is to use $\boldsymbol{J}^T\boldsymbol{J} + \lambda\text{diag}(\boldsymbol{J}^T\boldsymbol{J})$, however, this may not be the best way to regularize as mentioned in [33], which is also confirmed by our experimental results. There are several other approaches for choosing the damping parameter and the diagonal matrix at each iteration to ensure local convergence of the algorithm [33], but they require additional objective function or gradient calculations, which are costly in the context of CP decomposition, due to the high computational and communication costs associated with each iteration.

We provide a new heuristic for choosing the damping parameter by varying the regularization at each step. Variable regularization has been used in the past for the GN method, by increasing or decreasing the parameter depending on the value of the objective function at the next iteration [33]. We find that for CP decomposition,

Algorithm 3.1 CP-GN: GN with preconditioned implicit CG for CP decomposition.

Input: Tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{s_1 \times \cdots \times s_N}$, stopping criteria ε, CG stopping criteria ε_{cg}, rank R
Initialize $\{\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}\}$ so each $\boldsymbol{A}^{(n)} \in \mathbb{R}^{s_n \times R}$ is random
while $\sum_{i=1}^{N} \|\boldsymbol{G}^{(i)}\|_F > \varepsilon$ **do**
  Calculate $\boldsymbol{M}^{(n)} = \boldsymbol{X}_{(n)}\boldsymbol{P}^{(n)}$ for $n \in \{1, \ldots, N\}$
    ▷ Using dimension tree with $\boldsymbol{P}^{(n)}$ is defined as in (2.1)
  for $n \in \{1, \ldots, N\}$ **do**
    Calculate $\boldsymbol{\Gamma}^{(n,p)}$ for $p \in \{1, \ldots, N\}$ based on (3.3)
    $\boldsymbol{G}^{(n)} \leftarrow \boldsymbol{A}^{(n)}\boldsymbol{\Gamma}^{(n,n)} - \boldsymbol{M}^{(n)}$
  end for
  Define λ based on varying scheme described in section 3.2
  $\{\boldsymbol{V}^{(1)}, \ldots, \boldsymbol{V}^{(N)}\} \leftarrow$ CP-CG($\{\boldsymbol{G}^{(1)}, \ldots, \boldsymbol{G}^{(N)}\}$, $\{\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}\}$, $\{\boldsymbol{\Gamma}^{(n,p)} : n, p \in \{1, \ldots, N\}\}$, $\varepsilon_{cg}, \lambda$)
  for $n \in \{1, \ldots, N\}$ **do**
    $\boldsymbol{A}^{(n)} \leftarrow \boldsymbol{A}^{(n)} + \boldsymbol{V}^{(n)}$
  end for
end while
return factor matrices $\{\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}\}$ with $\boldsymbol{A}^{(n)} \in \mathbb{R}^{s_n \times R}$

Algorithm 3.2 CP-CG: Preconditioned implicit CG for CP decomposition.

Input: Gradient set $\{\boldsymbol{G}^{(1)}, \ldots, \boldsymbol{G}^{(N)}\}$, factor matrix set $\{\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}\}$, set of $R \times R$ matrices $\{\boldsymbol{\Gamma}^{(n,p)} : n, p \in \{1, \ldots, N\}\}$, stopping criteria ε_{cg}, regularization term λ
for $n \in \{1, \ldots, N\}$ **do**
  $\boldsymbol{P}_{\text{inv}}^{(n)} \leftarrow (\boldsymbol{\Gamma}^{(n,n)} + \lambda \boldsymbol{I})^{-1}$
  Initialize $\boldsymbol{V}^{(n)}$ to zeros
  $\boldsymbol{R}^{(n)} \leftarrow -\boldsymbol{G}^{(n)}$
  $\boldsymbol{Z}^{(n)} \leftarrow \boldsymbol{R}^{(n)}\boldsymbol{P}_{\text{inv}}^{(n)}$
  $\boldsymbol{W}^{(n)} \leftarrow \boldsymbol{Z}^{(n)}$
end for
while $\sum_{i=1}^{N} \|\boldsymbol{R}^{(i)}\|_F > \varepsilon_{cg} \sum_{i=1}^{N} \|\boldsymbol{G}^{(i)}\|_F$ **do**
  for $n \in \{1, \ldots, N\}$ **do**
    ▷ Using implicit matrix vector product as in (3.4)
    $\boldsymbol{Q}^{(n)} \leftarrow \lambda \boldsymbol{W}^{(n)} + \sum_{p=1}^{N} \text{MatVec}(\boldsymbol{A}^{(n)}, \boldsymbol{A}^{(p)}, \boldsymbol{\Gamma}^{(n,p)}, \boldsymbol{W}^{(p)}, n, p)$
  end for
  $\alpha \leftarrow \sum_{n=1}^{N} \langle \boldsymbol{R}^{(n)}, \boldsymbol{Z}^{(n)} \rangle / \sum_{n=1}^{N} \langle \boldsymbol{W}^{(n)}, \boldsymbol{Q}^{(n)} \rangle$
  for $n \in \{1, \ldots, N\}$ **do**
    $\boldsymbol{V}^{(n)} \leftarrow \boldsymbol{V}^{(n)} + \alpha \boldsymbol{W}^{(n)}$
    $\boldsymbol{R}^{(n)} \leftarrow \boldsymbol{R}^{(n)} - \alpha \boldsymbol{Q}^{(n)}$
    $\boldsymbol{Z}^{(n)} \leftarrow \boldsymbol{R}^{(n)}\boldsymbol{P}_{\text{inv}}^{(n)}$
  end for
  $\beta \leftarrow \sum_{n=1}^{N} \langle \boldsymbol{R}^{(n)}, \boldsymbol{Z}^{(n)} \rangle / \sum_{n=1}^{N} \langle \boldsymbol{W}^{(n)}, \boldsymbol{Q}^{(n)} \rangle$
  for $n \in \{1, \ldots, N\}$ **do**
    $\boldsymbol{W}^{(n)} \leftarrow \boldsymbol{Z}^{(n)} + \beta \boldsymbol{W}^{(n)}$
  end for
end while
return updates $\{\boldsymbol{V}^{(1)}, \ldots, \boldsymbol{V}^{(N)}\}$ to factor matrices

Algorithm 3.3 MatVec: Implicit matrix vector product in CP-CG.

Input: Factor matrices $\boldsymbol{A}^{(n)}$ and $\boldsymbol{A}^{(p)}$, $R \times R$ matrix $\boldsymbol{\Gamma}^{(n,p)}$, current guess $\boldsymbol{W}^{(p)}$, n, p
Initialize $\boldsymbol{M}$ to zeros
if $n \neq p$ **then**
    $\boldsymbol{M} \leftarrow \boldsymbol{A}^{(n)}(\boldsymbol{\Gamma}^{(n,p)} * (\boldsymbol{W}^{(p)T}\boldsymbol{A}^{(p)}))$
else
    $\boldsymbol{M} \leftarrow \boldsymbol{W}^{(p)}\boldsymbol{\Gamma}^{(n,p)}$
end if
return $\boldsymbol{M}$

variation of the regularization parameter is useful for getting out of swamps, and adjusting it eagerly helps avoid the need for expensive recomputation of the objective function.

