Parallel algorithms for computing low rank decompositions of matrices and tensors

Matthias Beaupère

Sorbonnes Université
Inria

Algorithmes parallèles pour le calcul des décompositions de rang faible des matrices et tenseurs

par Matthias Beaupère

Thèse de doctorat en mathématiques appliquées
dirigée par Laura Grigori

École doctorale Sciences Mathématiques de Paris Centre
Laboratoire Jacques-Louis Lions

Présentée et soutenue publiquement le 23 mars 2023

devant un jury composé de

Julien Langou	University of Colorado Denver	Rapporteur
Olivier Beaumont	Université Bordeaux 1	Rapporteur
Laura Grigori	Inria Paris	Directrice de thèse
Bora Uçar	ENS Lyon	Examinateur
Grey Ballard	Wake Forest University	Examinateur
Virginie Ehrlacher	École des Ponts ParisTech	Examinatrice
Frédéric Nataf	Sorbonne-Université	Examinateur

Sorbonnes Université
Inria

Parallel algorithms for computing low rank decompositions of matrices and tensors

by Matthias Beaupère

Ph.D thesis in applied mathematics
directed by Laura Grigori

Doctoral School Sciences Mathématiques de Paris Centre
Laboratoire Jacques-Louis Lions

Publicly defended on March 23rd 2023

with a jury composed of

Julien Langou	University of Colorado Denver	Reviewer
Olivier Beaumont	Université Bordeaux 1	Reviewer
Laura Grigori	Inria Paris	Ph.D Advisor
Bora Uçar	ENS Lyon	Examiner
Grey Ballard	Wake Forest University	Examiner
Virginie Ehrlacher	École des Ponts ParisTech	Examiner
Frédéric Nataf	Sorbonne-Université	Examiner

Acknowledgment

The preparation of my doctorate thesis lasted three and a half years in Paris. I would like to thank all people who accompagnied me during this time.

To those who warmly welcomed me on the third floor, Fabien Raphel, Mathieu Mezaches, Thibault Cimic, Igor Chollet, Antoine Lesieur, Jan Papez, Liudi Lu, Van Thanh Nguyen, thank you for setting up such a nice environment. To those who joined afterwards, Siwar Baddreddine, David Frenkiel, Daniel Torres, Édouard Timsit, Niels Guilbert, Mathieu Rigal, Apolline El Baz, Chourouk El Hassianeh, Jean-Guillaume de Damas, Victor Lederer, Amélia Ferhat, Haibo Liu, Juliette Dubois, Norbert Tognon, Oleg Balabanov, thank you for your joy every day. To my longest office mate Suraj Kumar, thank you for helping me in every way, and for your support throughout the thesis. Thank you to all researchers and staff from Inria, especially Sever Hirstoaga, Frédéric Nataf, Laurence Bourcier, Julien Guieu, Irène Vignon-Clémentel, Damiano Lombardi, from the LJLL, especially Mi-Song Dupuy, Xavier Claeys, Corentin Lacombe, Jean-François Venuti, and from the Cermics, especially Éric Cancès and Virginie Ehrlacher.

I am grateful to the reviewers of my thesis, as well as all jury members, for their fruitful feedbacks.

Thank you to my advisor Laura Grigori, you trusted me and guided me with great wisdom throughout the four years working together.

Thank you to my wife Marie. You followed me in this adventure and I can never be grateful enough for your support.

Contents

Résumé en français

Les matrices et tenseurs figurent parmi les outils les plus répandus de représentation et d'exploitation de l'information. Certaines sources produisent de grandes quantités de données, et déterminer l'information précise présente dans ces données est primordial au sein d'un grand nombre de domaines. D'une autre façon, les simulations numériques produisent parfois des objets très grands comparés à la quantité d'information réelle présente dans le système simulé, comprimer cette information permettrait alors d'effectuer une simulation efficace et peu coûteuse.

Pour mettre en place cette simplification, nous considérons les méthodes d'algèbre linéaire telles que la décomposition en valeurs singulières (SVD) ainsi que la décomposition QR. Elles s'appliquent sur tout type de matrices et fournissent des indices sur l'organisation de l'information, ainsi que des outils efficaces pour les manipuler, comme l'approximation de rang faible. L'approximation de rang faible permet de compresser des matrices de grande taille dans un format réduit, avec ou sans perte (erreur). Elle représente la matrice d'entrée comme un produit de matrices plus petites, pouvant être vues comme la projection de la matrice dans un sous-espace, selon deux principes: le sous-espace est de faible dimension; l'information perdue lors de la projection est moindre. La perte d'information correspond ici à l'erreur entre la matrice originale et son approximation. Ces deux principes antagonistes conduisent au double problème de minimisation sous-jacent: d'une part en fixant la taille du sous-espace et en minimisant l'erreur; d'autre part en fixant une contrainte d'erreur et en minimisant la taille du sous-espace.

Les approximations de rang faible sont appliquées dans nombre de domaines qui nécessitent de récupérer une information utile dans une grande quantité de données d'entrée (parfois redondante). L'étude des approximations de rang faible a débuté en psychométrique [43], où les données d'entrée étaient issues de l'étude d'un groupe d'individus. Bien qu'au début ces individus étaient référencés selon un unique attribut, le nombre d'attributs a par la suite augmenté, déplaçant ainsi l'étude dans le domaine tensoriel [111]. Les approximations de rang faible furent ensuite appliquées aux sciences des données, dont la fouilles de données [45, 9], les systèmes de recommandation [39, 121, 29], la recherche internet [77, 73], les base de données et l'indexation de corpus de documents [1, 98, 19], le partitionnement de données [49, 48, 40], le partitionnement de graphes [14], et la regression linéaire [89, 120]. On trouve également des approximations de rang faible dans des applications physiques comme l'astronomie [44], la géophysique [55, 108], la science du climat [122], le traitement du signal [113] ainsi que la dynamique des fluides [38]. Ces questions apparaissent aussi dans le domaine de la compression d'image et de vidéo [104], et dans le contexte de l'imagerie

médicale [46, 72].

Si ces méthodes sont facilement applicables à une échelle raisonnable au moyen d'un ordinateur moderne, elle deviennent difficiles à utiliser sur des matrices de grande tailles, en particulier dépassant l'espace mémoire de l'ordinateur. De nos jours, il est de plus en plus courant de manipuler de telles matrices en sciences des données ou en simulation, et un grand nombre de méthodes des deux dernières décennies permettent de s'y adapter. De telles méthodes suggèrent la division de la matrice (ou du tenseur) sur plusieurs processeurs associés en centre de calcul, et le calcul de l'approximation en parallèle. Le but est alors d'obtenir avec plusieurs ordinateurs une décomposition qui ressemble au mieux à celle calculée par un unique ordinateur. L'échange d'information est autorisé entre les nœuds, mais devrait rester minimal car la communication est la principale cause de ralentissement lors du passage à grande échelle.

Cette thèse vise à présenter de nouvelles méthodes parallèles diminuant plus encore le coût des calculs et des communications. Ces nouvelles méthodes permettent de manipuler des matrices de grande taille plus efficacement, économisant de l'espace de stockage et du temps. De plus, elles résument l'information contenue dans une matrice, fournissant une vue interne et découvrant la structure des données. Pour beaucoup d'applications, les données sont fonction de nombreuses variables, alors représentées par un tableau multidimensionnel, un tenseur. Ce document développe cet aspect en orientant une partie de l'étude vers le domaine tensoriel.

Ce travail traite principalement des approximations de rang faible obtenues par décomposition QR. Pour cela, on utilise la décomposition QR avec pivotement des colonnes (QRCP) [21]. Une implémentation couramment utilisée de QRCP est disponible dans la routine `dgeqpf` de LAPACK (Linear Algebra Package [7]), mais la dernière version est dans la routine `dgeqp3` [cf. 99]. Cette implémentation écrite en Fortran est inclue dans la plupart des langages et outils haut niveau tel que Matlab, Python et Julia. Cet algorithme a une implémentation parallèle dans le routine `pdgeqpf` de ScaLAPACK (Scalable Linear Algebra Package [18]). Cet algorithme produit la décomposition QRCP complète, cependant seule une petite partie est nécessaire pour l'approximation. Cette partie réduite, formant une décomposition partielle nommée QRCP tronquée, est le sujet de nombreuses discussions exposant plusieurs stratégies de pivotement [106, 25, 26, 66, 59, 96]. Nous retiendrons ici la méthode *strong rank revealing QR* [59], dont la stratégie de pivotement garantie des bornes sur l'erreur d'approximation. Cet algorithme fut étendu pour prendre en compte le pivotement indépendant de groupes séparés de colonnes, afin de permettre la parallélisation, et la borne d'erreur fut adaptée en conséquence, conduisant à l'algorithme CARRQR [36]. Dans ce contexte, nous généralisons CARRQR afin de pouvoir non seulement déterminer indépendamment des pivots sur des groupes de colonnes, mais également sur des groupes de lignes, permettant ainsi d'agir sur des sous-matrices; nous évaluons en pratique l'erreur d'approximation (par rapport à la borne théorique), ainsi que l'influence de la structure de la matrice d'entrée; nous étudions comment cet algorithme se comporte à grande échelle, en comparaison avec l'existant. D'un autre point de vue, la randomisation a connu un gain d'attractivité au cours de la dernière décennie, et plusieurs articles traite du couplage entre QRCP tronquée et l'échantillonnage statistique visant à réduire le nombre de lignes [42, 118, 86]. La randomisation est effectuée dans ce cas avec un échantillonnage gaussien. À ce propos, nous considérons une

autre transformation aléatoire, *subsampled randomized Hadamard transform*, et nous étudions si cet transformation est efficace à grande échelle. En déplaçant l'étude dans le domaine tensoriel, on étudie l'approximation de rang (multilinéaire) faible obtenue avec la décomposition de Tucker [111] et implémentée en parallèle dans la bibliothèque TuckerMPI [13]. La décomposition de Tucker utilise des approximations de rang faible, effectuées en général via des décompositions SVD tronquées. Récemment furent conçues plusieurs variantes intégrant la randomisation [47, 97, 90, 91, 28]. On se pose la question de substituer QRCP tronquée parallèle à SVD tronquée pour calculer la décomposition de Tucker, et de comparer l'erreur et le temps d'exécution avec la décomposition Tucker parallèle.

Ce document répondra à ces problèmes de recherche dans l'ordre suivant.

