PARALLEL GAUSS-NEWTON METHOD FOR CP DECOMPOSITION

BY

NAVJOT SINGH

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Applied Mathematics
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2020

Urbana, Illinois

Adviser:

Dr. Edgar Solomonik

ABSTRACT

In this thesis[1], we formulate the Gauss-Newton algorithm to make it viable for running on distributed memory architectures and comparable to alternating least squares algorithm for CP decomposition. alternating least squares may exhibit slow or no convergence, especially when high accuracy is required. CP decomposition problem can be formulated as a non-linear least squares problem to apply iterative Newton-like methods. Direct solution of linear systems involving an approximated Hessian is an expensive approach, however, recent advancements have shown that use of an implicit representation of the linear system makes these methods competitive with alternating least squares in terms of speed. We provide a 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 to that, we introduce a regularization scheme for the Gauss-Newton method which shows better convergence results across a variety of tensors. We study the convergence of variants of the Gauss-Newton method relative to alternating least squares for finding exact CP decompositions as well as approximate decompositions of real-world tensors.

[1]This thesis is based on the work [35] and acknowledges the work of all the authors

To my parents, for their love and support.

ACKNOWLEDGMENTS

I would like to thank Dr. Edgar Solomonik for his advice and support, the time he took to have discussions with me and giving me the opportunity to work with him. I would also like to thank Linjian Ma and all the LPNA members for their constant help and support, Karen Mortensen for her advice and support. Apart from my parents, my sister and brother in law have been a great support throughout my degree and I express my gratitude to them. I would also like to thank my friends, Shivika, Raghav, Avi, Mohit, Simran, Gauransh and others who have been there just a call away from me.

CONTENTS

Chapter 1

INTRODUCTION

1.1 Previous work and contributions

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 [12,16], performance optimization [20,33], and acceleration by randomization [8] for CP decomposition. 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 solution with high resolution is required, which is also called the 'swamp' phenomenon [22]. 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 [19, 26], line search [23, 27, 32] and gradient based methods [1, 28, 31, 38, 42, 44].

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 [28, 43, 44]. These approaches offer quadratic convergence and are better at avoiding the swamps inhibiting performance of ALS. Naive solution of linear equations arising in these method is expensive. For rank-R decomposition of an N dimensional tensor with all the dimension sizes equal to s, standard algorithms either perform Cholesky on the normal equations [28] or QR on the Jacobian matrix [44], yielding a complexity of $O(N^3 s^3 R^3)$. 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 $O(N^3 R^6)$ [31]. A successive study showed that the cost can be further reduced to $O(NR^6)$, albeit the approach can suffer from numerical instabil-

ity [42].

Another approach for performing Gauss-Newton with low cost is to leverage an implicit conjugate gradient (CG) method [38]. 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), an approach that can also be augmented with preconditioning to accelerate CG convergence rate [38]. In comparison to the aforementioned direct methods, this iterative approach 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, for the CP decomposition with rank $R < s$, Tucker decomposition (or simply HoSVD) [46] can be used to effectively compress the input tensor from dimensions of size s to R, and then CP decomposition can be performed.

The main objective of this thesis is to 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 of the smallest dimension size of the input tensor). We consider various approaches to regularize Gauss-Newton with implicit CG, to understand their efficacy, we quantify their ability to converge to exact CP decompositions of synthetic tensor of various CP rank, as well as to approximating 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 Chapter 5 .

The main contribution of this thesis 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 interpolates between a sequential approach based on the NumPy library and the Cyclops backend, enabling both sequential and parallel experimental studies. We detail our implementations in Chapter 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 5.3 and 5.4.

This thesis 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 [38].
- 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 decomposition of both synthetic and application-based tensors with high CP rank.

1.2 Notation and definitions

We use tensor algebra notation in both element-wise form and specialized form for tensor operations [18]. 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 $K = \prod_{m=1, m \neq n}^{N} s_m$ and the matrix is

augmented along the n^{th} mode. 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.

The Kronecker product of matrices $\boldsymbol{A} \in \mathbb{R}^{I \times J}$ and $\boldsymbol{B} \in \mathbb{R}^{K \times L}$ is defined by

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11}\boldsymbol{B} & a_{12}\boldsymbol{B} & \ldots & a_{1J}\boldsymbol{B} \\ a_{21}\boldsymbol{B} & a_{22}\boldsymbol{B} & \ldots & a_{2J}\boldsymbol{B} \\ & & \vdots & \\ a_{I1}\boldsymbol{B} & a_{I2}\boldsymbol{B} & \ldots & a_{IJ}\boldsymbol{B} \end{bmatrix}.$$

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.

1.3 CANDECOMP/PARAFAC decomposition

The CP (canonical polyadic or CANDECOMC/PARAFAC) tensor decomposition is widely used for data analytics in different scientific fields [11, 13, 21, 25, 34], machine learning applications [2, 5, 18], and quantum chemistry [41]. It is denoted by

$$\boldsymbol{\mathcal{X}} \approx [\![\boldsymbol{A}^{(1)}, \cdots, \boldsymbol{A}^{(N)}]\!], \quad \text{where} \quad \boldsymbol{A}^{(i)} = [\boldsymbol{a}_1^{(i)}, \cdots, \boldsymbol{a}_r^{(i)}], \tag{1.1}$$

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

$$\boldsymbol{\mathcal{X}} \approx \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \cdots \circ \boldsymbol{a}_r^{(N)}. \tag{1.2}$$

An n order tensor is said to be in kruskal form if

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

where $\forall r \in \{1 \cdots R\}$, $\lambda_r \in \mathbb{R}_+$ and $\forall i \in \{1 \cdots N\}$, $\|\boldsymbol{a}_r^{(i)}\| = 1$.

The CP decomposition is unique under the scaling and permutation constraints as the corresponding vectors in the factor matrices can be reordered arbitrarily

$$\boldsymbol{\mathcal{X}} = [\![\boldsymbol{A}^{(1)}, \cdots, \boldsymbol{A}^{(N)}]\!] = [\![\boldsymbol{A}^{(1)}\boldsymbol{P}, \cdots, \boldsymbol{A}^{(N)}\boldsymbol{P}]\!],$$

where $\boldsymbol{P}$ is a permutation matrix of size $R \times R$

The indeterminacy in scaling arises to the fact that we can scale the vectors of the corresponding factor matrices as follows

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

as long as $k_r^{(1)} \cdots k_r^{(N)} = 1$, $\forall r \in \{1 \cdots R\}$

As mentioned in [18] the most general result for uniqueness depends on the concept of the rank of factor matrices, where the rank, $\text{rank}(\boldsymbol{X})$ of a matrix $\boldsymbol{X}$ is the number of linearly independent columns of the matrix. A sufficient condition for uniqueness of the CP decompostion is:

$$\sum_{n=1}^{N} \text{rank}(\boldsymbol{A}^{(n)}) \geq 2R + (N - 1).$$

CP decomposition of an input tensor can be computed via different optimization techniques, such as variants of gradient descent [1, 28], deflations [2, 3], and alternating least squares [18]. Two of these, namely Gauss-Newton and alternating least squares will be discussed in the further chapters.