In particular, we define an upper threshold and a lower threshold, and initialize λ near the upper threshold. This large value ensures that we take steps towards the negative gradient direction, and enables CG to converge quickly. Next, we choose a constant hyperparameter $\mu > 1$ and update the λ at each iteration with $\lambda = \lambda/\mu$. This update is continued until λ reaches the lower threshold, and then it is increased by the update $\lambda = \lambda\mu$ until it reaches the upper threshold value and then decreased again. The lower threshold ensures that the conditioning of $\boldsymbol{J}^T\boldsymbol{J}$ does not affect the CG updates.

We show in section 5.1 that this type of varying regularization can significantly improve the convergence probability of the GN method relative to a fixed regularization parameter when an exact CP decomposition exists. We find that this strategy is robust in speed, accuracy, and probability of convergence to global minima across many experiments.

3.3. Preconditioning for CG. Preconditioning is often used to reduce the number of iterations in CG. For CP decomposition, the structure of the GN approximate Hessian $\boldsymbol{H} = \boldsymbol{J}^T\boldsymbol{J}$ admits a natural block-diagonal Kronecker product preconditioner [41]. Each of the N diagonal blocks $\boldsymbol{H}^{(n,n)}$ has a Kronecker product structure, $\boldsymbol{H}^{(n,n)} = \boldsymbol{\Gamma}^{(n,n)} \otimes \boldsymbol{I}$. Consequently, its inverse is

$$\boldsymbol{H}^{(n,n)^{-1}} = \boldsymbol{\Gamma}^{(n,n)^{-1}} \otimes \boldsymbol{I},$$

which can be computed using $O(R^3)$ work per GN iteration and applied with $O(sR^2)$ cost per CG iteration.

We can also use the Cholesky factorization $\boldsymbol{\Gamma}^{(n,n)} = \boldsymbol{L}\boldsymbol{L}^T$,

$$\boldsymbol{H}^{(n,n)} = \boldsymbol{\Gamma}^{(n,n)} \otimes \boldsymbol{I} = (\boldsymbol{L}\boldsymbol{L}^T) \otimes \boldsymbol{I} = (\boldsymbol{L} \otimes \boldsymbol{I})(\boldsymbol{L}^T \otimes \boldsymbol{I}),$$

in which case application of $\boldsymbol{H}^{(n,n)^{-1}}$ can be applied in a stable way via triangular solve. However, we found that performing triangular solves via ScaLAPACK [11] is a bottleneck for parallel execution as backward and forward substitution have polynomial depth. Consequently, we compute the inverse of $\boldsymbol{\Gamma}^{(n,n)}$ and use tensor contractions to apply it in our parallel implementation.

3.4. Complexity comparison between ALS and GN. We present the cost of our GN implementation in Table 1. The right-hand side of the GN iteration is the gradient of the residual function, which can be calculated using dimension trees similar to the ALS algorithm, thus requiring $O(s^N R)$ amount of work, the same as

TABLE 1
Cost comparison between dimension tree ALS and GN methods. Depth is quantified with $\tilde{O}$ to omit logarithmic depth factors associated with summations.

Method	Work	Depth
ALS dimension tree [24]	$O(s^N R + NR^3)$	$\tilde{O}(N + R)$
GN with Cholesky [37]	$O(s^N R + N^3 s^3 R^3)$	$\tilde{O}(NsR)$
GN with fast inverse [41]	$O(s^N R + N^3 R^6)$	$\tilde{O}(NR^2)$
GN with faster inverse [53]	$O(s^N R + NR^6)$	$\tilde{O}(R^2)$
Implicit GN CG step [48]	$O(N^2 sR^2)$	$\tilde{O}(1)$
GN step with I CG iter	$O(s^N R + IN^2 sR^2)$	$\tilde{O}(N + R + I)$
GN step with exact CG	$O(s^N R + N^3 s^2 R^3)$	$\tilde{O}(NsR)$

an ALS sweep. With the use of implicit CG to solve the linear least squares problems in GN, the cost is dominated by the number of CG iterations, each of which requires $O(N^2 sR^2)$ work. In exact arithmetic, CG should converge in at most NsR iterations.

We compare this iterative approach to the best known methods for direct inversion of the approximate Hessian for CP decomposition with GN. These approaches exploit the block structure of the approximate Hessian matrix, achieving a cost of $O(N^3 R^6)$ [41], which may be improved to $O(NR^6)$ at the sacrifice of some numerical stability [53]. These methods accelerate inversion when R is small.

However, CP decomposition may be accelerated by an initial Tucker factorization to reduce each dimension to $O(R)$ by using CANDELINC decomposition where the orthonormal factors of Tucker factorization are used as linear constraints (see [12, 25]). Tucker preserves exact CP rank and is easier to compute than CP (HoSVD is exact provided existence of an exact Tucker decomposition and is near optimal for approximation). When $R \geq s$, it is less clear whether the iterative or direct method is preferred. One overhead of the direct approach is a memory footprint overhead of $O(NR^4)$.

We quantify the work and depth (number of operations along critical path; lower bound on parallel cost) of ALS and alternative methods for GN in Table 1. The depth analysis for GN with CG assumes use of preconditioning with explicit inverse computation. To quantify the depth of direct linear system solves (necessary in ALS and direct GN), we assume standard approaches (e.g., Gaussian elimination), which have a depth equal to matrix dimension, as opposed to polylogarithmic-depth matrix inversion methods [15]. The communication costs associated with ALS and GN methods can be reduced to known analyses for MTTKRP [8], matrix multiplication [46], and Cholesky factorization [6]. This analysis of cost and depth suggests that GN with implicit CG achieves the best cost and parallelism among GN variants when $s = O(R)$. However, ALS generally offers more parallelism than GN with implicit CG when the number of CG iterations is sufficiently large so as to dominate cost.

4. Implementation. We implement both the dimension-tree-based ALS algorithm and GN algorithm in Python.[1] We leverage a backend wrapper for both NumPy and the Python version of Cyclops tensor framework [47], so that our code can be tested and efficiently executed both sequentially and with distributed-memory parallelism for tensor operations. Cyclops provides a high-level abstraction for distributed-memory tensors, including arbitrary tensor contractions and matrix factorizations such as Cholesky and SVD via ScaLAPACK [11]. The ALS implementation is based

[1] Our implementations are publicly available at https://github.com/cyclops-community/tensor_decomposition

on previous work [29] and uses dimension trees to minimize cost. We write both the ALS and GN optimization algorithms in an optimizer class, so that each ALS sweep and GN iteration is encapsulated as a step member function in the optimizer class. This framework can be easily extended to included other optimization algorithms for tensor decompositions.

Our tensor contraction formulation of the GN allows for the method to be easy to implement with NumPy and Cyclops. Both libraries provide an `einsum` routine for tensor contractions specified in Einstein summation notation. Using this routine, the GN method can be specified succinctly as in the following code snippet, where lists of tensors are used to store the factor matrices $\boldsymbol{A}^{(n)}$, components of the input and output matrices (set of vectors) $\boldsymbol{W}^{(p)}$ and $\boldsymbol{Q}^{(n)}$, and matrices $\boldsymbol{\Gamma}^{(n,p)}$.

```
Q = []
for n in range(N):
  Q.append(lamda * W[n])
  for p in range(N):
    if n == p:
      Q[n] += einsum("rz,kz->kr",Gamma[n,p],W[p])
    else:
      Q[n] += einsum("kz,lr,rz,lz->kr", A[n],A[p],Gamma[n,p],W[p])
```

LISTING 1
Implicit matrix-vector product (lines 10-12 *in Algorithm* 3.2*) in the GN method.*

Our current implementation parallelizes over the N^2 matrix vector products for the case of equidimensional tensors. We can embed the list of matrices in a tensor of size $N \times s \times R$ in which the nth slice of size $s \times R$ contains the nth matrix from the list, $\boldsymbol{A}^{(n)}$. We further embed the list of $\boldsymbol{\Gamma}^{(n,n)}$ matrices into a tensor of size $N \times R \times R$ and the list of $\boldsymbol{\Gamma}^{(n,p)}$ into a tensor of size $N \times N \times R \times R$, where the entries along the nth and pth modes, $n \neq p$, are $\boldsymbol{\Gamma}^{(n,p)}$, and zero otherwise. By embedding these matrices into tensors, we can cast the above contractions into two tensor contractions to achieve parallelization over the N^2 original contractions. The first contraction is a batch of N contractions corresponding to the $n = p$ case in (3.4) and the second contraction is a batch of $N^2 - N$ contractions corresponding to the $n \neq p$ case in (3.4).

```
R = einsum("niz,nzr->nir",V,D)
R += einsum("niz,pjr,npzr,pjz->nir",A,A,G,V)
```

LISTING 2
Implicit matrix-vector product with batched tensor contractions.