Le chapitre 1 donne les éléments théoriques d'algèbre linéaire et multilinéaire, et du calcul à grande échelle. Le lecteur trouvera dans ce chapitre les notations générales, les théorèmes et algorithmes liés aux décompositions QR et SVD, ainsi que les bornes d'erreur usuelles. De plus, le formalisme tensoriel est présenté, accompagné d'un exemple d'application pratique qui motive les différentes méthodes présentées dans ce travail. Cette application consiste en une simulation moléculaire, représentée par des opérations tensorielles, exposant différentes étapes où les approximations de rang faible de matrices et de tenseurs sont cruciales pour une fiabilité à grande échelle. La dernière section donne un tour d'horizon de l'environnement de calcul disponible pour les expériences, en termes matériels et logiciels.

Le chapitre 2 présente la parallélisation d'une technique alternative de randomisation appelée *subsampled randomized Hadamard transform* (SRHT). Destinée à la même utilisation que l'échantillonnage gaussien, avec une précision similaire elle produit de meilleurs temps d'exécution. *Block SRHT* est notre version distribuée de cette méthode, pouvant être appliquée sur des matrices de grande taille. De plus, *block SRHT* est utilisée pour calculer l'approximation de Nyström, une décomposition de rang faible pour les matrices positives semi-définies.

Le chapitre 3 généralise CARRQR pour un partitionnement quelconque de colonnes et de lignes de la matrice d'entrée, conduisant au nouvel algorithme QR avec tournoi par colonne (QRTP). QRTP calcule le vecteur de pivots d'une matrice distribuée quelconque à l'aide de décompositions QRCP tronquées locales. La borne de la méthode *strong RRQR* est adaptée pour offrir des garanties dans la nouvelle configuration, et l'implémentation logicielle est détaillée. On analyse le comportement de cette algorithme selon le profil des valeurs singulières de la matrice d'entrée, et on fournit des expériences de passage à l'échelle sur un grande nombre de nœuds de calcul, en comparant avec l'alternative QRCP randomisé.

Le chapitre 4 présente l'algorithme *high order QRTP* (HOQRTP), substituant la decomposition QRCP tronquée à la SVD tronquée dans la décomposition de Tucker. Il utilise notre algorithme QRTP pour calculer efficacement chaque approximation de rang faible en parallèle. Nous expliquons comment QRTP peut être appliqué dans chaque dimension sans mouvement de données, et fournissons des bornes d'erreur théoriques. Nous intégrons également les idées liées à l'algorithme *sequentially truncated HOSVD*, formant un nouvel algorithme appelé *sequentially truncated HOQRTP* (ST-HOQRTP). Cet algorithme est comparé à TuckerMPI sur des tenseurs de grande taille.

Cette thèse à mené aux publications suivantes.

Accepté Matthias Beaupère, David Frenkiel, Laura Grigori. *Higher-Order QR with Tournament Pivoting for Tensor Compression.* À paraître dans le *SIAM Journal on Matrix Analysis and Applications.* Épreuve: `https://hal.inria.fr/hal-03079236`.

Soumis Matthias Beaupère, Laura Grigori. *Communication avoiding low rank approximation based on QR with tournament pivoting.* 2022. Épreuve: `https://hal.inria.fr/hal-02947991`.

Soumis Oleg Balabanov, Matthias Beaupere, Laura Grigori, Victor Lederer. *Block subsampled randomized Hadamard transform for low-rank approximation on distributed architectures.* 2022. Épreuve: `https://arxiv.org/abs/2210.11295`.

Introduction

Matrices and tensors are amongst the most common tools to represent and exploit information. Some sources produce large quantities of data, and analyzing the information provided by this data is important in many domains. Additionally, numerical simulations sometimes produce objects that are very large compared to the actual quantity of information residing in the simulated system, and compressing these objects yields efficient and cost-saving simulations.

To achieve this compression, we consider linear algebra methods such as the singular value decomposition (SVD) and the QR decomposition. They apply on any matrix and provide hints on the information to compress as well as efficient tools to manipulate it, such as the low rank approximation. The low rank approximation compresses large matrices into a smaller format, with or without loss of information (error). It represents the input matrix as a product of factor matrices of smaller dimension, equivalently seen as projecting the matrix into a subspace with two principles: the subspace is small; little information is lost from the matrix—the loss of information is quantified as an error between the matrix and its approximation. This can be formulated as a double minimization problem, tackled from two possible angles: either by fixing the subspace size and minimizing the error, or by fixing an error constraint and minimizing the subspace size.

Applications of low rank approximation are multiple, in many domains where valuable information is sought from a large amount of (often redundant) input data. The study of low rank approximation began in the field of psychometrics [43], where the input data comes from studying a group of individuals. If, at first, the subjects were described with a single feature, the dimension increased afterwards and it became relevant to study the problem in tensor form [23]. Low rank approximation was later applied to data sciences including general data mining [45, 9], recommendation systems [39, 121, 29], web search [77, 73], database and document corpus indexing [1, 98, 19], clustering [49, 48, 40], graph partitioning [14], and linear regression [89, 120]. Low rank approximations are also used in physical applications such as astronomy [44], geophysics [55, 108], climate science [122], signal processing [113] and fluid dynamics [38]. The same question arises also in the field of image and video compression [104], and in the context of medical imagery [46, 72].

If these methods are quite easy to use at reasonable scale with a modern laptop, they can be more difficult to apply at larger scale especially when the matrix does not fit entirely in the laptop's memory. Such large matrices occur more and more in modern simulations and data science, and various methods were presented in the last two decades addressing this problem. These methods propose to divide the matrix (or tensor) across several computers, associated

in a computing cluster, and perform the decomposition in parallel. The goal is to obtain, with multiple computers, a decomposition that resembles as much as possible the decomposition obtained on a single computer. Communication is allowed between the nodes, but should be kept minimal, as it is the main factor of inefficiency at large scale (see e.g. [15]).

This thesis introduces novel parallel methods, further decreasing communication and computational cost with respect to existing algorithms. These new methods enable to manipulate very large matrices more efficiently, saving storage and time. As an additional benefit, they summarize the information contained in matrices, unveiling mathematical properties such as the column and row spaces of the matrix. For many applications, the data is a function of many variables, hence represented as a multidimensional array, a tensor. This document addresses this aspect by directing part of the analysis toward the higher dimension domain.

This work primarily focuses on low rank approximations obtained with the QR decomposition. To this end, we use the QR decomposition with column pivoting (QRCP) [21]. A broadly used implementation of QRCP is available as the routine `dgeqpf` from LAPACK (Linear Algebra Package [7]), but the recommended version is routine `dgeqp3` [see 99]. This implementation written in Fortran is included in most modern high-level languages and tools such as Matlab, Python and Julia. This algorithm has a parallel implementation in routine `pdgeqpf` of ScaLAPACK (Scalable Linear Algebra Package [18]). This algorithm provides the complete QRCP decomposition, while a small part is sufficient for the approximation. This reduced part, forming a partial decomposition named truncated QRCP, is the topic of many discussions, introducing different pivoting strategies [106, 25, 26, 66, 59, 96]. We retain here the strong rank revealing QR [59], where the pivoting strategy guarantees bounds on the approximation error. This algorithm was extended to consider independent pivoting on separate groups of columns of the matrix, with parallelism in mind, and the bound was accordingly adapted, leading to the communication avoiding rank revealing QR (CARRQR) algorithm [36]. In this context, we extend CARRQR to perform independent pivoting not only on groups of columns, but on submatrices, separating rows as well; we study the practical value of the error (compared to the bound) and its behavior depending on the singular value profile of the input matrix; we investigate the practical scalability of this algorithm, and compare it with existing ones. From another point of view, randomization has gained a lot of attention in the last decade, and various papers focused on coupling the parallel truncated QRCP with sampling to reduce the number of rows [42, 86, 118]. The randomization is performed here with Gaussian sampling. To follow this direction, we consider a different random transform, the subsampled randomized Hadamard transform. We introduce a parallel version of this transform and study its scalability. Translating this analysis in the tensor domain, we focus on the low multilinear rank approximation obtained with the Tucker decomposition [111], implemented in parallel by the TuckerMPI library [13]. The Tucker decomposition uses matrix low rank approximations, generally performed with a truncated SVD decomposition. Recently, several variants were designed incorporating randomization [47, 97, 90, 91, 28]. We address the question of substituting a parallel truncated QRCP to the truncated SVD to compute the Tucker decomposition, and compare the obtained error and speedup with the parallel Tucker decomposition.

These research problems are addressed in this document in the following order.

Chapter 1 provides the background information on linear and multilinear algebra, and large scale computing. The reader can find in this chapter the general notations, theorems and algorithms related to the SVD and QR decompositions as well as the error bounds from the literature. Additionally, the formalism of tensors is given along with a typical application of the material produced in this work. It presents molecular simulation from a tensor point of view, exposing different steps where matrix and tensor low rank approximations are key to scalability and accuracy. In the last section, the computing environment landscape available for the experiments is presented, both in terms of software and hardware.

Chapter 2 presents the parallelization of an alternative randomization technique called subsampled randomized Hadamard transform (SRHT). Intended for the same use as Gaussian sampling, it exhibits a similar accuracy yet a better speedup. Block SRHT is our distributed version of the method, such that it can be applied on large matrices. In this chapter, it is compared to the Gaussian sampling with a theoretical error estimate and numerical experiments. In addition, block SRHT is used to compute the Nyström approximation, a low rank decomposition for symmetric positive semi-definite matrices.

Chapter 3 generalizes CARRQR to any row and column partitioning of the input matrix, leading to a new algorithm called QR with tournament pivoting (QRTP). QRTP computes the column pivoting vector of any distributed matrix by means of local truncated QRCP decompositions. The strong RRQR error bound is adapted to provide guarantees on the new configuration, and the practical implementation is detailed. We analyze the behavior of this algorithm depending on the singular value profile of the input matrix, and provide scalability experiments on a large number of nodes, comparing with the alternative randomized QRCP algorithm.

Chapter 4 presents the higher order QRTP (HOQRTP) algorithm, substituting the truncated QRCP decomposition to the truncated SVD in the Tucker decomposition. It uses our QRTP algorithm to compute each low rank approximation efficiently in parallel. We explain how QRTP can be applied in each dimension without data mouvements, and provide theoretical error bounds. We also incorporate the sequentially truncated approach to speed up the computation, constituting a new version called sequentially truncated HOQRTP (ST-HOQRTP). This algorithm is compared to the TuckerMPI implementation on large scale tensors.

This thesis led to the following publications.

Accepted Matthias Beaupère, David Frenkiel, Laura Grigori. Higher-Order QR with Tournament Pivoting for Tensor Compression. To appear in SIAM Journal on Matrix Analysis and Applications. Preprint `https://hal.inria.fr/hal-03079236`.