Chapter 2

ALTERNATING LEAST SQUARES FOR CP DECOMPOSITION

2.1 Alternating least squares algorithm

Alternating least squares algorithm aims to minimize the following objective function iteratively:

$$f(\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}) := \frac{1}{2}||\boldsymbol{\mathcal{X}} - [\![\boldsymbol{A}^{(1)}, \cdots, \boldsymbol{A}^{(N)}]\!]||_F^2, \tag{2.1}$$

where $\boldsymbol{\mathcal{X}}$ is the input tensor and $[\![\boldsymbol{A}^{(1)}, \cdots, \boldsymbol{A}^{(N)}]\!]$ is the notation used as in equations 1.1 and 1.2

The algorithm starts with an initialization of the factor matrices and proceeds via alternating minimization, i.e., in the i^{th} subiteration, all the factor matrices except the i^{th} factor matrix are kept fixed and a quadratic subproblem is solved for the best $\boldsymbol{A}^{(i)}$ relative to the fixed factor matrices and input tensor. Each quadratic subproblem maybe solved by computing and solving for the normal equations. The normal equations with respect to the i^{th} factor matrix are derived in the following section.

The alternating minimization ensures that we have a decreasing residual after each iteration but it has guarantees of converging to a local minima under strict assumptions which may not always hold true [47]. Several other works corroborate this claim and our experiments also demonstrate similar results. Nevertheless, the algorithm is very simple which makes it easier to incorporate conditions like non-negativity. Moreover, the algorithm is easy to parallelize making it amenable for solving larger problems on the distributed architecture.

2.2 Normal equations

We derive the normal equations with respect to first factor matrix (as it is simpler in notation but can be easily generalised for all factor matrices) by taking the derivative of the objective function in 2.1 with respect to $\boldsymbol{A}^{(1)}$ and equating it to 0. The objective function in the elementwise notation with all factor matrices except first factor matrix fixed is as follows

$$f(\boldsymbol{A}^{(1)}) = \frac{1}{2} \sum_{i_1,\cdots i_N} \Big(x_{i_1\cdots i_N} - \sum_r a^{(1)}_{i_1 r} \cdots a^{(N)}_{i_N r} \Big) \Big(x_{i_1\cdots i_N} - \sum_s a^{(1)}_{i_1 s} \cdots a^{(N)}_{i_N s} \Big).$$

The derivative is as follows

$$(\nabla f(\boldsymbol{A}^{(1)}))_{i_1 s} = \sum_{i_2,\cdots i_N} \sum_r \Big(a^{(2)}_{i_2 r} \cdots a^{(N)}_{i_N r} a^{(1)}_{i_1 s} \cdots a^{(N)}_{i_N s} \Big) - \sum_{i_2,\cdots i_N} x_{i_1\cdots i_N} a^{(2)}_{i_2 s} \cdots a^{(N)}_{i_N s}. \tag{2.2}$$

Exchanging the sums in the first term and equating $(\nabla f(\boldsymbol{A}^{(1)}))_{i_1 s} = 0$, we get the following

$$\sum_r \Big(a^{(1)}_{i_1 r} \sum_{i_2,\cdots i_N} \big(a^{(2)}_{i_2 r} \cdots a^{(N)}_{i_N r} \big) \big(a_{i_2 s} \cdots a^{(N)}_{i_N s} \big) \Big) = \sum_{i_2,\cdots i_N} x_{i_1\cdots i_N} a^{(2)}_{i_2 s} \cdots a^{(N)}_{i_N s}.$$

When written in matrix form these are the same equations derived in several other previous works

$$\boldsymbol{A}^{(1)} \Big(\big(\boldsymbol{A}^{(2)T} \boldsymbol{A}^{(2)} \big) * \cdots * \big(\boldsymbol{A}^{(N)T} \boldsymbol{A}^{(N)} \big) \Big) = \boldsymbol{X}_{(1)} \big(\boldsymbol{A}^{(2)} \odot \cdots \odot \boldsymbol{A}^{(N)} \big).$$

These can be generalised to the normal equations wrt n^{th} factor matrix as follows

$$\boldsymbol{A}^{(n)} \boldsymbol{\Gamma}^{(n,n)} = \boldsymbol{X}_{(n)} \boldsymbol{P}^{(n)}, \tag{2.3}$$

where $\boldsymbol{\Gamma}^{(n,n)} = (\boldsymbol{A}^{(1)T} \boldsymbol{A}^{(1)}) * \cdots * (\boldsymbol{A}^{(n-1)T} \boldsymbol{A}^{(n-1)}) * (\boldsymbol{A}^{(n+1)T} \boldsymbol{A}^{(n+1)}) \cdots * (\boldsymbol{A}^{(N)T} \boldsymbol{A}^{(N)})$ and $\boldsymbol{P}^{(n)} = \big(\boldsymbol{A}^{(1)} \odot \cdots \odot \boldsymbol{A}^{(n-1)} \odot \boldsymbol{A}^{(n+1)} \cdots \odot \boldsymbol{A}^{(N)} \big)$.

The *Matricized Tensor Times Khatri-Rao Product* or MTTKRP to compute $\boldsymbol{X}_{(n)}\boldsymbol{P}^{(n)}$ is the main computational bottleneck of ALS [7]. Within MTTKRP, the bottleneck is the contraction between the input tensor and the first-contracted matrix, and we call this step *first contraction* of the MTTKRP. 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 of $2s^N R$ if $s_n = s$ for all $n \in \{1, \ldots, N\}$.

The dimension-tree algorithm for ALS [6, 17, 30, 48] uses a fixed amortization scheme to update MTTKRP in each ALS sweep. This scheme only needs to perform two first contraction calculations for each ALS sweep, decreasing the leading order cost of a sweep from $2Ns^N R$ to $4s^N R$.

Chapter 3

GAUSS-NEWTON FOR CP DECOMPOSTION

3.1 Gauss-Newton algorithm

The Gauss-Newton (GN) method is a modification of the Newton's method to solve non-linear 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 non-linear function of $\boldsymbol{x}$ given in the problem. The gradient and the Hessian matrix of $\phi(\boldsymbol{x})$ can be expressed as

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

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 the residual function r_i with respect to $\boldsymbol{x}$.

The Gauss-Newton 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 Gauss-Newton 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 the $\boldsymbol{x}$ at kth iteration. This linear system corresponds

to the normal equations for the linear least squares problem,

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

The CP decomposition can be formulated as a non-linear least square problem in (2.1). 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 element-wise as

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

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 Gauss-Newton has cost $O(N^3 s^3 R^3)$. More advanced approaches to solving Hessian can achieve a cost of $O(NR^6)$ [42], but this reduction is not substantial when CP rank is high, i.e., $R \geq s$.