In the above code snippet, we have the above described batched contractions, where $\boldsymbol{\mathcal{V}}$ embeds the list containing $\boldsymbol{W}^{(n)}$ on line 7 in Algorithm 3.2, $\boldsymbol{\mathcal{R}}$ embeds the list containing $\boldsymbol{Q}^{(n)}$ on line 11 in Algorithm 3.2, while $\boldsymbol{\mathcal{A}}$ is the embedding of the list of factor matrices (line 1 in Algorithm 3.2). $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{G}}$ embed $\boldsymbol{\Gamma}^{(n,n)}$ and $\boldsymbol{\Gamma}^{(n,p)}$, respectively, for $n, p \in \{1, \ldots, N\}$ with $n \neq p$ (line 1 in Algorithm 3.2). With this approach, an extra computation of $O(NsR^2)$ operations is incurred since the contraction includes computation with the diagonal of the embedding tensor $\boldsymbol{\mathcal{G}}$ (which contains $N \times R \times R$ zeros).

5. Numerical experiments. We perform numerical experiments to compare the performance of the dimension-tree-based ALS algorithm and the GN algorithm on both synthetic and application tensors. Our experiments consider four types of tensors.

Tensors made by random matrices. We create these tensors based on known uniformly distributed randomly generated factor matrices $\boldsymbol{A}^{(n)} \in (a,b)^{s\times R}$, $\boldsymbol{\mathcal{X}} = [\![\boldsymbol{A}^{(1)},\ldots,\boldsymbol{A}^{(N)}]\!]$.

Tensors made by Gaussian matrices. We create tensors based on known standard Gaussian distributed randomly generated factor matrices $\boldsymbol{A}^{(n)} \in \mathcal{N}(0,1)^{s\times R}$, $\boldsymbol{\mathcal{X}} = [\![\boldsymbol{A}^{(1)},\ldots,\boldsymbol{A}^{(N)}]\!]$.

Quantum chemistry tensors. We also consider the density fitting tensor (Cholesky factor of the two-electron integral tensor) arising in quantum chemistry. Its CP decomposition yields the tensor hypercontraction format of the two-electron integral tensor, which enables reduced computational complexity for a number of post-Hartree–Fock methods [19]. Acceleration of CP decomposition for this quantity has previously been a subject of study in quantum chemistry [20]. We leverage the PySCF library [50] to generate the three dimensional compressed density fitting tensor, representing the compressed restricted Hartree–Fock wave function of water molecule chain systems with a STO-3G basis set. We vary the number of molecules in the system from 3 to 40, comparing the efficacy of the ALS and GN methods under different settings.

Matrix multiplication tensor. A hard case for CP decomposition is the matrix multiplication tensor, defined as an order three unfolding (combining pairs of consecutive modes) of

$$t_{ijklmn} = \delta_{lm}\delta_{ik}\delta_{nj}.$$

This tensor simulates multiplication of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ via

$$c_{ij} = \sum_{klmn} t_{ijklmn}a_{kl}b_{mn} = \sum_{l} a_{il}b_{lj}.$$

Its exact CP decompositions give different bilinear algorithms for matrix multiplication, including classical matrix multiplication with rank $s^{3/2}$ and Strassen's algorithm [49] with rank $s^{\log_4(7)}$. Determining the minimal CP rank for multiplication of n-by-n matrices with $n \geq 3$ (so $s \geq 9$) is an open problem [38] that is of interest in theory and practice.

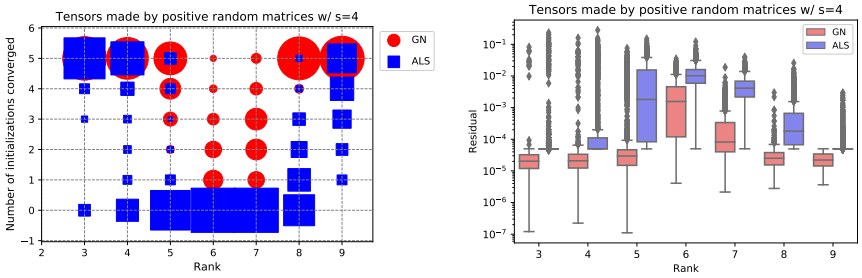
To maintain consistency throughout the experiments, we run CG until a relative tolerance of 10^{-3}. We use the metrics *relative residual* and *fitness* to evaluate the convergence. Letting $\tilde{\boldsymbol{\mathcal{X}}}$ denote the tensor reconstructed by the factor matrices, the relative residual and fitness are

$$r = \frac{\|\boldsymbol{\mathcal{X}} - \tilde{\boldsymbol{\mathcal{X}}}\|_F}{\|\boldsymbol{\mathcal{X}}\|_F}, \quad f = 1 - \frac{\|\boldsymbol{\mathcal{X}} - \tilde{\boldsymbol{\mathcal{X}}}\|_F}{\|\boldsymbol{\mathcal{X}}\|_F}.$$

We collect our experimental results with NumPy backend on a laptop computer, and with Cyclops backend on the Stampede2 supercomputer of the Texas Advanced Computing Center located at the University of Texas at Austin using XSEDE [56].

The laptop is a Macbook Pro with 1.4 GHz Quad-Core Intel Core i5 processor and 16 GB 2133 MHz LPDDR3 memory. On Stampede2, we leverage the Knight's Landing nodes exclusively, each of which consists of 68 cores, 96 GB of DDR RAM, and 16 GB of MCDRAM. These nodes are connected via a 100 Gb/sec fat-tree Omni-Path interconnect. We use Intel compilers and the MKL library for BLAS and batched BLAS routines within Cyclops. We use 64 processes per node on Stampede2 for all experiments.

We study the effectiveness of ALS and GN on CP decomposition based on three metrics.



(a) Tensors made by random factor matrices by elements in $(0, 1)$

(b) Tensors made by random factor matrices by elements in $(0, 1)$

FIG. 1. *Convergence of GN with varying identity regularization and ALS algorithms for recovery of exact CP decomposition with random positive factor matrices. The diameter of the circle and the side length of the square are proportional to the number of problems converged for the corresponding number of initializations in Figure* 1(a). *Figure* 1(b) *is a box-plot of final residual values over different ranks for GN and ALS. The gray points in the plot represent outliers for the corresponding distributions.*

Convergence likelihood. We compare the likelihood of the CP decomposition to recover the original low rank structure of the input tensor with both algorithms.