Submitted Matthias Beaupère, Laura Grigori. Communication avoiding low rank approximation based on QR with tournament pivoting. 2022. Preprint `https://hal.inria.fr/hal-02947991`.

Submitted Oleg Balabanov, Matthias Beaupere, Laura Grigori, Victor Lederer. Block subsampled randomized Hadamard transform for low-rank approximation on distributed architectures. 2022. Preprint `https://arxiv.org/abs/2210.11295`.

Chapter 1

Background on numerical linear and multilinear algebra

The present document introduces different methods to approximate large matrices and tensors. To prepare the introduction of each method, this chapter reviews basic notions surrounding them. It first provides the general definitions of linear and multilinear algebra, including related work on the topic. In addition, we present a typical application motivating the use of the methods presented in the following chapters. Finally, practical information on how test cases are generated is given, as well as the usage of high performance computing in the field of linear algebra.

1.1 Notation

Throughout this work we use the following notations.

A matrix is denoted $\mathbf{A}$, a vector $\mathbf{x}$, and a tensor $\mathcal{A}$. $\mathbb{R}^{m\times n}$ is the set of matrices having m rows and n columns. For any matrix $\mathbf{A}$, $\mathbf{A}^\intercal$ is the transpose of $\mathbf{A}$ and $\mathbf{A}^+$ is the pseudo-inverse of $\mathbf{A}$

For any matrix $\mathbf{A}$, $\mathbf{A}[i:j,k:l]$ denotes the submatrix containing rows i to j and columns k to l of $\mathbf{A}$, $\mathbf{A}[:,k:l]$ denotes the submatrix containing columns k to l of $\mathbf{A}$, $\mathbf{A}[:,:l]$ denotes the submatrix containing the l first columns of $\mathbf{A}$, and $\mathbf{A}[:,j]$ denotes jth column of $\mathbf{A}$. For any matrix $\mathbf{A}$, and a set of indices I, $\mathbf{A}[:,I]$ denotes the submatrix containing columns of indices I of $\mathbf{A}$.

For any matrix $\mathbf{A}\in\mathbb{R}^{m\times n}$, we define

$$\omega_i(\mathbf{A}) = \frac{1}{\|\mathbf{A}^{-1}[i,:]\|}, \quad \gamma_j(\mathbf{A}) = \|\mathbf{A}[:,j]\|\,.$$

Additionally, $\mathbf{O}$ denotes a matrix containing only zeroes, and $\mathbf{I}$ denotes the identity matrix.

For any matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ having the same number of rows, $[\mathbf{A}_1;\mathbf{A}_2] = \begin{bmatrix}\mathbf{A}_1\\ \mathbf{A}_2\end{bmatrix}$ denotes the vertical concatenation of $\mathbf{A}_1$ and $\mathbf{A}_2$. Similarly, for a common number of columns, $\begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2\end{bmatrix}$ denotes the horizontal concatenation of $\mathbf{A}_1$ and $\mathbf{A}_2$.

For any matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\sigma_k(\mathbf{A})$ denotes the kth largest singular value of $\mathbf{A}$. $\|\mathbf{A}\|_2 = \max_{\mathbf{x} \in \mathbb{R}^n} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|}$, is the ℓ_2 operator norm of $\mathbf{A}$, also called the 2-norm, or spectral norm. It corresponds to the maximum singular value of $\mathbf{A}$ (see below the definition of the singular value decomposition). $\|\mathbf{A}\|_F = \sqrt{\sum_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}} a_{ij}^2}$, where a_{ij} is a coefficient of $\mathbf{A}$, is the Frobenius norm of $\mathbf{A}$. $\|\mathbf{A}\|_* = tr(\mathbf{A})$, where $tr()$ denotes the trace, is the nuclear norm, also called the trace norm, of $\mathbf{A}$.

For any random variable X, $\mathbb{E}[X]$ is the expected value of X.

1.2 Matrix low-rank approximation

After defining the low-rank property for a matrix, we give several decompositions leading to a low-rank approximation.

1.2.1 Projections

Orthogonal projections are abundantly used throughout the document. As an introduction, the following gives a simple presentation of orthogonal projections.

An orthogonal matrix $\mathbf{Q}$ is square and obeys the rule $\mathbf{Q}\mathbf{Q}^\intercal = \mathbf{Q}^\intercal\mathbf{Q} = \mathbf{I}$. Its columns form a set of linearly independent vectors, orthogonal to one another. A partial orthogonal matrix $\mathbf{Q}_1$, sometimes also named shortly orthogonal through misuse of language, contains a column subset from an orthogonal matrix (there exists $\mathbf{Q}$ and $\mathbf{Q}_2$ such that $\mathbf{Q} = \begin{bmatrix}\mathbf{Q}_1 & \mathbf{Q}_2\end{bmatrix}$ is orthogonal), hence also has orthogonal columns to one another. A partial orthogonal matrix obeys the rule $\mathbf{Q}_1^\intercal\mathbf{Q}_1 = \mathbf{I}$, yet $\mathbf{Q}_1\mathbf{Q}_1^\intercal = \mathbf{P} \neq \mathbf{I}$. In particular, $\mathbf{Q}_1^\intercal$ operates an orthogonal change of basis to a smaller space.

$\mathbf{P}$ belongs to the set of projection matrices: it obeys the rule $\mathbf{P}\mathbf{P} = \mathbf{P}$. In particular, while defined on a vector space E, its range lies in the vector subspace $\text{span}(\mathbf{Q}_1) \subset E$ generated by the columns of $\mathbf{Q}_1$.

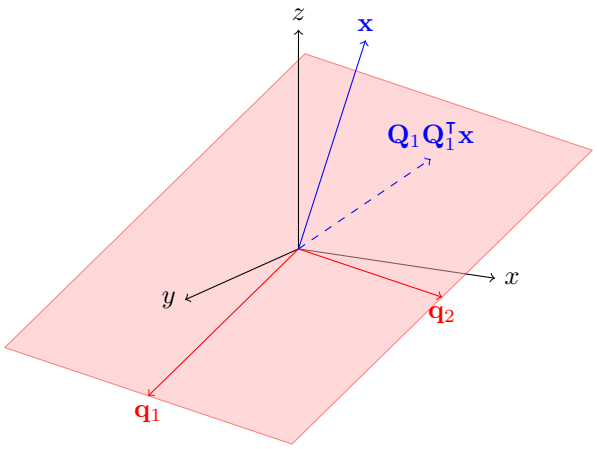


Figure 1.1: Projection plan, image of the projector $\mathbf{P}$ in $\mathbb{R}^3$, along with one vector $\mathbf{x}$ of $\mathbb{R}^3$ and its orthogonal projection.

As an example, consider a randomly chosen 3×3 matrix

$$\mathbf{Q} = \begin{bmatrix} -0.306899 & 0.935221 & -0.176561 \\ -0.792075 & -0.353832 & -0.497414 \\ -0.527665 & -0.0128059 & 0.849356 \end{bmatrix}$$

and select two columns to form a partial orthogonal matrix

$$\mathbf{Q}_1 = \begin{bmatrix} -0.306899 & 0.935221 \\ -0.792075 & -0.353832 \\ -0.527665 & -0.0128059 \end{bmatrix}.$$

The transpose $\mathbf{Q}_1^\intercal$ applied to any vector of $\mathbb{R}^3$ writes its orthogonal projection on $\mathrm{Span}(\mathbf{Q}_1)$ as a linear combination of these two columns, its image lies in $\mathbb{R}^2$. The projection matrix $\mathbf{P} = \mathbf{Q}_1\mathbf{Q}_1^\intercal$ projects any vector of $\mathbb{R}^3$ onto the plane generated from the column vectors of $\mathbf{Q}_1$. The image is still in $\mathbb{R}^3$ but lies in a vector subspace of dimension 2. This plane is represented in Figure 1.1. The vector $\mathbf{x}$ in Figure 1.1 is a random vector of $\mathbb{R}^3$, applying $\mathbf{Q}_1\mathbf{Q}_1^\intercal$ comes down to projecting it on the plane.

1.2.2 Preliminary definitions

The *singular value decomposition* (SVD) of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is defined as

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\intercal,$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices composed of the left and right singular vectors, and $\mathbf{\Sigma}$ is a diagonal matrix composed of the singular values, the set of which we call the spectrum of $\mathbf{A}$. This decomposition is essentially unique under the constraint that the coefficients of $\mathbf{\Sigma}$ are positive and sorted.

The *rank* k of $\mathbf{A} \in \mathbb{R}^{m \times n}$ is the number of non-zeros on the diagonal of $\mathbf{\Sigma}$. If $\mathbf{\Sigma}$ contains any zero, i.e. if $k < \min(m, n)$, the matrix $\mathbf{A}$ is called singular, or rank-deficient. In practice, in linear algebra computations, matrices that are expected to be singular are not necessarily so, due to round-off errors and other approximations. Thus, we rather study the ratio $\frac{\sigma_i(\mathbf{A})}{\sigma_1(\mathbf{A})}$. We call numerical rank the index i such that for a given threshold ε, $\forall j > i$, $\frac{\sigma_j(\mathbf{A})}{\sigma_1(\mathbf{A})} < \varepsilon$.

A matrix $\mathbf{A}$ is of *low rank* if it has rank k, and $k \ll \min(m, n)$. It typically manifests itself as rapidly decaying singular values, meaning that the coefficients on the diagonal of $\mathbf{\Sigma}$ decrease fast enough. A low-rank matrix is numerically rank deficient, but the converse is not necessarily true: a numerically rank-deficient matrix is not necessarily low-rank.

The low-rank property is a good indicator that a matrix can be compressed, meaning that a lower amount of coefficients is sufficient to represent the matrix. Such a matrix can be stored in an alternative format, decreasing the storage and computational cost.

Other indicators are also useful to consider. The *condition number* of $\mathbf{A}$ defined as $\kappa(\mathbf{A}) = \sigma_{\max}(\mathbf{A})/\sigma_{\min}(\mathbf{A})$ describes the spectrum width of $\mathbf{A}$. The *gap* between two consecutive singular values i and $i+1$ is $\sigma_i(\mathbf{A})/\sigma_{i+1}(\mathbf{A})$. If there exists a gap of several orders of magnitudes at index k in the spectrum of $\mathbf{A}$, then $\mathbf{A}$ can be compressed with an accuracy depending on σ_{k+1}, and if additionally $k \ll \min(m, n)$, then $\mathbf{A}$ has a low numerical rank. We also define

the *tail* of $\mathbf{A}$ from index k as $\sqrt{\sum_{i=k+1}^{n} \sigma_i^2(\mathbf{A})}$. The tail often helps to measure the information lost (in Frobenius norm) when compressing a matrix at rank k.