Alternatively, conjugate gradient (CG) with implicit matrix products can be used to solve the linear least squares problems in this Gauss-Newton method [38]. Instead of performing a factorization or inversion of the approximate Hessian matrix, this approach needs only 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.

3.2 Gauss-Newton with implicit Conjugate Gradient

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

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

We can first take a look at the diagonal blocks of the Hessian

$$h^{(n,n)}_{krlz} = \sum_{i_1 \ldots i_N} \big(- \prod_{m=1, m\neq n}^{N} a^{(m)}_{i_m r} \big) \delta_{i_n k} \big(- \prod_{m=1, m\neq n}^{N} a^{(m)}_{i_m z} \big) \delta_{i_n l}.$$

Taking the sum across i_n and interchanging sum and products we get

$$h^{(n,n)}_{krlz} = \delta_{kl} \prod_{m=1, m\neq n}^{N} \Big(\sum_{i_m} a^{(m)}_{i_m r} a^{(m)}_{i_m z} \Big).$$

The off-diagonal terms are as follows

$$h^{(n,p)}_{krlz} = \sum_{i_1 \ldots i_N} \big(- \prod_{m=1, m\neq n}^{N} a^{(m)}_{i_m r} \big) \delta_{i_n k} \big(- \prod_{m=1, m\neq p}^{N} a^{(m)}_{i_m z} \big) \delta_{i_p l}.$$

Taking $a^{(n)}_{i_n r}$ and $a^{(p)}_{i_p r}$ out of the product and summing across i_n and i_p, we get

$$h^{(n,p)}_{krlz} = \sum_{i_1 \ldots i_{p-1} i_{p+1} \ldots i_{n-1} i_{n+1} \ldots i_N} \big(\prod_{m=1, m\neq n,p}^{N} a^{(m)}_{i_m r} \big) a^{(p)}_{lr} \big(\prod_{m=1, m\neq n,p}^{N} a^{(m)}_{i_m z} \big) a^{(n)}_{kz}.$$

Interchanging the sum and products,

$$h^{(n,p)}_{krlz} = a^{(n)}_{kz} a^{(p)}_{lr} \prod_{m=1, m\neq n,p}^{N} \Big(\sum_{i_m} a^{(m)}_{i_m r} a^{(m)}_{i_m z} \Big).$$

So, the Hessian can be written in the matrix form as follows

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

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

Note that $\boldsymbol{\Gamma}^{(n,n)}$ was also defined in (2.3). The matrix-vector product $\boldsymbol{Hw}$ can be written as

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

The contractions in the innermost summation have the form,

$$\sum_{l,z} h^{(n,p)}_{krlz} w^{(p)}_{lz} = \begin{cases} \sum_{z} \Gamma^{(n,n)}_{rz} w^{(n)}_{kz}, & \text{if } n = p, \\ \sum_{l,z} a^{(n)}_{kz} a^{(p)}_{lr} \Gamma^{(n,p)}_{rz} w^{(p)}_{lz} & \text{otherwise.} \end{cases} \tag{3.4}$$

Computation of $\sum_{n=1}^{N}\sum_{p=1}^{N}\sum_{l,z} h^{(n,p)}_{krlz} w^{(p)}_{lz}$ requires N^2 contractions of the form $\sum_{l,z} h^{(n,p)}_{krlz} w^{(p)}_{lz}$ 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 right hand side is a list of matrices of $\boldsymbol{\mathcal{G}}$ each of size $s_i \times R$ and can be derived as follows

$$g^{(n)}_{kr} = \sum_{i_1 \ldots i_N} j^{(n)}_{i_1 \ldots i_N kr} \Big(x_{i_1 \ldots i_N} - \sum_{z} \prod_{m=1}^{N} a^{(m)}_{i_m z} \Big).$$

Using the equation 3.1 for Jacobian,

$$g^{(n)}_{kr} = \sum_{i_1 \ldots i_N} \Big(- \prod_{m=1, m\neq n}^{N} a^{(m)}_{i_m r} \Big) \delta_{i_n k} \Big(x_{i_1 \ldots i_N} - \sum_{z} \prod_{m=1}^{N} a^{(m)}_{i_m z} \Big).$$

Interchanging sums and product and summing over i_n,

$$\begin{aligned} g^{(n)}_{kr} = - \sum_{i_1 \ldots i_{n-1}, i_{n+1} \ldots i_N} & x_{i_1 \ldots i_{n-1}, k, i_{n+1}, \ldots i_N} a^{(1)}_{i_1 r} \ldots a^{(n-1)}_{i_{n-1} r} a^{(n+1)}_{i_{n+1} r} \ldots a^{(N)}_{i_N r} \\ & + \sum_{z} a^{(n)}_{kz} \prod_{m=1, m\neq n}^{N} \sum_{i_m} a^{(m)}_{i_m z} a^{(m)}_{i_m r}. \end{aligned} \tag{3.5}$$

Equation 3.5 in the matrix form includes the right hand side encountered in equation 2.3

$$\boldsymbol{G}^{(n)} = -\boldsymbol{X}_{(n)} \boldsymbol{P}^{(n)} + \boldsymbol{A}^{(n)} \boldsymbol{\Gamma}^{(n,n)}. \tag{3.6}$$

The right hand side again includes the MTTKRP operation and can be parallelized similar to ALS and $\boldsymbol{\Gamma}^{(n,n)}$ and $\boldsymbol{P}^{(n)}$ we defined in the equation 2.3

Algorithm 1 CP-GN: Gauss-Newton 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**
    ▷ Using dimension tree with $\boldsymbol{P}^{(n)}$ is defined as in (2.3)
    Calculate $\boldsymbol{M}^{(n)} = \boldsymbol{X}_{(n)}\boldsymbol{P}^{(n)}$ for $n \in \{1, \ldots, N\}$
    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 **??**
    $\{\boldsymbol{V}^{(1)}, \ldots, \boldsymbol{V}^{(N)}\} \leftarrow$ CP-CG($\boldsymbol{\mathcal{X}}, \{\boldsymbol{G}^{(1)}, \ldots, \boldsymbol{G}^{(N)}\}$, $\{\boldsymbol{A}^{(1)}, \ldots, \boldsymbol{A}^{(N)}\}$, $\{\boldsymbol{\Gamma}^{(n,p)} : n, p \in \{1, \ldots, N\}\}$, $\varepsilon_{cg}, \lambda$)
    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 $\boldsymbol{A}^{(n)} \in \mathbb{R}^{s_n \times R}$

Also, note that the dimension trees can be used to save cost in the MTTKRP operations and moreover additional parallelization can be achieved by creating copies of the intermediate tensors, i.e., using up more memory and performing the MTTKRP operations in parallel as the same factor matrices are used for each mode unlike ALS where the factor matrices are updated after each subiteration

Our Gauss-Newton algorithm is summarized in algorithm 1.