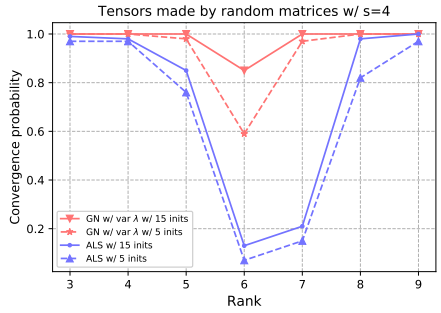
Convergence behavior. We compare the convergence progress with respect to the execution time of ALS and GN for all of the tensors listed above. Experiments are performed with NumPy backend for small- and medium-sized tensors, while the Cyclops backend is used for large tensors.

Parallel performance. We perform a parallel scaling analysis to compare the time for one ALS sweep of the dimension-tree-based ALS algorithm and for a CG iteration of the GN algorithm.

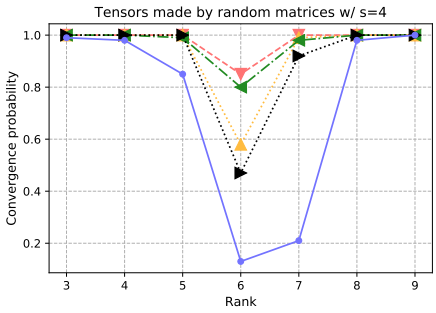
5.1. Convergence likelihood. We compare the convergence likelihood of CP decomposition for random low-rank tensors, optimized with the ALS algorithm and the GN algorithm with constant and varying regularization. We run the algorithms for 100 random samples of each problem. We set the stopping criteria to be that the residual norm is less than 5×10^{-5}, or the norm of the residual change is less than 10^{-7}, or a maximum iteration count is reached (500 and $10,000$ iterations for GN and ALS respectively). We say that the method converged successfully if the solution residual norm is below 5×10^{-5}.

We set the tensor order $N = 3$, size in each dimension $s = 4$, and compare the convergence likelihood under different CP ranks in Figures 1, 2, and 3(a). These results are representative of behavior observed across a variety of choices of s and R.

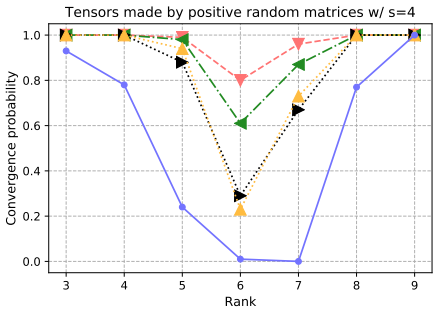
In Figures 1(a) and 1(b), we run GN and ALS with factor matrices sampled from $(0, 1)$ uniformly at random with 5 initializations each for CP ranks ranging from 3 to 9. Note that one can incorporate nonnegativity constraints to improve the speed and convergence of both ALS [7] and GN [39]. In this work, we compare the algorithms without incorporating any constraints. The diameter of the circle and the side length of the square are proportional to the number of problems converged for the corresponding number of initializations in Figure 1(a). It is evident that GN exhibits a higher probability of convergence than ALS as the circles are always bigger than the squares for higher numbers of initializations converged. We observe in the box-plot in Figure 1(b) that GN with varying regularization is more likely to reach a lower



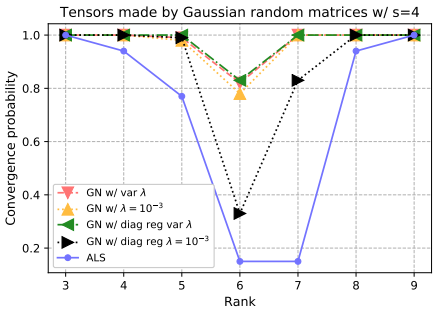
(a) Convergence results for random tensor with 5 and 15 initializations



(b) Convergence results of all the variants for random tensor

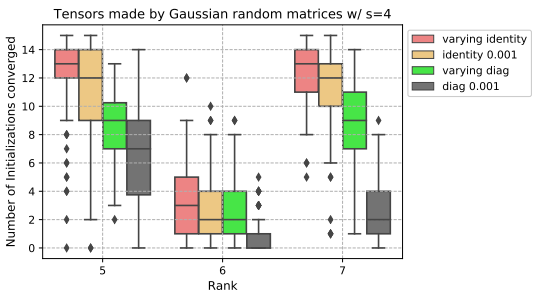


(c) Convergence results of all the variants for positive random tensor

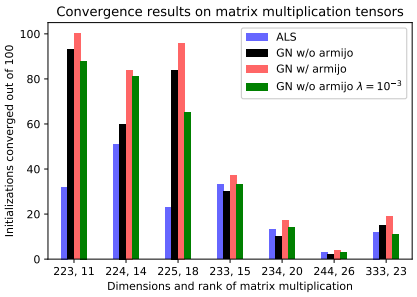


(d) Convergence results of all the variants for Gaussian random tensor

FIG. 2. *Convergence results of various versions of regularization of GN and ALS for recovery of exact CP decomposition with factor matrices with entries selected using uniform random, positive uniform random, and Gaussian distributions.*



(a) Tensors made by factor matrices with standard Gaussian distribution



(b) Matrix multiplication tensors

FIG. 3. *Convergence results of various versions of regularization of GN and ALS for recovery of exact CP decomposition for Gaussian random tensors and matrix mutliplication tensors.*

residual when compared to ALS as the median values of GN are always lower than that of ALS for the corresponding ranks.

In Figures 2(a), we run both algorithms with factor matrices sampled from $(-1, 1)$ uniformly at random with 5 and 15 initializations. A point in the graph represents the probability of at least one initialization converging out of all the initializations. We

observe similar behavior over the various ranks, 6 being the most difficult to converge. The reason for this may be related to the uniqueness of CP decomposition, as the probability is lowest at $R = 6$, when we violate Kruskal's sufficient condition for uniqueness [27]. However, afterwards this probability increases as we increase the rank.

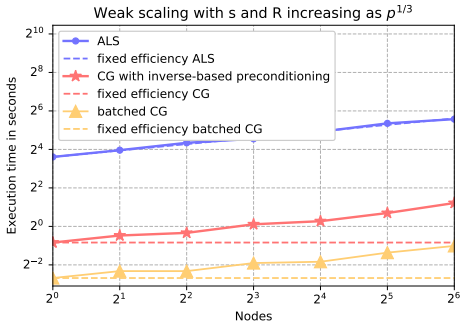
In Figures 2(b), 2(c) and 2(d), we compare all the algorithms with different types of tensors with 15 initializations. These plots indicate that varying regularization achieves the best performance among variants of regularization. ALS is the least robust for these tensors. The convergence probability for varying diagonal regularization for tensors constructed with Gaussian matrices is greater than random matrices. The convergence probability for varying identity regularization remains high for various tensors. This suggests that the convergence behavior with varying identity regularization is more robust. Note that for all the variants, the probability of convergence is lowest for tensors constructed with positive random matrices as the probability of columns of a factor matrix becoming nearly colinear is larger when compared to other tensors. Nearly colinear columns of the factor matrices lead to swamps and make the decomposition problem harder [1].

In Figure 3(a), we compare GN with different regularization techniques for tensors with factor matrices sampled from the standard Gaussian distribution for the "harder" cases (ranks 5 to 7) with 15 initializations. Plotting the number of converged initializations per problem for these variants over the harder cases, we observe that GN with varying identity regularization performs better than other variants of regularization. We also observe that varying the regularization parameter increases the number of converged problems, which corroborates our claim that varying regularization improves the probability of convergence.