The following recalls various methods from the literature to compute the compressed format of a dense matrix.

1.2.3 Truncated Singular Value Decomposition

The truncated SVD (see Figure 1.2) is defined as $\mathbf{A}_{opt,k} = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^\intercal$, where $\mathbf{\Sigma}_k \in \mathbb{R}^{k \times k}$ is a diagonal matrix formed by the k leading singular values of $\mathbf{A}$, $\sigma_1(\mathbf{A}), \ldots, \sigma_k(\mathbf{A})$, $\mathbf{U}_k \in \mathbb{R}^{m \times k}$ and $\mathbf{V}_k \in \mathbb{R}^{n \times k}$ are the corresponding left and right singular vectors, respectively. The truncated SVD provides the best low rank approximation in terms of Frobenius norm and 2-norm [43], with $\|\mathbf{A} - \mathbf{A}_{opt,k}\|_2 = \sigma_{k+1}(\mathbf{A})$ and $\|\mathbf{A} - \mathbf{A}_{opt,k}\|_F = \sqrt{\sum_{i=k+1}^{\min(m,n)} \sigma_i^2(\mathbf{A})}$.

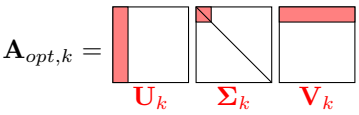


Figure 1.2: Truncated singular value decomposition.

This method is the reference to evaluate the performance of a compression method. Nevertheless for large matrices the truncated SVD is expensive to compute. The reasons are the following: one often has to compute the full SVD to obtain the truncated version; the sequential implementation of the accurate SVD has a complexity of $O(n^3)$ where n is the number of rows of a square matrix [54, Fig. 8.6.1]; the SVD is not easy to parallelize.

1.2.4 QR, QRCP and strong RRQR

The QR decomposition of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is denoted by

$$\mathbf{A} = \mathbf{QR},$$

where $\mathbf{Q}$ is an orthogonal matrix, and $\mathbf{R}$ is an upper triangular matrix. These matrices are typically computed by applying the Gram-Schmidt process to the columns of $\mathbf{A}$, alternatively using Householder reflectors [68] or Givens rotations [53].

If $\mathbf{A}$ is a tall matrix, we define an alternative *thin* QR decomposition $\mathbf{A} = \mathbf{Q}_1\mathbf{R}_1$ where $\mathbf{Q}_1 \in \mathbb{R}^{m \times n}$ is tall with orthonormal columns and $\mathbf{R}_1 \in \mathbb{R}^{n \times n}$ is square, upper-triangular and relatively small.

Throughout this work orthogonal matrices such as $\mathbf{Q}$ are typically split such that $\mathbf{Q} = \begin{bmatrix}\mathbf{Q}_1 & \mathbf{Q}_2\end{bmatrix}$, as mentioned in Section 1.2.1. Recall the behavior of partial orthogonal matrices: $\mathbf{Q}_1^\intercal \mathbf{Q}_1 = \mathbf{I}$ is a small identity matrix and $\mathbf{Q}_1\mathbf{Q}_1^\intercal$ is a projection matrix. Theorem 1 and Theorem 2 provide useful inequalities on the behavior of the singular value, the former for a submatrix of $\mathbf{A}$, the latter for a matrix projected with $\mathbf{Q}_1$ or $\mathbf{Q}_2$.

Theorem 1 (Interlacing singular values [109])**.** *Let* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *and* $\mathbf{B} \in \mathbb{R}^{p \times q}$ *a submatrix of* $\mathbf{A}$*. Then for* $1 \leq i \leq \min(p, q)$

$$\sigma_{\min(p,q)-i+1}(\mathbf{A}) \leq \sigma_i(\mathbf{B}) \leq \sigma_i(\mathbf{A})$$

See the proof in the reference.

Theorem 2. *Let* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *and* $\mathbf{Q} \in \mathbb{R}^{m\times p}$ *be two matrices such that* $p \leq m$ *and* $\mathbf{Q}$ *has orthonormal columns. Then for* $1 \leq i \leq \min(p, n)$

$$\sigma_i(\mathbf{Q}^\intercal \mathbf{A}) \leq \sigma_i(\mathbf{A}).$$

Proof. We use the following well-known singular value inequalities. Given two matrices $\mathbf{A}, \mathbf{Q} \in \mathbb{R}^{m\times n}$, using [67, Thm. 3.3.16(d)] restricted to the real case, we have that for $1 \leq i \leq min(m, n)$, $\sigma_i(\mathbf{A}\mathbf{Q}^\intercal) \leq \sigma_i(\mathbf{A})\sigma_1(\mathbf{Q})$. If $\mathbf{Q}$ has orthonormal columns, then $\sigma_1(\mathbf{Q}) = 1$, and we obtain $\sigma_i(\mathbf{Q}^\intercal \mathbf{A}) \leq \sigma_i(\mathbf{A})$.

By using the correct padding and Theorem 1 it holds with $\mathbf{Q} \in \mathbb{R}^{m\times p}$ for $p \geq 1$. □

The QR decomposition with column pivoting [21], QRCP (see Figure 1.3), derives the QR decomposition introducing a permutation matrix $\mathbf{\Pi}$ such that

$$\mathbf{A\Pi} = \mathbf{QR}.$$

This method introduces a pivoting at each iteration of Householder QR, in order to permute first the column of largest norm. The reference implementation is the `dgeqp3` routine from LAPACK. There is not yet an equivalent distributed routine `pdgeqp3` in the official release of ScaLAPACK, at the time of writing. ScaLAPACK only has `pdgeqpf`, an equivalent to LAPACK's `dgeqpf` routine.

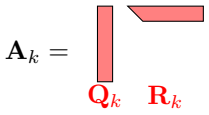


Figure 1.3: Truncated QR decomposition

The QR decomposition with column pivoting relies on a permutation matrix $\mathbf{\Pi}$ and computes the decomposition $\mathbf{A\Pi} = \mathbf{QR}$. There are several approaches for choosing a permutation $\mathbf{\Pi}$ that reduce the error of the low rank approximation $\|\mathbf{A} - \mathbf{A}_k\|_2$. One approach, referred to as QRCP, is to select the column with the largest norm [21] at each step of the QR factorization. Another approach is to reject $n - k$ columns one-by-one using the lowest singular value of the non-rejected columns [25, 26]. This strategy is sometimes referred to as reverse pivoting [96, 66, 106]. Strong rank revealing QR (strong RRQR) [59] is another approach that relies on computing QRCP followed by a number of additional column permutations. In our implementation we use `dlaqps` from LAPACK [7, 99, 41] which implements a block QRCP based on the original algorithm from Businger and Golub. In prior works, implementation of the truncated QRCP decomposition was obtain by modifying of the `dgeqp3` routine from LAPACK [42, 118]. However, our theoretical results are based on the bounds obtained by strong RRQR, that we present in the following two theorems.

Theorem 3. Gu and Eisenstat [59, Lem. 3.1 and Alg. 4] *Let* $\mathbf{A}$ *be an* $m \times n$ *matrix and* $1 \leq k \leq \min(m, n)$. *For any* $f > 1$ *there exists a permutation matrix* $\mathbf{\Pi}$ *such that the decomposition*

$$\mathbf{A\Pi} = \mathbf{QR} = \begin{bmatrix}\mathbf{Q}_1 & \mathbf{Q}_2\end{bmatrix} \begin{bmatrix}\mathbf{R}_{11} & \mathbf{R}_{12} \\ & \mathbf{R}_{22}\end{bmatrix}, \tag{1.1}$$

verifies for all $(i, j) \in [1, k] \times [1, n-k]$,

$$(\mathbf{R}_{11}^{-1}\mathbf{R}_{12})_{ij}^2 + \omega_i^2(\mathbf{R}_{11})\gamma_j^2(\mathbf{R}_{22}) \leq f^2. \tag{1.2}$$

A factorization satisfying Eq. (1.2) is thus called a strong RRQR factorization. We also use the relaxed form by summing over i [36, Cor. 2.3],

$$\gamma_j^2(\mathbf{R}_{11}^{-1}\mathbf{R}_{12}) + \gamma_j^2(\mathbf{R}_{22})/\sigma_{\min}^2(\mathbf{R}_{11}) \leq kf^2, \tag{1.3}$$

leading to the following theorem.

Theorem 4. [59, Thm. 3.2] and [36, Thm. 2.4] *Let* $\mathbf{A}$ *be an* $m \times n$ *matrix and* $1 \leq k \leq \min(m, n)$. *Let* $f > 1$ *and* $\mathbf{\Pi}$ *be a permutation matrix such that the decomposition described in Eq.* (1.1) *verifies for all* $(i, j) \in [1, k] \times [1, n-k]$

$$\gamma_j^2(\mathbf{R}_{11}^{-1}\mathbf{R}_{12}) + \gamma_j^2(\mathbf{R}_{22})/\sigma_{\min}^2(\mathbf{R}_{11}) \leq kf^2$$

Then for any $1 \leq j \leq n-k$ *and* $1 \leq i \leq k$

$$1 \leq \frac{\sigma_i(\mathbf{A})}{\sigma_i(\mathbf{R}_{11})} \leq \sqrt{1 + kf^2(n-k)}, \quad 1 \leq \frac{\sigma_j(\mathbf{R}_{22})}{\sigma_{k+j}(\mathbf{A})} \leq \sqrt{1 + kf^2(n-k)} \tag{1.4}$$

Note that in Eq. (1.4), being greater than or equal to 1 is satisfied in each case for any $\mathbf{\Pi}$ as a consequence of Theorem 1.

1.2.5 Cross approximation

Any matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ of rank r can be written as

$$\mathbf{A} = \mathbf{C}\hat{\mathbf{A}}^{-1}\mathbf{R},$$

where $\mathbf{C}$ is a subset of r columns of $\mathbf{A}$, and $\mathbf{R}$ is a subset of r rows of $\mathbf{A}$. $\hat{\mathbf{A}}$ is the submatrix at the intersection of both these rows and columns. This decomposition is known as *cross approximation* [112, 17]. Even though we did not focus on this during this work, it is closely connected to the QR decomposition (see e.g. Savostyanov and Tyrtyshnikov [101, Sec. 4]). Furthermore, this concept is explored in details in Oseledets [94], Oseledets et al. [95], Oseledets and Tyrtyshnikov [93] and especially the tensor train decomposition is partially based on this concept.

It is also important to mention at this point the *max-volume* concept [56]. In the case that $\mathbf{A}$ is square, the volume is given by its determinant. In the cross approximation, the best submatrix suited for the decomposition is determined by comparing the volume of the candidate submatrices.