3.3 Regularization for Gauss-Newton

Since the approximated Hessian is inherently rank-deficient [44], 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 [24]. The convergence behavior of the Gauss-Newton method for CP decomposition as well as the CG method used within each Gauss-Newton 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 regularise as mentioned in [24] and we observe this case with a constant λ parameter. There are several other approaches for choosing the damping parameter and the diagonal matrix at each iteration to ensure local convergence of the algorithm [24], but those are costly in the context of CP decomposition, due to the computational and communication expense associated with each iteration.

We provide a new heuristic approach for choosing the damping parameter by varying the regularization at each step. Variable regularization has been used in the past for the Gauss-Newton method, by increasing or decreasing the parameter depending on the value of the objective function at the next iteration [24]. We find that for CP decomposition, 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.

In particular, we define an upper threshold and a lower threshold, and initialize λ near the upper threshold. This larger value ensures that we take steps towards the negative gradient direction, and enables CG to converge quickly. Next, we choose a constant hyper parameter $\mu > 1$ 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 Gauss-Newton method relative to a fixed regularization parameter when an exact CP decomposition exists. We find that this strategy is robust in speed and convergence probability across many experiments.

3.4 Preconditioning for Gauss-Newton with implicit Conjugate Gradient

Preconditioning is often used to reduce the number of iterations in conjugate gradient. For CP decomposition, the structure of the Gauss-Newton approx-

imate Hessian $\boldsymbol{H} = \boldsymbol{J}^T\boldsymbol{J}$ admits a natural block-diagonal Kronecker product preconditioner [31]. 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 Gauss-Newton iteration and applied in parallel 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 [10] 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.

Chapter 4

IMPLEMENTATION

We implement both dimension tree based ALS algorithm and Gauss-Newton algorithm in Python[1]. We leverage a backend wrapper for both NumPy and the Python version of Cyclops Tensor Framework [37], so that our code can be tested and efficiently executed both sequentially and with distributed-memory parallelism for tensor operations. In addition, we write both the ALS and Gauss-Newton optimization algorithms in an optimizer class, and each ALS sweep / Gauss-Newton 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. Cyclops provides a high-level abstraction for distributed-memory tensors, including arbitrary tensor contractions and matrix factorizations such as Cholesky and SVD via ScaLAPACK [10]. The ALS implementation is based on previous work [20] and uses dimension trees to minimize cost.

Our tensor contraction formulation of the Gauss-Newton method makes it 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 Gauss-Newton 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 $\boldsymbol{U}^{(n)}$, and matrices $\boldsymbol{\Gamma}^{(n,p)}$.

```
u = []
for n in range(N):
  u.append(zeros((s,R)))
  for p in range(N):
    if n == p:
      U[n] += einsum("rz,kz->kr",Gamma[n,p],W[p])
    else:
```

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

```
U[n] += einsum("kz,lr,rz,lz->kr", \
                A[n],A[p],Gamma[n,p],W[p])
```

Listing 4.1: Implicit Matrix-Vector Product in GN Method

Our current implementation does not parallelize over the N^2 matrix vector products yet. This is due to the fact that the Cyclops tensor framework does not support contractions of 'list of tensors' yet. However, for the case of equidimensional tensors, we can cast the list of factor matrices as a tensor and cast the above contractions into two tensor contractions to achieve parallelization over the N^2 contractions. In the following code snippet we have the batched contraction where the input and output list of matrices are tensors $\boldsymbol{\mathcal{V}}$ and $\boldsymbol{\mathcal{R}}$ respectively, $\boldsymbol{\mathcal{D}}$ is the list of $\boldsymbol{\Gamma}^{(n,n)}$, $\boldsymbol{\mathcal{G}}$ is a fourth order tensor where the entries along the first two modes, n^{th} and p^{th} mode is $\boldsymbol{\Gamma}^{(n,p)}$ when $n \neq p$ and a matrix of zeros of size $R \times R$ along the diagonal and $\boldsymbol{\mathcal{A}}$ is the list of factor matrices:

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

Listing 4.2: Implicit Matrix-Vector Product with batched tensor contractions

Note that we still are computing extra work due to the zeros on the diagonal of $\boldsymbol{\mathcal{G}}$ tensor and not computing the contractions with diagonal terms of the Hessian concurrently which could be addressed with 'list of tensors' contraction implemented leading to a single contraction which may even lead to greater speed-ups for a CG iteration.

Chapter 5

EXPERIMENTS

We performed numerical experiments to compare the performance of dimension tree based ALS algorithm and Gauss-Newton algorithm on both synthetic and application tensors. Our experiments consider three types of tensors:

Tensors made by random matrices (Random tensors. We create 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 (Gaussian tensors. We create tensors based on known 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 performed CP decomposition on the density fitting intermediate (Cholesky factor of the two-electron integral tensor) arising in quantum chemistry. This 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 [14]. Acceleration of CP decomposition for this quantity has previously been a subject of study in quantum chemistry [15]. We leverage the PySCF library [40] to generate the three dimensional compressed density fitting tensor, representing the compressed restricted Hartree-Fock wave function of a 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 ALS and Gauss-Newton method 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 [39] 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 [29] that is very important in theory and practice.

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

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

We collect our experimental results with NumPy backend on a Mac OS computer with a 1.4 GHz i5 Quad-core Intel processor with a 16 GB 2133 MHz LPDDR3 RAM and our results with Cyclops backend on the Stampede2 supercomputer Texas Advanced Computing Center located at the University of Texas at Austin using XSEDE [45].

On Stampede2, we leverage the Knight's Landing (KNL) 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 Gauss-Newton on CP decomposition based on the following metrics:

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 w.r.t. execution time of ALS and Gauss-Newton for all 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 simulation time for one ALS sweep of the dimension tree based ALS algorithm and the conjugate gradient iteration of the Gauss-Newton algorithm.

5.1 Convergence likelihood

We compare the convergence likelihood of CP decomposition for random low-rank tensors, optimized with ALS algorithm and Gauss-Newton algorithm with constant and varying regularization. We run the algorithms until the residual norm is less than 5×10^{-5}, or the norm of the update is less than 10^{-7}, or a maximum of 500 and 10,000 iterations for Gauss-Newton and ALS, respectively. The results are presented in Figure 5.1. We set the tensor order $N = 3$, size in each dimension $s = 4$, and compare the convergence likelihood under different CP ranks. These results are representative of behavior observed across a variety of choices of s and R.

In figure 5.1a and 5.1b, we run Gauss-Newton and ALS on 100 problems with factor matrices sampled from $(0, 1)$ with 5 initializations each for CP rank ranging from 3 to 9. 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 5.1a. It is evident that Gauss-Newton exhibits a higher probability of convergence than ALS as the circles are always bigger than the squares for higher number of initializations converged. We can observe in 5.1b that Gauss-Newton with varying regularization is more likely to reach a lower residual when compared with ALS (giving both ample number of iterations).

In figure 5.1d, we compare Gauss-Newton with different regularization techniques for tensors with the same set up and 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 Gauss-Newton with varying identity regularization performs better than varying diagonal regularization which is better than the constant diagonal regularization at convergence as the initializations converged for the corresponding techniques are statistically greater corroborating our claim that varying regularization

improves the probability of convergence.