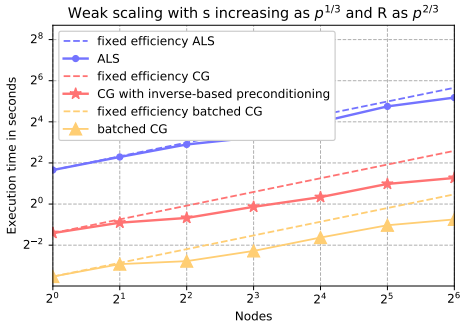
In Figure 3(b), we find the CP decomposition of matrix multiplication tensors with the best known ranks [10]. We use 100 initializations, and set the convergence criteria of the residual as 10^{-8}. We denote by XYZ the matrix multiplication tensor corresponding to the multiplication of matrices of size $X \times Y$ and $Y \times Z$. One can verify that the output CP factors are close to the exact solution if the entries of these matrices are not too large. In our experiments, the maximum infinity norm of the factor matrices for all the solutions is below 50 (49.33 for the 244 tensor with $R = 26$). Therefore, any of these factorizations could be chosen and refined to get an exact solution via techniques used in [10, 45, 52].

For the ALS algorithm, we start with a high regularization parameter, $\lambda = 0.01$ and decrease it gradually, by a factor of 2 after every 100 iterations, which is suggested in [45] to increase the probability for finding the solution. We run ALS for 20,000 iterations. For the GN method, we initialize it with 200 iterations of ALS with $\lambda = 0.01$ and then use GN with proposed regularization and with constant $\lambda = 10^{-3}$. Initialization via ALS with high regularization is done to ensure that we come closer to the solution with a small magnitude of factor matrix entries. Note that this may lead the algorithm to a spurious local minima, and a more robust way is to avoid such an initialization and instead introduce constraints within the GN method. A detailed description of how these constraints on factor matrices can be added along with others, such as sparsity and rational entries, is given in [52]. Without incorporating constraints, the probability of finding the CP decomposition of matrix multiplication tensors decreases with an increase in size. We are not able to find a CP decomposition for the 444 tensor with rank 49 with any of the variants.

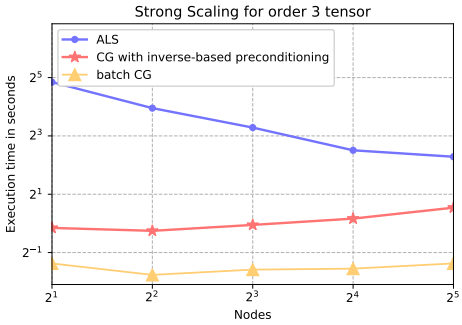
We find that, in this case, using Armijo–Wolfe's condition [4] for step-size control increases the probability of convergence for the GN method, improving upon both



(a) Weak scaling with fixed tensor size to number of processors ratio and tensor dimension to rank ratio



(b) Weak scaling with fixed tensor size to processors ratio and compression ratio



(c) Strong scaling with fixed tensor size

FIG. 4. *Benchmark results for one ALS sweep versus one CG iteration. Each data point is the mean time across* 5 *iterations.*

the constant and variable regularization strategy. The Armijo–Wolfe condition for step size is used for backtracking line search in unconstrained minimization problems. Step size α is initialized to be 1 and reduced by a factor of τ until the condition $f(\boldsymbol{x}) - f(\boldsymbol{x} + \alpha\Delta\boldsymbol{x}) \geq -\alpha c \boldsymbol{\nabla} f(\boldsymbol{x})^T \Delta\boldsymbol{x}$ is satisfied where $c, \tau \in (0, 1)$ or the maximum number of reductions are performed. We set $c = \tau = 0.5$ and perform a maximum of 10 iterations of the line search. We set the relative CG tolerance to be 0.5. This type of step-size control is expensive for larger problems as it requires multiple evaluations of the objective function.

5.2. Parallel performance. We perform a parallel scaling analysis to compare the time for one dimension-tree-based ALS sweep and one CG iteration of the GN algorithm. In Figure 4(a), we consider weak scaling with p processors of order $N = 3$ tensors, starting with dimension $s = 800$ and rank $R = 800$ then growing both by $p^{1/3}$ with increasing number of nodes p. This scaling maintains a fixed memory footprint of the tensor per processor. The work per processor for ALS is $O(s^3 R/p)$, so it increases by a factor of $O(p^{1/3})$ with p processors. For a CG iteration, the work per processor is $O(N^2 s R^2/p)$, which remains constant per processor. Figure 4(a) shows that with the increase of the number of nodes, the time for one ALS sweep scales perfectly while the efficiency for one CG iteration drops to 24% at 64 nodes due to the limited number of operations involved in the Hessian contraction. The Hessian contraction takes up

about half of the time of a CG iteration as inner product and norm calculation take up a significant amount of time. One CG iteration is consistently around 20 times faster than one ALS sweep for different simulation sizes. We observe that explicit calculation and use of the inverse eliminates a significant overhead compared to preconditioning using Cholesky and triangular solves.

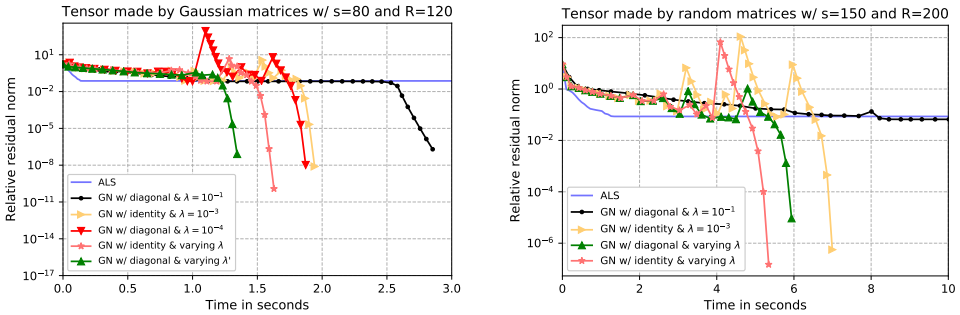
We also implement batch CG which uses batched Hessian contractions as described in section 4. Due to the fact that we extract more parallelism over the N^2 contractions by batching the contractions into a bigger tensor contraction, one CG iteration speeds up by a factor of 3.58 on 1 node and 4.68 on 64 nodes with this implementation, relative to the nonbatched approach.

In Figure 4(b), we consider weak scaling with p processors of order $N = 3$ tensors, starting with dimension $s = 600$ and rank $R = 300$, then growing s as $p^{1/3}$ and R as $p^{2/3}$ with increasing number of nodes p. We consider this type of scaling as the maximum rank of a tensor is $O(s^2)$. This scaling maintains a fixed memory footprint of the tensor and factor matrices per processor while the work per processor for ALS and CG increases by a factor of $O(p^{2/3})$ per processor. Figure 4(b) shows that with the increase of the number of nodes, the time for one ALS sweep and one CG iteration increases and the efficiency improves with growing size. Although Hessian contraction takes up only half of the time of a CG iteration, it takes 0.15 seconds with 1 node and scales up with an efficiency of more than 200%, taking 1.12 seconds with 64 nodes. The increase in efficiency is because the increase in arithmetic intensity is increasing by $O(p^{2/3})$ per process, which leads to a speedup of greater than $O(p)$ for both the algorithms. These observations demonstrate good weak scaling of CG iteration with increasing rank.