1.3 Tensor approximation

This section introduces multilinear algebra with the definition of tensors and associated properties.

1.3.1 Introduction to tensors

Any discrete multivariate function $f(x_1, \dots x_d)$ can be represented as a hyper-rectangle of $\mathbb{R}^n$, a *tensor*. For a review of tensors and their decompositions, the reader is referred to Kolda and Bader [78]. Grasedyck et al. [57] is also very handy, as it lists the reference papers for many applications of tensor low-rank approximation. A tensor is a d-dimensional array of values, identified by a set of d indices (coordinates) each ranging from 1 to n_j, where $1 \leq j \leq d$. Then a value of the tensor is noted $\mathcal{A}(i_1 \dots i_d)$. If this tensor was built from a discretization [see 80] we have $\mathcal{A}(i_1 \dots i_d) = f(x_1(i_1), \dots x_d(i_d))$ where for a given i, x_i is a discrete mapping from the range $1 \dots d_i$ to the definition space of the ith variable of f. The letter i in this context represents a dimension, or *mode* of the tensor. The number d is the *order* of the tensor. Figure 1.4 represents a 3-order tensor. The red cube is the position of the value for indices (3,3,3).

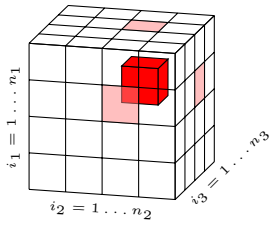


Figure 1.4: 3-order tensor, the inner red cube represents a specific value in the tensor.

When manipulating tensors, one can exchange, merge or expand indices without changing the number of coefficients. This change of structure in the tensor is a *reshape*. Merging indices, i.e. *unfolding* or *matricizing* the tensor, is done using the Kronecker product. In general indices are grouped into two new indices forming a matrix. To unfold a tensor $\mathcal{A}$, its indices $(i_1, \dots, i_d)$ are split into two groups, leading to the matrix indices $(I, J) = (i_1 \otimes \dots \otimes i_k, i_{k+1} \otimes \dots \otimes i_d)$. If I contains a single index i_j of $\mathcal{A}$, then the matricization is a j-mode unfolding of $\mathcal{A}$, noted $\mathcal{A}_j$.

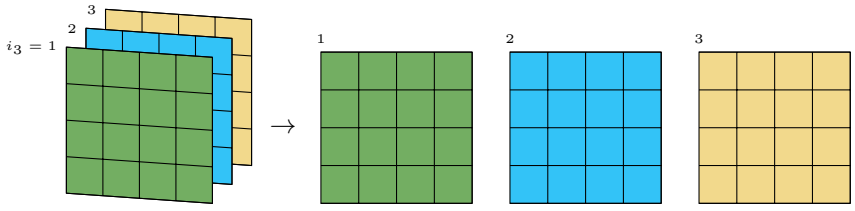


Figure 1.5: 1-mode unfolding of a 3-order tensor.

Figure 1.5 displays an 1-mode unfolding of a 3-order tensor. In this example, dimensions 2 and 3 are combined to form the column space of the matrix, such that $(I, J) = (i_1, i_2 \otimes i_3)$.

The matrix rank definition is generalized to the higher order tensor case, i.e. when $d > 2$, using the CP decomposition [63]. This decomposition consists of

a sum of rank-one tensors. A rank-one tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ can be written as the outer product of d vectors $\mathcal{X} = \mathbf{a}_1 \otimes \ldots \otimes \mathbf{a}_d$. The CP decomposition is written $\mathcal{A} = \sum_{i=1}^{r} \mathbf{a}_1^{(i)} \otimes \ldots \otimes \mathbf{a}_d^{(i)}$. The rank is then r, the minimal amount of rank-one tensor needed to totally represent $\mathcal{A}$. Unfortunately, the problem of finding the rank is NP-hard [70].

Matrix singular values do not have a straightforward equivalent for tensor. The simple alternative is to study the singular values of each unfolding.

Bader and Kolda [10] provide further details about tensor unfolding, and mentions several tensor products: inner product, outer product and contracted product. The latter, also known as *contraction*, is of interest to us hence recollected in the following. A contraction between two tensors is the summation of these tensors over a common index. For instance let tensors $\mathcal{A} \in \mathbb{R}^{n_1 \ldots n_d}$ have indices $(x_1, \ldots, x_d)$ and $\mathcal{B} \in \mathbb{R}^{m_1 \ldots m_d}$ have indices $(y_1, \ldots, y_d)$. For $1 \leq i, j \leq d$, if $n_i = m_j$ then the result of the contraction over index x_i and y_j is the tensor

$$\mathcal{C}(x_1 \ldots x_{i-1}, x_{i+1} \ldots x_d, y_1 \ldots y_{j-1}, y_{j+1} \ldots y_d) = \sum_{x_i, y_j, x_i = y_j} \mathcal{A}(x_1 \ldots x_d)\mathcal{B}(y_1 \ldots y_d).$$

Very similarly to the matrix product, two indices vanish and the remaining ones are combined in the resulting tensor. As a matter of fact, the contraction can be represented as the matrix product of the i and j-mode unfoldings $\mathcal{A}_i$ and $\mathcal{B}_j$, followed by the reshape of the resulting matrix into tensor $\mathcal{C}$. If $i = j$ the contraction is written $\mathcal{A} \times_i \mathcal{B}$.

The tensor decompositions studied in the present work involve a great amount of contractions, making it a common operation despite its complexity. However, the diagram representation introduced in Section 1.3.2 greatly simplifies its notation.

1.3.2 Tensor train

To compress a tensor, this thesis focuses on the Tucker decomposition, presented in Chapter 4. This decomposition is particularly efficient when the tensor order is reasonably small. For higher order tensors, one of the components of the Tucker decomposition (the core tensor) is of high order as well, thus having an exploding number of coefficients. To break this *curse of dimensionality*, we present another well-known tensor decomposition, proved useful for high dimensional data handling. Even though this decomposition is not directly involved in one of our contributions, it is highly connected to our work and plays an important part in the next section.

Any tensor can be decomposed as a contraction (product) of lower order tensors, called a tensor train. Before having a formal study in Oseledets [94], this decomposition was previously exposed in a form specific to quantum chemistry [115].

Algorithm 1 presents an algorithm to compute a sequence of 3-order tensors, forming a tensor train from a n-order tensor. One can verify this by multiplying (contracting) all tensors over the created indices $(\beta_1, \ldots, \beta_{n-1})$.

Figure 1.6 illustrates this contraction in diagram notation: each vertex represents a tensor and each edge a summation over an index, meaning that we multiply the two connected tensors over this index.

Algorithm 1 TT-SVD (Algorithm 1 from Oseledets [94]).

Require: A n-order tensor $\mathcal{A}(x_1, \ldots, x_n)$

1: Extend $\mathcal{A}$ by adding two constant indices (β_0, β_n) such that

$$\begin{aligned}\forall (x_1, \ldots, x_n) \in \mathbb{R}^n, \tilde{\mathcal{A}}_1(\beta_0, x_1 \ldots, x_n, \beta_n) &= \tilde{\mathcal{A}}_1(0, x_1 \ldots, x_n, 0) \\ &= \mathcal{A}(x_1 \ldots, x_n)\end{aligned}$$

2: **for** i from 1 to n **do**

3: Compute the truncated SVD of the (β_{i-1}, x_i)-mode unfolding $\tilde{\mathbf{A}}_i^{(x_i)} = \mathbf{USV}^\top$, introducing a new index β_i corresponding to the columns of $\mathbf{U}$ and the rows of $\mathbf{S}$

4: $\mathcal{M}_i$ is the tensorization of $\mathbf{U}$, thus has indices $(\beta_{i-1}, x_i, \beta_i)$

5: $\tilde{\mathcal{A}}_{i+1}$ is the tensorization of $\mathbf{SV}^\top$, thus has indices $(\beta_i, x_{i+1}, \ldots, x_n, \beta_n)$

6: **end for**

7: $\mathcal{M}_n \leftarrow \tilde{\mathcal{A}}_n$

Ensure: A sequence $(\mathcal{M}_1, \ldots, \mathcal{M}_n)$ of 3-order tensors such that $\forall (x_1, \ldots, x_n) \in \mathbb{R}^n$, $\mathcal{A}(x_1, \ldots, x_n) = \sum_{\beta_0 \ldots \beta_n} \mathcal{M}_1(\beta_0, x_i, \beta_1) \ldots \mathcal{M}_n(\beta_{n-2}, x_i, \beta_{n-1})$

Figure 1.6: An n-order tensor train in diagram notation.

Algorithm 1 can be generalized to decompose a tensor $\mathcal{A} \in \mathbb{R}^{k^n}, k \in \mathbb{N}$ into a train of 3-order tensors. The algorithm can be adapted by setting $x'_i = (x_{ki+1}, \ldots, x_{ki+k})$.

The diagram of the tensor train is in one dimension, but alternative methods take advantage of a two dimensional topology, or more, to obtain a tensor network [4].

1.4 Application to chemistry

This section motivates the development of low rank approximations of large matrices and tensors, by describing a numerical simulation employing these methods in the domain of physics and chemistry. This work being founded under the Extreme-Scale Mathematically-based Computational Chemistry (EMC^2) project, aiming at accelerating molecular simulations, along the preparation of the thesis we discussed with chemists to understand how advances in tensor algebra can be used to speed up molecular simulations. In this section we recollect our understanding in this domain, and present several angles to apply the methods introduced in the next chapters to improve molecular simulations.

To apply tensor algebra in the context of molecular simulation, we retained the DMRG algorithm (see Section 1.4.3). Applied to any molecular system, it models the electronic interactions and exposes inherent properties of the system such as the ground state energy. When the number of electrons is larger than 50, or even 100, state-of-the-art algorithms cannot cope with the exponential computational cost simulating fine-grained molecular models. Thanks to scalable tensors, this method is useful for larger molecules.

The first part of this section presents the quantum chemistry context, followed by a formulation of the problem in the tensor space, and finally the DMRG algorithm is introduced.

1.4.1 Introduction to the electronic Schrödinger equation

This section describes modelling a molecule using the electronic Schrödinger equation, introducing molecular orbitals and the Born-Oppenheimer approximation.

Consider a system (e.g. a molecule), and denote by R the positions of the nuclei and by $\mathbf{r}$ the positions of the electrons. In the Born-Oppenheimer approximation, nuclei are fixed points, hence R is a constant of the system.

The Schrödinger equation is written

$$H(\mathbf{r}, R)\Psi(\mathbf{r}, R) = E\Psi(\mathbf{r}, R), \tag{1.5}$$

where E and Ψ are the energy (a scalar) and its associated state of the system, unknowns of the problem, and H is the Hamiltonian, a given operator.