In figure 5.1c, we run both algorithms with the same set-up and factor matrices sampled from $(-1, 1)$ with 5 and 15 initializations. A point in the graph represents the probability of at least one initialization converging out of the total initializations. We observe similar behaviour over the various ranks, 6 being the most difficult to converge. However, with increasing the initializations we observe increase in the convergence probability for both the algorithms but Gauss-Newton with identity varying regularization outperforms ALS.

In figure 5.1e, 5.1f and 5.1g, we compare all the algorithms with different types of tensors under the same set up with 15 initializations. These plots indicate that varying regularization improves convergence for both the variants of regularization in various types of tensors whereas ALS does not do well at convergence for these 2 types of tensors for the 'harder' cases. Moreover, the cluster for the varying identity regularization for various tensors suggests that the convergence probability of the method is invariant to how the tensors were constructed.

In figure 5.1h, we find the CP decomposition of matrix multiplication tensors with best known ranks [9] with a 100 initializations. For 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 [36] to increase the probability for finding the CP decomposition. We run ALS for 20000 iterations and the convergence criteria is set at 10^{-8}. For Gauss-Newton method, we initialize it with 200 iterations of ALS with $\lambda = 0.01$ and then use Gauss-Newton with proposed regularization and with constant $\lambda = 10^{-3}$. We found that in this case using Armijo's condition [4] for step-size control increases the probability of convergence for Gauss-Newton method which is more than both the constant and variable regularization strategy. We do not observe the same pattern where varying the regularization increases the convergence probability for Gauss Newton in this case as the regularization is not on the L^2 norm of the factor matrices as opposed to ALS (which is shown to work better in this case). However, with Armijo's condition incorporated, convergence probability of Gauss-Newton is more than ALS with the mentioned regularization strategy.

5.2 Parallel Performance

We perform a parallel scaling analysis to compare the simulation time for one dimension tree based ALS sweep and one conjugate gradient iteration of the Gauss-Newton algorithm. For weak scaling, on p processors, we consider order $N = 3$ tensors starting with dimension $s = 800$ and rank $R = 800$ and growing both as $p^{1/3}$ with increasing number of nodes p. Figure 5.2a shows that with the increase of number of nodes, the time for both one ALS sweep and one conjugate gradient iteration increases. This increase is expected, as the amount of work per processor grows as $p^{1/3}$. One conjugate gradient iteration is consistently around 8 times faster than one ALS sweep for different simulation sizes, and both approaches achieve good weak scalability. We observe that explicit calculation and use of the inverse eliminates a significant overhead associated with preconditioning using Cholesky and triangular solves.

We also implement the all at once CG or batch CG as described in chapter 4. A CG iteration speeds up by a factor of 3 approximately (exact value of 2.91) with this implementation for each node count and demonstrates a better weak scaling which is due to the fact that we extract more parallelism over the N^2 contractions by batching the contractions into a bigger tensor contraction.

For strong scaling, we consider order $N = 3$ tensors with dimension size $s = 1600$ and a rank $R = 1600$ CP decomposition. Figure 5.2b shows that the conjugate gradient iteration time increases with the number of nodes, while the ALS sweep time decreases at first, and increases with more than 32 nodes. The conjugate gradient iteration involves smaller matrix multiplications, and becomes dominated by communication and hence the contraction time does not scale with increasing node counts, moreover for operations like calculating norm the scaling becomes worse as they are latency bound and hence the time increases for the iteration. For the batch CG we have a bigger contraction which does scale a little bit which results in improving the scaling but the time taken is dominated by the norm and inner product calculations which take up about more than half time of the CG iteration.

The ALS sweep is dominated by the MTTKRP calculations, which are more easily parallelizable and therefore make ALS achieves better scaling. Overall, we observe that the Gauss-Newton CG iterations contain less par-

allelism than MTTKRP, but are weakly scalable.

5.3 Exact CP decomposition

We compare the convergence behavior of different variants of the Gauss-Newton algorithm with ALS for exact (synthetic) CP decomposition in Figure 5.3. We generate low rank tensors of different sizes, the smaller tensors are tested with NumPy backend and the larger ones with parallel Cyclops backend.

In Figure 5.3a we use CP decomposition on the gaussian low rank tensor with tensor order $N = 3$, size of each dimension $s = 80$ and CP rank $R = 120$ with NumPy backend. We plot different types of regularization for Gauss-Newton along with ALS to study the convergence behavior the best variants of Gauss-Newton. We observe that Gauss-Newton with varying diagonal regularization performs the best and the varying identity regularization is also comparable. The sensitivity to regularization of the Gauss-Newton method is revealed in the plot as constant regularization variants are very different from each other. ALS algorithm does not make any improvement over a long time and appears to be stuck in a swamp, suggesting Gauss-Newton method is preferable for Gaussian tensors.

In Figure 5.3b, we consider the computation of CP decomposition for random low rank tensor of order 3, size of each dimension $s = 150$ and rank $R = 200$. Here we can observe that the diagonal constant regularization may not be useful for this tensor as we don't make any improvement over a long time however, the other two variants with varying regularization converge fast enough again suggesting that varying the regularization is a robust technique for random tensors in terms of speed as well.

We test large random low-rank tensors in parallel with $s = 500$, $R = 500$ on 4 nodes with 256 processes as well as $s = 2000$, $R = 2000$ on 16 nodes with 1024 processes using the Cyclops backend. Gauss-Newton with identity varying regularization outperforms ALS in terms of speed and accuracy in both the cases. For $s = 500$, $R = 500$ as shown in Figure 5.3c Gauss-Newton with identity varying regularization converges to an exact solution about 1.25x faster than ALS which converges to a relative residual of around 10^{-6} and for the tensor made with standard gaussian matrices in Figure 5.3d ALS

gets stuck in a swamp and Gauss-Newton converges to the exact solution in about 300 seconds suggesting that for larger tensors Gauss-Newton maybe a more suitable algorithm. For $s = 2000$, $R = 2000$ as shown in Figure 5.3e, due to constraint in resources we let the algorithms run for a fixed time and observe Gauss-Newton with identity varying regularization converging to a lower relative residual of about 10^{-3} and ALS being about 1.3x slower than Gauss-Newton (till the max run time).

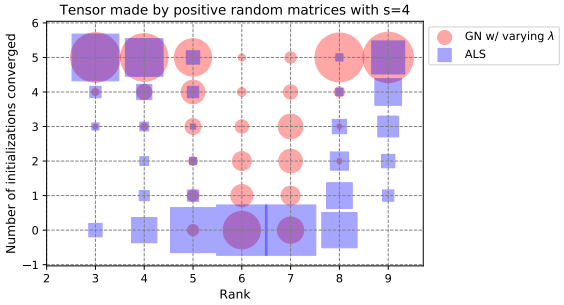
Note that the irregularity in time taken of one Gauss-Newton iteration arises because of the variable number of CG iterations taken to solve the system of equations.