For strong scaling, we consider order $N = 3$ tensors with dimension size $s = 1200$ and rank $R = 1200$. Figure 4(c) shows that the CG iteration time increases with the number of nodes, while the ALS sweep time decreases at first, and increases with more than 32 nodes due to communication costs dominating afterwards. The CG iteration involves smaller matrix multiplications, and the contraction time does not scale with increasing node counts on account of the communication costs. The Hessian contraction takes 0.45 seconds with 2 nodes and scales to 0.35 seconds with 16 nodes. Operations such as norm calculation scale worse as they are latency bound, causing the CG iteration time to increase. The batch CG performs larger contractions, which results in improving the time and scaling. The batched contraction takes 0.22 seconds with 2 nodes and scales to 0.13 seconds with 16 nodes. The time taken at 16 nodes is also dominated by the norm and inner product calculations. The ALS sweep is dominated by the MTTKRP calculations, which are more easily parallelizable and therefore allow ALS to achieve better parallel scaling. Overall, we observe that the GN CG iterations contain less parallelism than MTTKRP, but are weakly scalable when the rank is increasing.

5.3. Exact CP decomposition. We compare the convergence behavior of different variants of the GN algorithm with ALS for exact (synthetic) CP decomposition in Figures 5 and 6. We generate low rank tensors of different sizes, the small- and medium-sized tensors are tested with the NumPy backend and the large ones are tested with Cyclops.

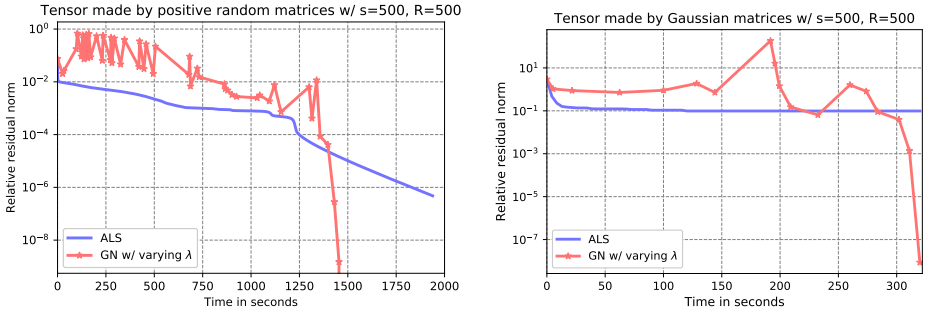
In Figure 5(a) we use CP decomposition on tensors made with standard Gaussian matrices of order $N = 3$, with dimension $s = 80$, and CP rank $R = 120$, using the NumPy backend. We plot different types of regularization for GN along with ALS to study the convergence behavior of different variants of GN. We observe that



(a) Tensor made by random factor matrices with elements distributed with standard normal distribution

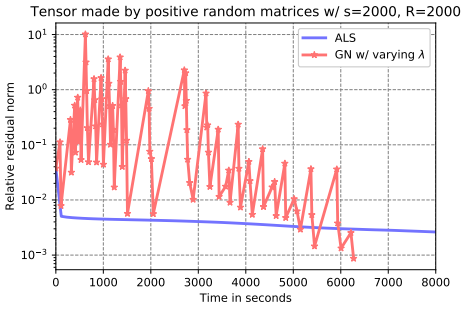
(b) Tensor made by random factor matrices with elements in $(-1, 1)$

FIG. 5. *Relative residual norm versus time for the CP decomposition of synthetic tensors with different sizes. Timings collected using the NumPy backend (sequential).*



(a) Tensor made by random factor matrices with elements in $(0, 1)$ using 256 cores of Stampede2

(b) Tensor made by random factor matrices with elements distributed with standard Gaussian distribution using 256 cores of Stampede2



(c) Tensor made by random factor matrices with elements in $(0, 1)$ using 1024 cores of Stampede2

FIG. 6. *Relative residual norm versus time for the CP decomposition of synthetic tensors with different sizes. Timings collected using the Cyclops backend.*

GN with varying diagonal regularization performs the best and the varying identity regularization is also comparable. The sensitivity to regularization of the GN method is revealed in the plot as constant regularization variants are different from each other. As can be seen in the figure, there are time periods when ALS makes very little progress, and appears to be stuck in a swamp, allowing the GN method to achieve faster convergence for these tensors.

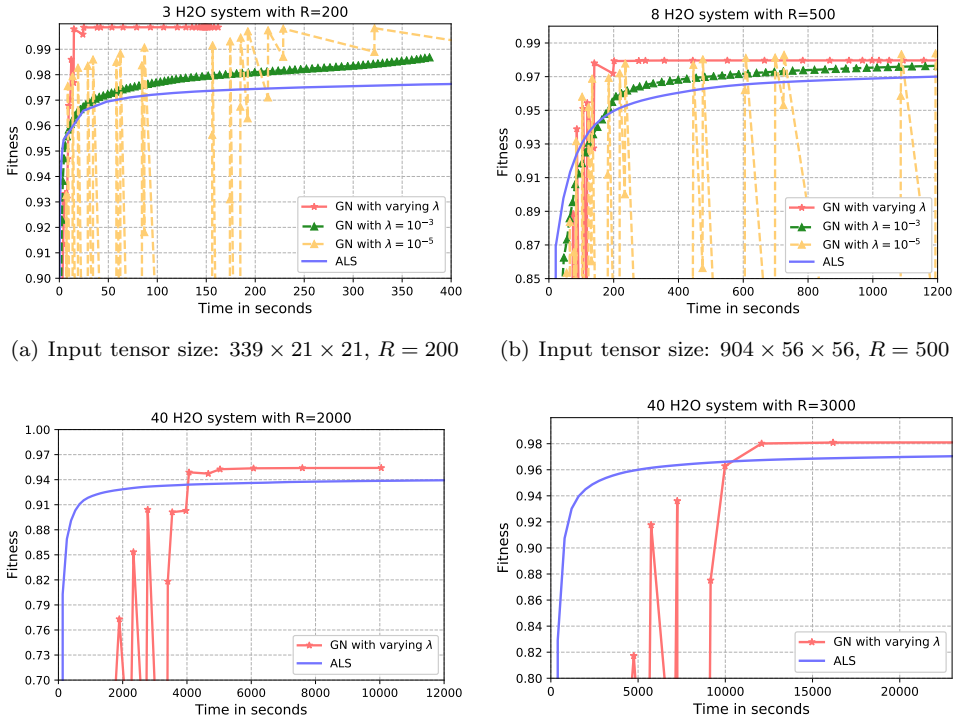
In Figure 5(b), we consider the computation of the CP decomposition for a random low rank tensor of order $N = 3$, with dimension $s = 150$, and rank $R = 200$. We can observe that constant diagonal regularization with $\lambda = 0.1$ is substantially less effective for this tensor. However, the two variants with varying regularization converge quickly, suggesting that varying regularization is a robust technique for random tensors in terms of time to solution.

We test large random low-rank tensors in parallel with $s = 500$ and $R = 500$ on 4 nodes with 256 processes as well as $s = 2000$ and $R = 2000$ on 16 nodes with 1024 processes using the Cyclops backend. GN with identity varying regularization outperforms ALS in terms of speed and accuracy in both cases. As shown in Figure 6(a), for $s = 500$ and $R = 500$, GN with identity varying regularization converges to an exact solution about $1.25\times$ faster than ALS. For the tensor made with standard Gaussian matrices in Figure 6(b), ALS gets stuck in a swamp (makes very little progress in reducing the objective) and GN converges to the exact solution in about 300 seconds, suggesting that even for larger problems GN performs better than ALS. For $s = 2000$ and $R = 2000$ (Figure 6(c)), we let the program run for a fixed time and observe GN with identity varying regularization converge to a lower relative residual, which is about $2.4\times$ more accurate than ALS while running for $0.6\times$ of the time of ALS. Note that the irregularity in time taken of one GN iteration comes from the varying number of CG iterations taken to solve each system of equations.