Expression of the Hamiltonian

In practice the Hamiltonian has the following expression.

$$H = \frac{1}{2}\sum \nabla_i - \sum_{\substack{i \text{ electron} \\ A \text{ nucleus}}} \frac{Z_A}{r_{Ai}} + \sum_{A>B \text{ nuclei}} \frac{Z_A Z_B}{R_{AB}} + \sum_{i>j \text{ electrons}} \frac{1}{r_{ij}} \quad (1.6)$$

$$= \hat{T}_e(r) + \hat{V}_{eN}(r,R) + \hat{V}_{NN}(R) + \hat{V}_{ee}(r), \quad (1.7)$$

where Z_A is the charge of nucleus A, r_{Ai} is the distance between nucleus A and electron i, R_{AB} is the distance between nuclei A and B, and r_{ij} is the distance between electrons i and j. In addition, $\hat{T}_e$ is the kinetic energies of the electrons, $\hat{V}_{eN}$ is the potential attraction energy between electrons and nuclei, $\hat{V}_{ee}$ is the potential repulsion energy between electrons and $\hat{V}_{NN}$ is the potential repulsion energy between nuclei.

Expression of the state function

Atomic orbitals are volumes containing (occupied) or not (unoccupied) exclusively one electron, according to the Pauli principle. Let φ_i be a basis function corresponding to the atomic orbital i. φ_i is usually built as a linear combination of Gaussian functions, different from zero only in the area of the orbital. An occupation state $I = (i_1, \ldots, i_N)$ is a combination of occupied orbital. We note N the number of electrons, and L the number of orbitals. We must have $L \geq N$. If $L > N$, excited state are considered. Even when looking for the ground state, excited states (higher layers orbitals) are included in the probabilistic model to obtain an accurate state function. An occupation state is described by a basis function ϕ_I, called molecular orbital, and written as a Slater determinant to respect the antisymmetric property: $\phi_I = \varphi_{i_1} \wedge \ldots \wedge \varphi_{i_L}$. The state function is the superposition of all physically possible occupation states I, weighed with a probability c_I, hence it is decomposed on the basis $(\phi_I)_I$, such that

$$\Psi = \sum_I c_I \phi_I.$$

This method involves Hilbert spaces and Galerkin approximation. For a complete mathematical overview of the problem, including spin considerations, consider reading Szalay et al. [107] . There are many ways to find the basis (Φ_I). As an example, one can compute the φ_i functions by solving local Schrödinger equations, and subsequently the Φ_I functions using the Hartree-Fock method. The Hartree-Fock method finds each coefficient c_I corresponding to an orbital by considering all other orbitals as constant interaction potentials. This work does not focus on building the basis set, and uses instead predefined techniques, see e.g. Hehre et al. [62]. In order to build the tensor train representation addressed thereafter, and to apply the DMRG algorithm, the coefficients of the second quantization can be computed with the Quantum Package [52].

Minimization problem

The first goal of molecular simulation is finding the ground state energy of the system. The different levels of energy are given by the eigenvalues of the Hamiltonian in the Schrödinger equation. The larger eigenvalue corresponds to

most excited state when considering a given set of orbitals, and a number of electrons, whereas the smallest signed eigenvalue corresponds to the ground state energy. Finding the smallest eigenvalue comes down to solving the minimization problem (more details can be found in Holtz et al. [64])

$$E_0 = \min_{\psi} \frac{(H\psi, \psi)}{(\psi, \psi)}, \tag{1.8}$$

where H is the Hamiltonian operator. The solution E_0 is the lowest energy level and the associated argument Ψ_0 is the corresponding state of the system. The scalar product denoted by $(,)$ is the scalar product of ℓ^2.

Discretization as a tensor

The construction of the basis set (ϕ_I) exposed in the previous section is independent to this research. To focus on the tensor expression, the basis set is abstracted by considering a set of given orbitals, and only the occupation information remains. We consider two spaces to pose this problem:

The CAR algebra The Pauli exclusion principle, the conservation of the particle number N, as well as the symmetries are encoded in the algebra. Operators are expressed as products composed of two basic operators: the creation operator $a_i^\dagger$, consisting of creating an electron in orbital i, and the annihilation operator a_i, consisting of removing an electron from orbital i.

The occupation state I corresponding to orbitals $(i_1, \ldots, i_L)$ occupied is written in the CAR algebra

$$\phi_I = a_{i_1} a_{i_2} \ldots a_{i_L} \mathcal{O},$$

with $\mathcal{O}$ being the empty state. The full state function hence writes as the different occupation states weighed by their probability

$$\Psi = \sum_I h_{i_1 \ldots i_L} a_{i_1} \ldots a_{i_L} \mathcal{O},$$

where $h_{i_1 \ldots i_L}$ is the probability that the system described by Ψ occupies orbitals $i_1 \ldots i_L$.

In the CAR algebra, the Hamiltonian can be written as

$$\hat{H} = \sum_{ij} h_{ij} \hat{c}_i^\dagger \hat{c}_j + \frac{1}{2} \sum_{ijkl} u_{ijkl} \hat{c}_i^\dagger \hat{c}_j^\dagger \hat{c}_l \hat{c}_k + V_{NN}. \tag{1.9}$$

This expression is called the second quantization formula. It transforms each operator from the Hamiltonian given in Eq. (1.6) with creation and annihilation operators. In each sum indices i, j, k and l loop over each orbital. The coefficient h_{ij} corresponds to the one electron integral, translating $\hat{T}_e(r) + \hat{V}_{eN}(r, R)$ with CAR algebra operators. The coefficient u_{ijkl} is the two electrons integral, translating $\hat{V}_{ee}(r)$ to the CAR algebra. V_{NN} is a constant scalar. With this expression, we can store the Hamiltonian of L orbitals as the combination of a basis set, a tensor $h_{ij} \in \mathbb{R}^{L^2}$, a tensor $u_{ijkl} \in \mathbb{R}^{L^4}$ and a scalar. All these elements can be computed using modern methods, for example with the Quantum Package [52]. It involves computing sums of integrals, hence their name [88, 62].

The tensor u_{ijkl} is large for complex molecules. Therefore, an interesting future goal would be to use the ST-HOQRTP algorithm introduced in Chapter 4 to approximate this tensor, in order to speedup its handling during standard simulations. See Xing et al. [119], Huang et al. [69] for existing research results on low rank representations of this tensor.

Encoding the problem in this way is very efficient, and used by many molecular simulation methods. But it makes it hard to use standard linear algebra methods, due to the particular laws of creation and annihilation operators. Therefore, we further transform this formulation to express it in a standard vector space. Even though this adds zeros and redundancies, it can be mitigated with tensor approximation.

The tensor algebra The state function is expressed as a vector of $\bigoplus_{i=1}^{L} \mathbb{R}^2$, where L is the total number of orbitals considered, and each $\mathbb{R}^2$ space corresponds to the state of one orbital with the following basis: $\begin{pmatrix}1\\0\end{pmatrix}$ corresponds to an occupied orbital and $\begin{pmatrix}0\\1\end{pmatrix}$ corresponds to an unoccupied orbital.

We can then translate all vectors and operators from the CAR algebra in this formalism. In particular,

$$\begin{aligned}
\mathcal{O} &= \begin{pmatrix}1\\0\end{pmatrix}_1 \otimes \ldots \otimes \begin{pmatrix}1\\0\end{pmatrix}_L, \\
a_i^\dagger &= \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix} \otimes \cdots \otimes \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix} \otimes \begin{pmatrix}0 & 0\\1 & 0\end{pmatrix}_i \otimes \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix} \otimes \cdots \otimes \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}, \\
a_i &= \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix} \otimes \cdots \otimes \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix} \otimes \begin{pmatrix}0 & 1\\0 & 0\end{pmatrix}_i \otimes \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix} \otimes \cdots \otimes \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}.
\end{aligned}$$

All operators in this context are in $\mathbb{R}^{2^L \times 2^L}$, including $a_i, a_i^\dagger$ and H. Operators can be summed and multiplied such that the Hamiltonian in tensor form can be deduced from Eq. (1.9). The Pauli principle, the conservation number and the symmetries are not encoded anymore, therefore a particular attention has to be paid on validating them. This new formulation lies in the well-known multilinear algebra domain. Yet to obtain this a very high-dimensional object was created, containing a lot of redundancy and sparsity. In the next section we present how to alleviate this using a new compression format.

1.4.2 Hamiltonian and state function as tensors

The previous section exposed the electronic Schrödinger equation, and reformulated it as a multilinear equation. Even though each dimension is small, each tensor involved has a very large number of dimensions, meaning that its number of coefficients is very large. In addition, these tensors are likely to be easily compressed, as they are quite sparse, and observe a lot of symmetries.

In this section we address the compression and manipulation of these tensors using the tensor train compression format (introduced in Section 1.3.2).

Two specific forms of tensor trains are used. Let n be the number of orbitals considered for the basis set. The first form, the Matrix Product State (MPS), represents the state function Ψ from Eq. (1.5). It is a train of n 3-order tensors,

each having one dimension from the original tensor. See Figure 1.7 for the diagram notation of the MPS.

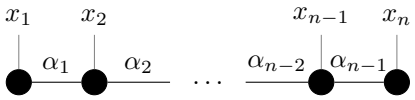


Figure 1.7: Representation of an MPS in diagram notation.

We distinguish two groups of indices. The indices $(x_1 \dots x_n)$ are fixed, of dimension 2, corresponding to the indices of the original tensor. The indices $(\alpha_1 \dots \alpha_{n-1})$ are much more intriguing. They can grow very large, especially in the middle of the train. Keeping them low, if possible, requires contracting tensors in the train. A method is given in Section 1.4.2, yet this question is still of interest, see e.g. Holtz et al. [65]. In the context of quantum chemistry, these indices are called bond dimensions. The index α_i represents the interactions between the electrons represented by the indices $(x_1, \dots, x_i)$ and those represented by the indices $(x_{i+1}, \dots, x_n)$. If we consider the matricization putting $(x_1, \dots, x_i)$ has rows and $(x_{i+1}, \dots, x_n)$ as columns, the rank of this matrix will be the minimal bond dimension α_i (without compression) [107, Thm. 3.4].

The second form, the Matrix Product Operator (MPO), represents the Hamiltonian operator H from Eq. (1.5). It is a train of n 4-order tensors, each having two dimensions from the original tensor. See Figure 1.8 for the diagram notation of the MPO.