5.4 Approximate CP decomposition

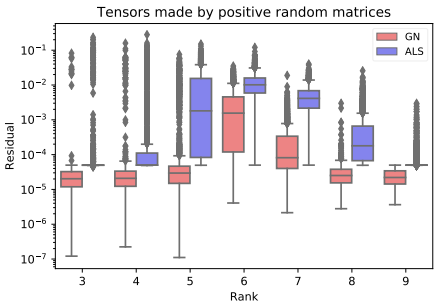
We also compare the convergence behavior of Gauss-Newton method with ALS for approximate CP decomposition, in which case the tensor reconstructed from factor matrices can only approximate the input tensor rather than fully recover it. We test on the quantum chemistry tensors (density fitting intermediates). Our results are shown in Figure 5.4. We test the problem with different input tensor sizes and different CP ranks. We run the two small sized problems shown in Figure 5.4a, 5.4b with NumPy backend. We observe that for both problems, Gauss-Newton method outperforms ALS algorithm in speed and final fitness, both with the constant regularization parameter and the regularization variation scheme. In addition, Gauss-Newton with constant regularization may suffers from low optimization stability, as can be seen when $\lambda = 10^{-5}$, or low accuracy, as can be seen when $\lambda = 10^{-3}$. The regularization variation scheme collects the advantages of both cases, and can reach high accuracy with a stable convergence.

We run the large sized problems set up with 40 water molecules' system shown in Figure 5.4c, 5.4d in parallel with Cyclops backend. We observe that for these large problems, Gauss-Newton also beats ALS in speed and fitness. With CP rank equals 2,000, Gauss-Newton can reach the fitness of 0.952 in 5,000 seconds which is higher than the best fitness of ALS (0.94) in about half the time (in 12,000 seconds), i.e., a speed up of more than 2x. The oscillations in Gauss-Newton maybe controlled by using a smaller factor μ and the number of CG iterations can be reduced by using a lower

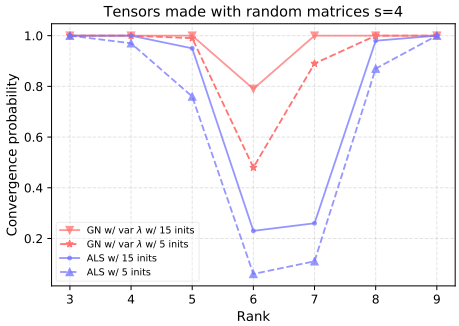
regularization near the optimal solution so as to reduce the perturbation in the system of equations. The observations are similar when we increase the rank to 3000.



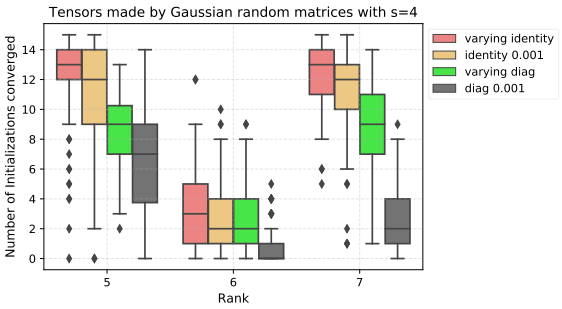
(a) Tensor made by random matrices with elements in $(0, 1)$



(b) Tensor made by random matrices with elements in $(0, 1)$

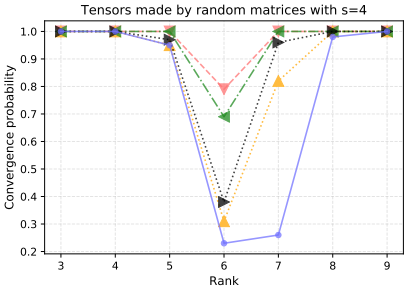


(c) Convergence results with 5 and 15 initializations

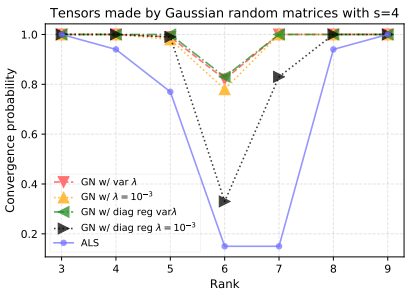


(d) Tensors made by matrices with standard gaussian distribution

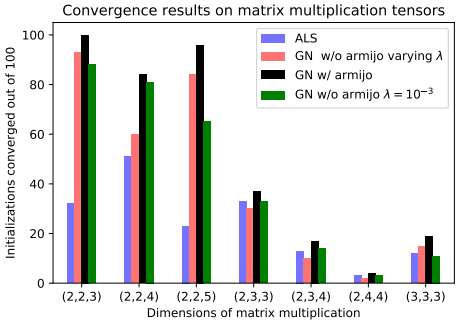
(e) Convergence results of all the variants with 15 initializations



(f) Convergence results of all the variants with 15 initializations

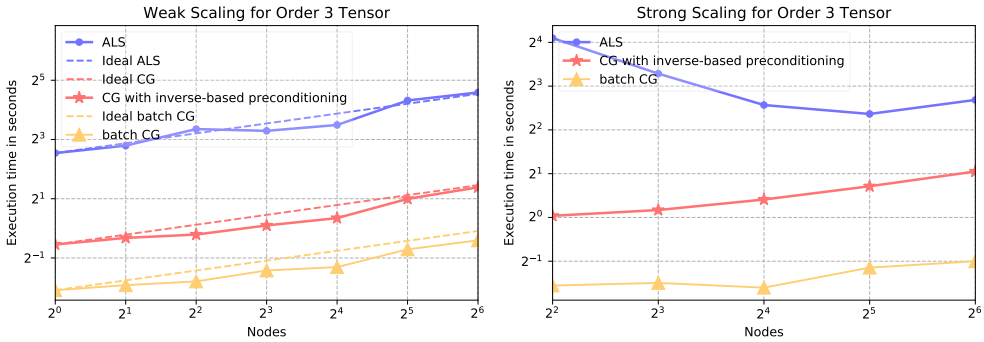


(g) Convergence results of all the variants with 15 initializations



(h) Matrix multiplication tensors

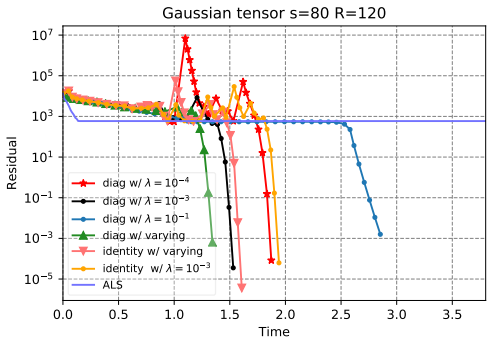
Figure 5.1: Convergence results to compare versions of regularization of Gauss Newton and ALS algorithms