5.4. Approximate CP decomposition. We also compare the convergence behavior of the GN method with ALS for approximate CP decomposition. In this case, the tensor reconstructed from factor matrices can only approximate the input tensor rather than fully recover it. These experiments consider the density fitting tensors introduced at the beginning of section 5

Our results are shown in Figure 7. We test the problem with different input tensor sizes and different CP ranks. We consider the two smaller problems shown in Figure 7(a) and 7(b), for which we use the NumPy backend. We observe that for both problems, the GN method outperforms the ALS algorithm in speed and final fitness, both with the constant regularization parameter and the regularization variation scheme. In addition, GN with constant regularization may suffer from low optimization stability (when $\lambda = 10^{-5}$) or low accuracy (when $\lambda = 10^{-3}$). The regularization variation scheme collects the advantages of both cases, and can reach high accuracy with a stable convergence.

We consider the larger problems with 40 water molecules in Figures 7(c) and 7(d). These are executed in parallel with Cyclops. Results are collected on 4 nodes using 256 processors on Stampede2. We observe that for these large problems, GN outperforms ALS in terms of speed and fitness. With CP rank 2000, GN can reach a fitness of 0.952 in 5000 seconds, which is higher than the best fitness of ALS (0.94) in about half the time (12,000 seconds), i.e., a speedup of more than $2\times$. We observe a similar convergence behavior of both the algorithms when we increase the rank to 3000.



(a) Input tensor size: $339 \times 21 \times 21$, $R = 200$ (b) Input tensor size: $904 \times 56 \times 56$, $R = 500$

(c) Input tensor size: $4520 \times 280 \times 280$, $R = 2000$ (d) Input tensor size: $4520 \times 280 \times 280$, $R = 3000$

FIG. 7. *Fitness versus time for the CP decomposition of quantum chemistry tensors with different sizes and ranks. The results* (a) *and* (b) *are collected with the NumPy backend, while* (c) *and* (d) *are collected with the Cyclops backend using* 256 *cores of Stampede*2.

6. Conclusion. In this paper, we provide the first efficient parallel implementation of a GN method for CP decomposition. We evaluate a formulation that employs tensor contractions for implicit matrix-vector products within the CG method. The use of tensor contractions enables us to employ the Cyclops library for distributed-memory tensor computations to parallelize the GN approach with a high-level Python implementation. Our results demonstrate good weak scalability for the current implementation of the GN method and show how this formulation could lead to even greater speedups in the Hessian contraction. Additionally, we propose a regularization scheme for the GN method to improve convergence properties without any additional cost. We perform extensive experimentation on different kinds of input tensors and compare the convergence and performance of the GN method relative to ALS. We observe that the GN method typically achieves better convergence as well as performance results for both synthetic as well as quantum chemistry tensors with high CP rank.

REFERENCES

[1] E. ACAR, D. M. DUNLAVY, AND T. G. KOLDA, *A scalable optimization approach for fitting canonical tensor decompositions*, J. Chemom., 25 (2011), pp. 67–86.

[2] A. ANANDKUMAR, R. GE, D. J. HSU, S. M. KAKADE, AND M. TELGARSKY, *Tensor decompositions for learning latent variable models*, J. Mach. Learn. Res., 15 (2014), pp. 2773–2832.

[3] A. Anandkumar, R. Ge, and M. Janzamin, *Guaranteed Non-orthogonal Tensor Decomposition via Alternating Rank-1 Updates*, preprint, arXiv:1402.5180, 2014.
[4] L. Armijo, *Minimization of functions having Lipschitz continuous first partial derivatives*, Pacific J. Math., 16 (1966), pp. 1–3, https://projecteuclid.org:443/euclid.pjm/1102995080.
[5] E. Bailey and S. Aeron, *Word Embeddings via Tensor Factorization*, preprint, arXiv:1704.02686, 2017.
[6] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, *Communication-optimal parallel and sequential Cholesky decomposition*, SIAM J. Sci. Comput., 32 (2010), pp. 3495–3523.
[7] G. Ballard, K. Hayashi, and K. Ramakrishnan, *Parallel nonnegative CP decomposition of dense tensors*, in Proceedings of the 2018 IEEE 25th International Conference on High Performance Computing (HiPC), IEEE Computer Society, Los Alamitos, CA, 2018, pp. 22–31, https://doi.org/10.1109/HiPC.2018.00012.
[8] G. Ballard, N. Knight, and K. Rouse, *Communication lower bounds for matricized tensor times Khatri-Rao product*, in Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, Piscataway, NJ, 2018, pp. 557–567.
[9] C. Battaglino, G. Ballard, and T. G. Kolda, *A practical randomized CP tensor decomposition*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 876–901.
[10] A. R. Benson and G. Ballard, *A framework for practical parallel fast matrix multiplication*, ACM SIGPLAN Not. 50, 2015, pp. 42–53.
[11] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK User's Guide*, Software Environ. Tools 4, SIAM, Philadelphia, 1997.
[12] R. Bro and C. A. Andersson, *Improving the speed of multiway algorithms: Part* II: *Compression*, Chemom. Intell. Lab. Syst., 42 (1998), pp. 105–113, https://doi.org/10.1016/S0169-7439(98)00011-2.
[13] J. D. Carroll, S. Pruzansky, and J. B. Kruskal, *CANDELINC: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters*, Psychometrika, 45 (1980), pp. 3–24.
[14] F. Cong, Q.-H. Lin, L.-D. Kuang, X.-F. Gong, P. Astikainen, and T. Ristaniemi, *Tensor decomposition of EEG signals: A brief review*, J. Neurosci. Methods, 248 (2015), pp. 59–69.
[15] L. Csanky, *Fast parallel matrix inversion algorithms*, in proceedings of the 16th Annual Symposium on Foundations of Computer Science (SFCS 1975), IEEE Computer Society, Los Alamitos, CA, 1975, pp. 11–12.
[16] R. A. Harshman, *Foundations of the PARAFAC Procedure: Models and Conditions for an Explanatory Multimodal Factor Analysis*, manuscript.
[17] K. Hayashi, G. Ballard, Y. Jiang, and M. J. Tobia, *Shared-memory parallelization of MTTKRP for dense tensors*, SIGPLAN Not. 53, 1 (2018), pp. 393–394, https://doi.org/10.1145/3200691.3178522.
[18] F. L. Hitchcock, *The expression of a tensor or a polyadic as a sum of products*, Stud. Appl. Math., 6 (1927), pp. 164–189.
[19] E. G. Hohenstein, R. M. Parrish, C. D. Sherrill, and T. J. Martínez, *Communication: Tensor hypercontraction.* III. *Least-squares tensor hypercontraction for the determination of correlated wavefunctions*, J. Chem. Phys., 137 (2012), 221101, https://doi.org/10.1063/1.4768241,
[20] F. Hummel, T. Tsatsoulis, and A. Grüneis, *Low rank factorization of the Coulomb integrals for periodic coupled cluster theory*, J. Chem. Phys., 146 (2017), 124105.
[21] L. Karlsson, D. Kressner, and A. Uschmajew, *Parallel algorithms for tensor completion in the CP format*, Parallel Comput., 57 (2016), pp. 222–234.
[22] O. Kaya, *High Performance Parallel Algorithms for Tensor Decompositions*, Ph.D. thesis, University of Lyon, Lyon, France, 2017.
[23] O. Kaya and Y. Robert, *Computing dense tensor decompositions with optimal dimension trees*, Algorithmica, 81 (2019), pp. 2092–2121.
[24] O. Kaya and B. Uçar, *Parallel CP Decomposition of Sparse Tensors Using Dimension Trees*, Ph.D. thesis, Inria-Research Centre Grenoble–Rhône-Alpes, Montbonnet-Saint Martin, France, 2016.
[25] H. A. Kiers and R. A. Harshman, *Relating two proposed methods for speedup of algorithms for fitting two- and three-way principal component and related multilinear models*, Chemom. Intell. Lab. Syst., 36 (1997), pp. 31–40, https://doi.org/10.1016/S0169-7439(96)00074-3.
[26] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500.
[27] J. B. Kruskal, *Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics*, Linear Algebra Appl., 18 (1977), pp. 95–138, https://doi.org/10.1016/0024-3795(77)90069-6.