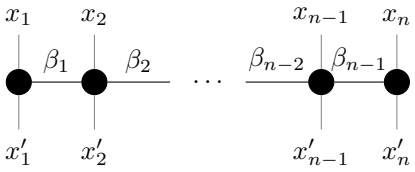


Figure 1.8: Representation of an MPO in diagram notation.

In the following we recall a few operations involving MPO or MPS, mentioned in Oseledets [94].

Product of tensor trains

Consider the MPS corresponding to wave functions $\Psi(x_1, \dots, x_n)$ and $\Psi'(x_1, \dots, x_n)$. The product of this two MPS is represented in Figure 1.9. This is equivalent to computing the scalar product (Ψ, Ψ') and the result is a scalar.

We now present two additional operations. When applying an operator $H(y_1 \dots y_n x_1 \dots x_n)$ to a wave function $\Psi(x_1, \dots, x_n)$ to compute the wave function $\Psi' = H\Psi$, having H in MPO form and Ψ in MPS form leads to the tensor product represented in Figure 1.10.

Finally, when composing two operators $H(y_1, \dots, y_n, x_1 \dots x_n)$ and $H'(x_1, \dots, x_n, x'_1 \dots x'_n)$ in tensor form, it leads to a tensor product as represented in Figure 1.11.

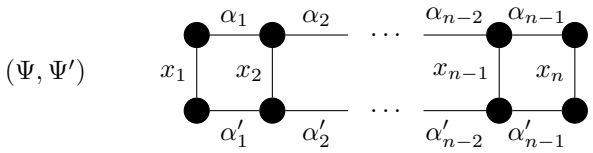


Figure 1.9: Scalar product of two MPSs. The result is a scalar.

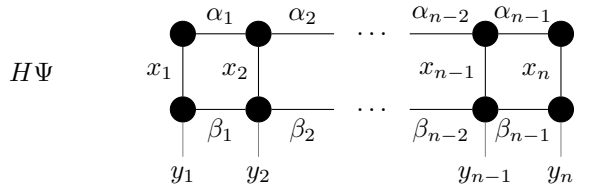


Figure 1.10: Apply an MPO H to an MPS Ψ.

Sum of tensor trains

The sum $\mathcal{G} = \mathcal{H} + \mathcal{F}$ of two tensors $\mathcal{H}(x_1, \ldots, x_n)$ and $\mathcal{F}(x_1, \ldots, x_n)$ of the same size is directly applicable to tensors in tensor train format. In this case the result $\mathcal{G}$ will also be in tensor train format.

As an illustration, let $\mathcal{H}_i(x_i, y_i, \beta_{i-1}, \beta_i)$ be the ith tensor in the tensor train format. $\mathcal{H}_i$ can be represented as a matrix of matrices, with the outer indices (β_{i-1}, β_i) and the inner indices (x_i, y_i), hence noted $\mathbf{H}_{i,x_i,y_i}$. For given indices $(x_1, \ldots, x_n, y_1, \ldots, y_n)$, the matrix product

$$\mathcal{H}(x_1, \ldots, x_n, y_1, \ldots, y_n) = \mathbf{H}_{1,x_1,y_1} \ldots \mathbf{H}_{n,x_n,y_n}$$

produces the corresponding coefficient of the tensor. Hence, the tensor addition translates as $\forall(x_1, \ldots, x_n, y_1, \ldots, y_n)$,

$$\begin{aligned}
\mathcal{G}(x_1 \ldots x_n y_1 \ldots y_n) &= \mathcal{H}(x_1 \ldots x_n y_1 \ldots y_n) + \mathcal{F}(x_1 \ldots x_n y_1 \ldots y_n) \\
&= \mathbf{H}_{1,x_1 y_1} \ldots \mathbf{H}_{n,x_n y_n} + \mathbf{F}_{1,x_1 y_1} \ldots \mathbf{F}_{n,x_n y_n} \\
&= \begin{pmatrix} \mathbf{H}_{1,x_1 y_1} & \mathbf{F}_{1,x_1 y_1} \end{pmatrix} \begin{pmatrix} \mathbf{H}_{2,x_2 y_2} & \\ & \mathbf{F}_{2,x_2 y_2} \end{pmatrix} \cdots \begin{pmatrix} \mathbf{H}_{n,x_n y_n} \\ \mathbf{F}_{n,x_n y_n} \end{pmatrix},
\end{aligned}$$

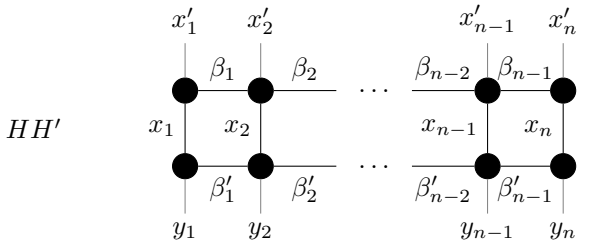


Figure 1.11: Composing two MPOs.

involving the block matrix product. Therefore, each tensor in the tensor train of $\mathcal{G}$ can be identified. This property is summarized in the following proposition.

Proposition 1. *Consider two operators $\mathcal{H} \in \mathbb{R}^{n_1 \cdots n_L}$ and $\mathcal{F} \in \mathbb{R}^{n_1 \cdots n_L}$ in MPO form. Their global indices have identical sizes, but the bond dimensions may be of different sizes. In the notations above, we denote by $\mathcal{G}$ the sum of operators $\mathcal{G} = \mathcal{H} + \mathcal{F}$ in MPO form. It has bond indices $\{\gamma_1, \ldots, \gamma_L\}$ such that $|\gamma_i| = |\alpha_i| + |\beta_i|$. Computing $\mathcal{G}$ is equivalent to performing the following local computation on each tensor of the tensor train format.*

If $1 < i < L$,

$$\mathcal{G}_i(x_i, y_i, \gamma_{i-1}, \gamma_i) = \begin{cases} \mathcal{H}_i(x_i, y_i, \alpha_{i-1}, \alpha_i), \text{ if } \gamma_{i-1} \leq \alpha_{i-1} \text{ and } \gamma_i \leq \alpha_i \\ \mathcal{F}_i(x_i, y_i, \beta_{i-1}, \beta_i), \text{ if } \gamma_{i-1} > \alpha_{i-1} \text{ and } \gamma_i > \alpha_i \\ 0, \text{ otherwise} \end{cases},$$

if $i = 1$,

$$\mathcal{G}_1(x_1, y_1, \gamma_1) = \begin{cases} \mathcal{H}_i(x_1, y_1, \alpha_1), \text{ if } \gamma_1 \leq \alpha_1 \\ \mathcal{F}_i(x_1, y_1, \beta_1), \text{ if } \gamma_1 > \alpha_1 \end{cases},$$

if $i = 1$,

$$\mathcal{G}_i(x_L, y_L, \gamma_{L-1}) = \begin{cases} \mathcal{H}_i(x_L, y_L, \alpha_{L-1}), \text{ if } \gamma_{L-1} \leq \alpha_{L-1} \\ \mathcal{F}_i(x_L, y_L, \beta_{L-1}), \text{ if } \gamma_{L-1} > \alpha_{L-1} \end{cases}.$$

It is clear that the summation of tensor trains introduces sparsity to the final MPO. In fact, a sequence of 2×2 blocks will appear on the diagonal. Another important question is optimizing this operation. Note that this is the difference between naive and compact construction in Keller et al. [76].

Multiplication by a scalar

The product $\mathcal{F} = \alpha\mathcal{H}$ where $\alpha \in \mathbb{R}$ and $\mathcal{H}$ is a tensor in MPO form is equivalent to scaling only one of the tensors in the tensor train.

$$\mathcal{F}_i = \begin{cases} \alpha\mathcal{H}_i, \text{ if } i = 1 \\ \mathcal{H}_i, \text{ if } i > 1 \end{cases}$$

Normalization and recompression of a tensor train

They are two analogous operations necessary to complete the list of tools to manipulate tensors in the context of quantum chemistry. The first one, normalizing a tensor train, applies to an MPS Ψ and requires $(\Psi, \Psi) = 1$ (involving the tensor train product). The second one, compressing a tensor train, aims at finding close to optimal bond dimensions. Both operations involve a similar technique often referenced as optimizing the tensor train, mentioned in Snajberk [105, Sec. 4.1.3], Dolfi et al. [37, Sec. 2.2] and Fröwis et al. [51, Sec. IV.A].

Normalization consists of applying sequentially the QR decomposition on each unfolded tensor of the train. The $\mathbf{Q}$ factor reshaped becomes the new tensor, and the $\mathbf{R}$ is integrated in the next tensor before applying QR again. The last $\mathbf{R}$ factor is discarded. Very similarly, the compression uses a k-truncated

SVD instead of QR, and does not discard the last right factor. The right singular vectors are transferred to the next tensor, and the left singular vectors together with the singular values form the new current tensor. The new bond dimension has size k.

The presentation of the tensor train format shows that this representation benefits the manipulation of the Hamiltonian and the state function in different ways. First it enables to break the curse of dimensions, by converting an L^2-order tensor to a sequence of 4-order tensors. It also proves efficient for basic operations such as product and sum of MPO and MPS. For these operations, only a local operation on each tensor of the train is needed, hence making it possible to perform the operation using a distributed memory architecture [see e.g. 82, 87]. Furthermore, the compression of tensor trains is tightly related to low rank approximation, and we can expect methods presented in this work to improve current implementations. The QRTP algorithm presented in Chapter 3 is a good candidate for finding the ranks of the tensor trains (the bond dimension). The randomized methods presented in Chapter 2 are already used to speed up the tensor train compression [27, 3], and using bSRHT is a promising direction.

1.4.3 Density Matrix Renormalization Group

The Density Matrix Renormalization Group (DMRG) algorithm is the key to linking tensor algebra and electronic molecular simulation. It was introduced in White [115] and reviews include Schollwoeck [103], Chan et al. [24], Baiardi and Reiher [11]. More mathematical overviews include Holtz et al. [64], Keller et al. [76].

The goal is to solve the minimization problem of Eq. (1.8) where the state function and the Hamiltonian are in tensor train formats.

The solution is to perform local minimizations on each index. The unknown is the ground state energy E_0 (the smallest eigenvalue), and its associated eigenvector Ψ_0. The procedure to find Ψ_0 is as follows.

1. Build the Hamiltonian in tensor train from the second quantization decomposition.
2. Initialize Ψ as a random orthonormal tensor.
3. For $1 \leq i \leq L$, minimize $(H\Psi_i(x_i, \alpha_i, \alpha_{i+1}), \Psi_i(x_i, \alpha_i, \alpha_{i+1}))$ to obtain the smallest eigenvalue E and associated argument Ψ_i. Replace the argument into Ψ.
4. Output the final E and Ψ.