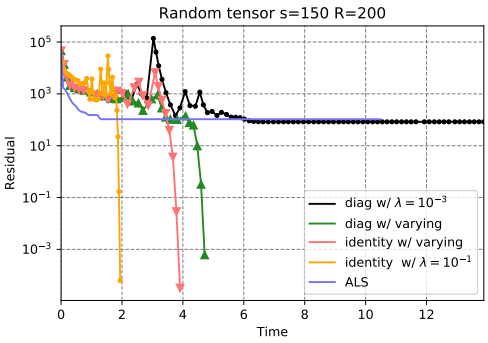
(a) Weak scaling with tensor size to number of processors used ratio is fixed

(b) Strong scaling with fixed tensor size

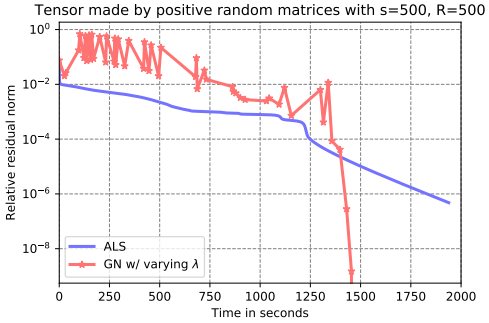
Figure 5.2: Benchmark results for one ALS sweep vs one CG iteration. Each data point is the mean time across 5 iterations.



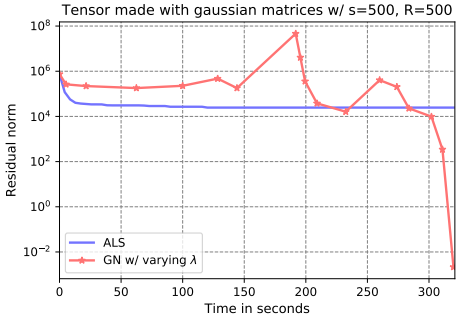
(a) Tensor made by random matrices with elements distributed with standard Gaussian distribution



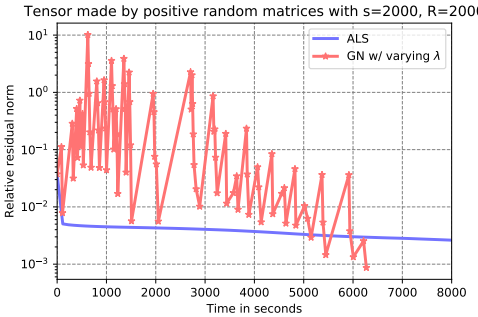
(b) Tensor made by random matrices with elements in $(-1, 1)$



(c) Tensor made by random matrices with elements in $(0, 1)$

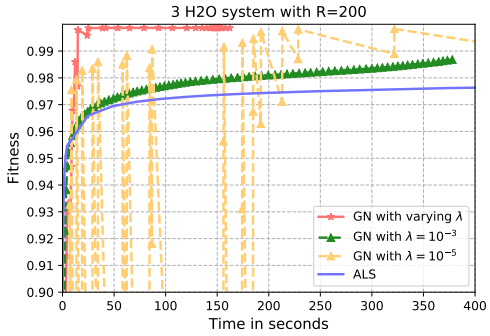


(d) Tensor made by random matrices with elements in $(0, 1)$

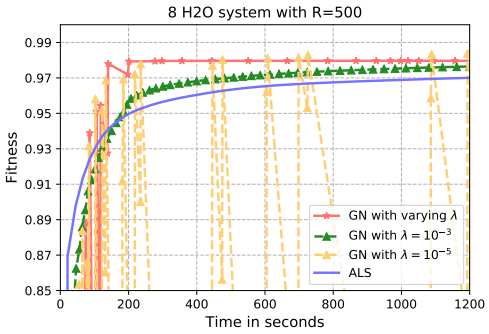


(e) Tensor made by random matrices with elements in $(0, 1)$

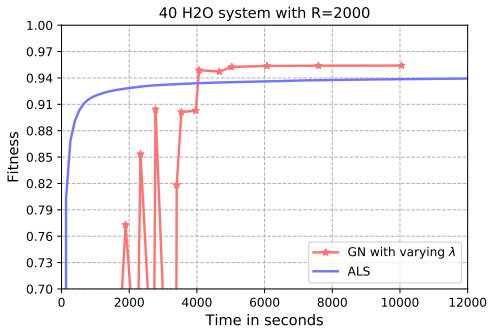
Figure 5.3: Relative residual norm vs time for the CP decomposition of synthetic tensors with different sizes. The results (a) and (b) are collected with the NumPy backend, while (c) and (d) are collected with the Cyclops backend.



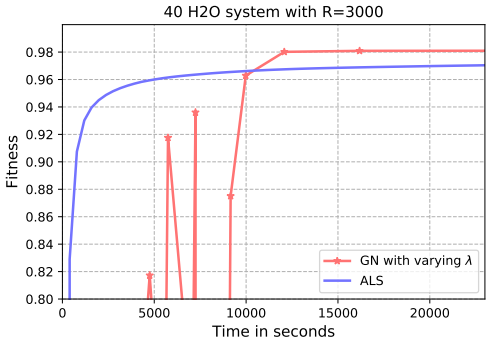
(a) Input tensor size: $339 \times 21 \times 21$, $R = 200$



(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: $4520 \times 280 \times 280$, $R = 3000$

Figure 5.4: Fitness vs time for the CP decomposition of quantum chemistry tensors with different size and rank. The results (a) and (b) are collected with the NumPy backend, while (c) and (d) are collected with the Cyclops backend.

Chapter 6

CONCLUSION

In this thesis, we provide a formulation for Gauss-Newton method for CP decomposition which can leverage the advantage of parallel 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. Our results demonstrate good weak scalability for the Gauss-Newton method for the current implementation and also show room for improvement in scaling as well as speed for the matrix-vector products. Additionally, we propose a regularization scheme for Gauss-Newton method to improve convergence properties without additional cost. We perform extensive experimentation on different kinds of input tensors and compare the convergence and performance of the Gauss-Newton method relative to ALS. We observe that the Gauss-Newton method typically achieves better convergence as well as performance results for both synthetic as well as real-life tensors with high CP rank.

Chapter 7

BIBLIOGRAPHY

[1] E. Acar, D. M. Dunlavy, and T. G. Kolda. A scalable optimization approach for fitting canonical tensor decompositions. *Journal of Chemometrics*, 25(2):67–86, 2011.

[2] A. Anandkumar, R. Ge, D. J. Hsu, S. M. Kakade, and M. Telgarsky. Tensor decompositions for learning latent variable models. *Journal of Machine Learning Research*, 15(1):2773–2832, 2014.

[3] A. Anandkumar, R. Ge, and M. Janzamin. Guaranteed non-orthogonal tensor decomposition via alternating rank-1 updates. *arXiv preprint arXiv:1402.5180*, 2014.

[4] L. Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific J. Math.*, 16(1), 1966.