[28] N. Li, S. Kindermann, and C. Navasca, *Some convergence results on the regularized alternating least-squares method for tensor decomposition*, Linear Algebra Appl., 438 (2013), pp. 796–812.
[29] L. Ma and E. Solomonik, *Accelerating Alternating Least Squares for Tensor Decomposition by Pairwise Perturbation*, preprint, arXiv:1811.10573, 2018.
[30] K. Maruhashi, F. Guo, and C. Faloutsos, *MultiAspectForensics: Pattern mining on large-scale heterogeneous networks with tensor analysis*, in Proceedings of the 2011 International Conference on Advances in Social Networks Analysis and Mining, IEEE, Computer Society, Los Alamitos, CA, 2011, pp. 203–210.
[31] B. C. Mitchell and D. S. Burdick, *Slowly converging PARAFAC sequences: Swamps and two-factor degeneracies*, J. Chemom., 8 (1994), pp. 155–168.
[32] D. Mitchell, N. Ye, and H. De Sterck, *Nesterov acceleration of alternating least squares for canonical tensor decomposition: Momentum step size selection and restart mechanisms*, Numer. Linear Algebra Appl., 27 (2020), e2297.
[33] J. J. Moré, *The Levenberg-Marquardt algorithm: Implementation and theory*, in Numerical Analysis, Springer, Berlin, 1978, pp. 105–116.
[34] K. R. Murphy, C. A. Stedmon, D. Graeber, and R. Bro, *Fluorescence spectroscopy and multi-way techniques. PARAFAC*, Anal. Methods, 5 (2013), pp. 6557–6566.
[35] C. Navasca, L. De Lathauwer, and S. Kindermann, *Swamp reducing technique for tensor decomposition*, in Proceedings of the 2008 16th European Signal Processing Conference, IEEE, Piscataway, NJ, 2008, pp. 1–5.
[36] D. Nion and L. De Lathauwer, *An enhanced line search scheme for complex-valued tensor decompositions. Application in DS-CDMA*, Signal Process., 88 (2008), pp. 749–755.
[37] P. Paatero, *A weighted non-negative least squares algorithm for three-way PARAFAC factor analysis*, Chemom. Intell. Lab. Syst., 38 (1997), pp. 223–242.
[38] V. Pan, *How to Multiply Matrices Faster*, Springer. New York, 1984.
[39] A. H. Phan, P. Tichavský, and A. Cichocki, *Fast damped Gauss-Newton algorithm for sparse and nonnegative tensor factorization*, in Proceedings of the 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, Piscataway, NJ, 2011, pp. 1988–1991, https://doi.org/10.1109/ICASSP.2011.5946900.
[40] A.-H. Phan, P. Tichavskỳ, and A. Cichocki, *Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations*, IEEE Trans. Signal Process., 61 (2013), pp. 4834–4846.
[41] A.-H. Phan, P. Tichavský, and A. Cichocki, *Low complexity damped Gauss-Newton algorithms for CANDECOMP/PARAFAC*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 126–147.
[42] M. Rajih, P. Comon, and R. A. Harshman, *Enhanced line search: A novel method to accelerate PARAFAC*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1128–1147.
[43] M. D. Schatz, T. M. Low, R. A. van de Geijn, and T. G. Kolda, *Exploiting symmetry in tensors for high performance: Multiplication with symmetric tensors*, SIAM J. Sci. Comput., 36 (2014), pp. C453–C479.
[44] 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 Trans. Signal Process., 65 (2017), pp. 3551–3582.
[45] A. Smirnov, *The bilinear complexity and practical algorithms for matrix multiplication*, Comput. Math. Math. Phys., 53 (2013), pp. 1781–1795.
[46] E. Solomonik and J. Demmel, *Communication-optimal parallel* $2.5D$ *matrix multiplication and LU factorization algorithms*, in European Conference on Parallel Processing, Springer, Berlin, 2011, pp. 90–109.
[47] E. Solomonik, D. Matthews, J. R. Hammond, J. F. Stanton, and J. Demmel, *A massively parallel tensor contraction framework for coupled-cluster computations*, J Parallel Distrib. Comput., 74 (2014), pp. 3176–3190.
[48] L. Sorber, M. Van Barel, and L. De Lathauwer, *Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-*$(l_r, l_r, 1)$ *terms, and a new generalization*, SIAM J. Optim., 23 (2013), pp. 695–720.
[49] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356, https://doi.org/10.1007/BF02165411.
[50] Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. D. McClain, E. R. Sayfutyarova, S. Sharma, S. Wouters, and G K.-L. Chan, *PySCF: The Python-based simulations of chemistry framework*, Wiley Interdiscip.-Rev. Comput. Mol. Sci., 8 (2018), e1340.

[51] P. S. Thomas and T. Carrington Jr, *An intertwined method for making low-rank, sum-of-product basis functions that makes it possible to compute vibrational spectra of molecules with more than* 10 *atoms*, J. Chem. Phys., 146 (2017), 204110.

[52] P. Tichavský, A.-H. Phan, and A. Cichocki, *Numerical CP decomposition of some difficult tensors*, J. Comput. Appl. Math., 317 (2017), pp. 362–370, https://doi.org/10.1016/j.cam.2016.12.007.

[53] P. Tichavskỳ, A. H. Phan, and A. Cichocki, *A further improvement of a fast damped Gauss-Newton algorithm for CANDECOMP-PARAFAC tensor decomposition*, in Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, Piscataway, NJ, 2013, pp. 5964–5968.

[54] G. Tomasi and R. Bro, *PARAFAC and missing values*, Chemom. Intell. Lab. Syst., 75 (2005), pp. 163–180.

[55] G. Tomasi and R. Bro, *A comparison of algorithms for fitting the PARAFAC model*, Comput. Statist. Data Anal., 50 (2006), pp. 1700–1734.

[56] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. Scott, and N. Wilkins-Diehr, *XSEDE: Accelerating scientific discovery*, Comput. Sci. Eng., 16 (2014), pp. 62–74, https://doi.org/10.1109/MCSE.2014.80.

[57] L. R. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311.

[58] N. Vannieuwenhoven, K. Meerbergen, and R. Vandebril, *Computing the gradient in optimization algorithms for the CP decomposition in constant memory through tensor blocking*, SIAM J. Sci. Comput., 37 (2015), pp. C415–C438.