[5] E. Bailey and S. Aeron. Word embeddings via tensor factorization. *arXiv preprint arXiv:1704.02686*, 2017.

[6] G. Ballard, K. Hayashi, and R. Kannan. Parallel nonnegative CP decomposition of dense tensors. *arXiv preprint arXiv:1806.07985*, 2018.

[7] G. Ballard, N. Knight, and K. Rouse. Communication lower bounds for matricized tensor times Khatri-Rao product. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 557–567. IEEE, 2018.

[8] C. Battaglino, G. Ballard, and T. G. Kolda. A practical randomized CP tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, 39(2):876–901, 2018.

[9] A. R. Benson and G. Ballard. A framework for practical parallel fast matrix multiplication. In *ACM SIGPLAN Notices*, volume 50, pages 42–53. ACM, 2015.

[10] L. S. Blackford, J. Choi, A. Cleary, E. D'Azeuedo, J. Demmel, I. Dhillon, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[11] F. Cong, Q.-H. Lin, L.-D. Kuang, X.-F. Gong, P. Astikainen, and T. Ristaniemi. Tensor decomposition of EEG signals: A brief review. *Journal of neuroscience methods*, 248:59–69, 2015.

[12] K. Hayashi, G. Ballard, J. Jiang, and M. Tobia. Shared memory parallelization of MTTKRP for dense tensors. *arXiv preprint arXiv:1708.08976*, 2017.

[13] F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Studies in Applied Mathematics*, 6(1-4):164–189, 1927.

[14] 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. *The Journal of Chemical Physics*, 137(22):221101, 2012.

[15] F. Hummel, T. Tsatsoulis, and A. Grüneis. Low rank factorization of the Coulomb integrals for periodic coupled cluster theory. *The Journal of chemical physics*, 146(12):124105, 2017.

[16] L. Karlsson, D. Kressner, and A. Uschmajew. Parallel algorithms for tensor completion in the CP format. *Parallel Computing*, 57:222–234, 2016.

[17] O. Kaya and B. Uçar. *Parallel CP decomposition of sparse tensors using dimension trees*. PhD thesis, Inria-Research Centre Grenoble–Rhône-Alpes, 2016.

[18] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[19] N. Li, S. Kindermann, and C. Navasca. Some convergence results on the regularized alternating least-squares method for tensor decomposition. *Linear Algebra and its Applications*, 438(2):796–812, 2013.

[20] L. Ma and E. Solomonik. Accelerating alternating least squares for tensor decomposition by pairwise perturbation. *arXiv preprint arXiv:1811.10573*, 2018.

[21] K. Maruhashi, F. Guo, and C. Faloutsos. Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis. In *2011 International Conference on Advances in Social Networks Analysis and Mining*, pages 203–210. IEEE, 2011.

[22] B. C. Mitchell and D. S. Burdick. Slowly converging PARAFAC sequences: swamps and two-factor degeneracies. *Journal of Chemometrics*, 8(2):155–168, 1994.

[23] D. Mitchell, N. Ye, and H. De Sterck. Nesterov acceleration of alternating least squares for canonical tensor decomposition. *arXiv preprint arXiv:1810.05846*, 2018.

[24] J. J. Moré. The Levenberg-Marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

[25] K. R. Murphy, C. A. Stedmon, D. Graeber, and R. Bro. Fluorescence spectroscopy and multi-way techniques. PARAFAC. *Analytical Methods*, 5(23):6557–6566, 2013.

[26] C. Navasca, L. De Lathauwer, and S. Kindermann. Swamp reducing technique for tensor decomposition. In *2008 16th European Signal Processing Conference*, pages 1–5. IEEE, 2008.

[27] D. Nion and L. De Lathauwer. An enhanced line search scheme for complex-valued tensor decompositions. Application in DS-CDMA. *Signal Processing*, 88(3):749–755, 2008.

[28] P. Paatero. A weighted non-negative least squares algorithm for three-way PARAFAC factor analysis. *Chemometrics and Intelligent Laboratory Systems*, 38(2):223–242, 1997.

[29] V. Pan. *How to Multiply Matrices Faster*. Springer-Verlag New York, Inc., New York, NY, USA, 1984.

[30] A.-H. Phan, P. Tichavskỳ, and A. Cichocki. Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations. *IEEE Transactions on Signal Processing*, 61(19):4834–4846, 2013.

[31] A.-H. Phan, P. Tichavsky, and A. Cichocki. Low complexity damped Gauss-Newton algorithms for CANDECOMP/PARAFAC. *SIAM Journal on Matrix Analysis and Applications*, 34(1):126–147, 2013.

[32] M. Rajih, P. Comon, and R. A. Harshman. Enhanced line search: A novel method to accelerate PARAFAC. *SIAM journal on matrix analysis and applications*, 30(3):1128–1147, 2008.

[33] 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 Journal on Scientific Computing*, 36(5):C453–C479, 2014.

[34] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582.

[35] N. Singh, L. Ma, H. Yang, and E. Solomonik. Comparison of accuracy and scalability of gauss-newton and alternating least squares for cp decomposition. *arXiv preprint arXiv:1910.12331*, 2019.

[36] A. Smirnov. The bilinear complexity and practical algorithms for matrix multiplication. *Computational Mathematics and Mathematical Physics*, 53(12):1781–1795, 2013.

[37] E. Solomonik, D. Matthews, J. R. Hammond, J. F. Stanton, and J. Demmel. A massively parallel tensor contraction framework for coupled-cluster computations. *Journal of Parallel and Distributed Computing*, 74(12):3176–3190, 2014.

[38] 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 Journal on Optimization*, 23(2):695–720, 2013.

[39] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[40] 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, et al. PySCF: The Python-based simulations of chemistry framework. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 8(1):e1340, 2018.

[41] 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. *The Journal of chemical physics*, 146(20):204110, 2017.

[42] P. Tichavskỳ, A. H. Phan, and A. Cichocki. A further improvement of a fast damped Gauss-Newton algorithm for CANDECOMP-PARAFAC tensor decomposition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5964–5968. IEEE, 2013.

[43] G. Tomasi and R. Bro. PARAFAC and missing values. *Chemometrics and Intelligent Laboratory Systems*, 75(2):163–180, 2005.

[44] G. Tomasi and R. Bro. A comparison of algorithms for fitting the PARAFAC model. *Computational Statistics & Data Analysis*, 50(7):1700–1734, 2006.

[45] 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. *Computing in Science and Engineering*, 16(05):62–74, sep 2014.

[46] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[47] A. Uschmajew. Local convergence of the alternating least squares algorithm for canonical tensor approximation. *SIAM Journal on Matrix Analysis and Applications*, 33(2):639–652, 2012.

[48] N. Vannieuwenhoven, K. Meerbergen, and R. Vandebril. Computing the gradient in optimization algorithms for the CP decomposition in constant memory through tensor blocking. *SIAM Journal on Scientific Computing*, 37(3):C415–C438, 2015.