We now detail each step. 1) The one-electron and two electron integrals describe the Hamiltonian. By performing multiplications, additions and multiplications with scalars of operators in tensor train formats, a tensor train for the full Hamiltonian is obtained (see the previous section for details on these operations). It may require several compressions, or a block diagonal storage to optimize its use. 2) We require an initial guess for Ψ. The most common way is to generate it randomly with a bond dimensions of size 1 and normalize it such that $(\Psi, \Psi) = 1$. 3) We craft a particular $\Psi_i(x_i, \alpha_i, \alpha_{i+1})$ by removing the

ith tensor in the MPS. By contracting all other indices in the inner product, we obtain a 6-order tensor. Unfolding the tensor by regrouping 3 indices together, and computing the smallest eigenvalue of the unfolding, we obtain an associated eigenvector. We can reshape this vector into a 3-order tensor Ψ_i that we can replace in the train of the MPS. 4) After iterating on each index of the MPS (several passes are sometimes needed to converge, called sweeping) the resulting E approximates the global ground state energy and Ψ the associated state.

1.4.4 Conclusion

This section introduced the quantum chemistry application, relatively to the research project funding this thesis. We presented the DMRG algorithm involving tensor low rank approximation, with the goal to simulate large molecules. At various steps of this method the distributed algorithms introduced in the next chapter, bSRHT, QRTP and ST-HOQRTP, could be used to speed up computations: randomization and QR are adequate for compressing the tensor train, as well as computing *a priori* properties of the tensor train; the Tucker decomposition done with ST-HOQRTP would improve handling the two electrons integral more efficiently. This goal is an active interest of our research team, and, as one of the first study of this project, the role of this thesis in this area is merely introductory. Building efficiently the MPO and solving local minimization problems using optimized routines are challenges ahead. This question is also a great opportunity to study the tensor train format, widely used in numerical multilinear algebra.

1.5 Matrix generation

This section is dedicated to exhibit various methods to generate a dense distributed matrix. Several criteria are important to select a generation method. The perfect matrix is deterministic, low-rank with parametrizable rank and has a structure originated from a real life problem. Having an implicit formula is a plus because it makes it possible to generate it locally on each processor. The alternative is to store the matrix in different files, growing quickly with the number of blocks, and then read the corresponding part of the corresponding file on each processor.

1.5.1 Random matrices

The most naive method is to generate each coefficient as independent random variables from the uniform distribution $\mathcal{N}(0, 1)$. It doesn't have any internal structure due to independent randomness.

As displayed in Figure 1.12 the first singular value is larger, e.g. 500 for a 1000×1000 matrix, and subsequent singular values decrease slowly. This method is easy to implement and doesn't require storing any file.

1.5.2 Matrices with prescribed singular values

This method produces a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ from a given profile of singular values. First initialize a vector $\mathbf{s} = (s_1, \ldots, s_{\min(m,n)})$ with those singular values,

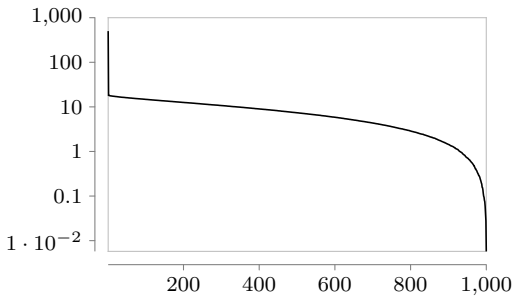


Figure 1.12: Singular values of a 1000×1000 random matrix drawn from independent variables in $\mathcal{N}(0, 1)$.

for instance it can be formed with inverse decay coefficients such that

$$\mathbf{s}_i = \alpha/i, \alpha > 1,$$

or with exponential decay such that

$$\mathbf{s}_i = e^{-\alpha i},$$

or with two segments decay such that for a given separating index k,

$$\begin{cases} \mathbf{s}_i = nk - ni + 1 & \text{if } i \leq k \\ \mathbf{s}_i = 1 - i/n & \text{if } i > k. \end{cases}$$

It additionally requires generating two random orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ of respective sizes $m \times \min(m, n)$ and $\min(m, n) \times n$. We can then finally build the matrix $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}$ where $\mathbf{\Sigma}$ is a diagonal matrix having $\mathbf{s}$ as coefficients.

This method is useful to set the desired singular value profile, and compare the approximated singular values to the original ones.

1.5.3 Specific problems

These are several matrices generated from simulating specific problems, exhibited in Table 1.1.

These problems offer matrices with various singular values profiles, including gaps, or very close values as well. The matrix size is an input parameter, such that any size can be obtained for a given problem. One limitation lies in the sequential nature of the computational scheme of each problem. Therefore, to obtain a distributed matrix, we can only write the whole matrix at once in an intermediate file, then reading a part on each node. This finally limits the maximum matrix size to a single node's memory size.

1.5.4 Matrix and tensor repositories

The following mentions several websites offering test matrices and tensors.

- Matrix market - small sparse matrices.

Number	Name	Size	Description
1	`baart`	1000×1000	1st kind Fredholm integral equation
2	`blur`	900×900	Digital image deblurring
3	`deriv2`	1000×1000	Computation of the second derivative
4	`foxgood`	1000×1000	Severely ill-posed problem
5	`gravity`	1000×1000	1-D gravity surveying model problem
6	`heat`	1000×1000	Inverse heat equation
7	`parallax`	28×1000	Stellar parallax problem with 28 obs.
8	`phillips`	1000×1000	Phillips' "famous" problem
9	`shaw`	1000×1000	1-D image restoration model
10	`spikes`	1000×1000	Problem with a "spiky" solution
11	`tomo`	900×900	Create a 2D tomography test problem
12	`ursell`	1000×1000	Integral equation with no ℓ^2 solution
13	`wings`	1000×1000	Problem with discontinuous solution

Table 1.1: Matrices generated from integral equations

- SuiteSparse Matrix Collection - medium size sparse matrices.
- LIBSVM Data - medium size to very large machine learning datasets, useful to flow into the RBS kernel method.
- UCI Machine Learning Repository - medium size to very large machine learning datasets, useful to flow into the RBS kernel method.
- FROSTT - large to very large sparse tensors.

These datasets being very structural, they are good candidates for compression. Nevertheless, most matrices and tensors are sparse or too small to fit the need for large dense matrix compression. A possible explanation is the apparent difficulty to store such an object online. Furthermore, there is no possibility to generate the matrix on each processor (except by using the RBS kernel method) when considering the matrices available online. The only option is reading from the disk. This makes it difficult to use for high performance computing.

1.6 High performance computing

The computational aspect of this thesis makes it important to present various inherent characteristics of high performance computing. The different algorithms introduced in the subsequent chapters are run with scalability in mind, therefore the following high performance computing principles were considered for their design.

1.6.1 Numerical Linear Algebra softwares

C++ and Fortran are low level languages. Implementing with these languages requires more time, nevertheless they provide a fine-grained control over each memory byte and each floating point operation (flop). They make it very easy to understand how the time is spent when executing the program, find bottlenecks

and optimize them. Hence, they allow to provide very fast and memory efficient programs.

Julia and Python are high-level languages. One can translate easily mathematical expressions to the programming language. But the distance to the compiled code is larger and numerous choices of implementation are made by the interpreter and package developers, which are hence not specific to an application.

LAPACK [7] is a linear algebra package developed in Fortran widely used by the community. It is known to contain the state-of-the-art algorithms of the common used linear algebra tools. BLAS (Basic Linear Algebra Subroutines) is at the heart of LAPACK, providing optimized low level linear algebra routines. LAPACK only implements sequential methods, i.e. running on a single computer. Note however that multiprocessing is possible.

When one wants to involve more than one node, having separate memory spaces, one has to use internode communication. This is the role of MPI (Message Passing Interface). Julia also includes a master-workers communication layer. In the MPI paradigm however all processes have equivalent roles.

ScaLAPACK [18], based on PBLAS and BLACS, implements linear algebra routines in Fortran intended for distributed environment.

The Intel corporation provides tools to run specifically on its own hardware. In particular, Intel develops its own modified implementation of LAPACK and ScaLAPACK called MKL (mathematical kernel library), provides its own Fortran and C++ compilers, as well as its own MPI implementation. These implementations often prove more optimized than the Netlib ones when running on Intel CPUs.

Running Linear Algebra programs frequently involves operations on large matrices. Numerous low-level and hardware optimizations have to be taken into account when manipulating large matrices. Vectorization enables to apply an identical operation to several registers at the same time. Pipelining enables to use various modules of a CPU at the same time when executing the same sequence of operations a number of times. These enhancements are oriented towards the use of large data operations rather than many small tasks. For example, it is more efficient to multiply two matrices rather than computing each coefficient separately with numerous scalar products.

1.6.2 Supercomputer topology

The large scale numerical experiments presented in this document were done on the TGCC Joliot-Curie cluster. It was installed at the end of 2017. In November 2022, it was ranked 83 in the TOP500, listing the 500 most computation-powerful clusters in the world.

Considering only the Skylake hardware we used, the cluster contains 1656 nodes shared amongst 6 cabinets. Each cabinet can host 48 blades of 3 nodes each.

Each node is composed of 2 Intel Xeon Platinum 8168 processors. This processor was released in the 3rd quarter of 2017, it runs at 2.7 GHz with 33 MB of L3 cache. In addition, each node contains 180 GB of shared memory. All nodes are interconnected in a fat tree topology with a 100 GB/s Infiniband EDR network.

To have an idea of the environmental impact of the experiments, Table 1.2 displays the energy consumption for several of the runs exposed in Chapter 3. This energy is given by the job scheduler of the cluster, for each job independently.

Table 1.2: Energy consumption on the TGCC Joliot cluster.

Matrix size	time (s)	Node count	Processor count	Energy (KJ)
2048×2048	5	1	4	1.7
8192×8192	8	2	64	3.6
65536×65536	40	86	4128	1237
131072×131072	420	342	16416	36135

On this cluster, the experiments used in total 65306.31 hours core. By taking an average of 20 KJ per hour cores, it means that this project used, over three years, 1306 MJ of energy, i.e. 362 kWh. It is equivalent to having a 100-watt light bulb operating for 150 days.

Chapter 2

Subspace projection with the block subsampled randomized Hadamard transform

We study in this chapter how randomness is used to build a projection space together with methods to implement it on a parallel computer. This projection is key to constructing a low rank approximation of a matrix, the matter at hand in this work. This work partly lead to the writing of Balabanov et al. [12].

The first section introduces the theoretical background for randomization, as well as standard sequential algorithms. The subsequent sections present our contribution. Given a very large matrix $\mathbf{A}$, we have two goals: sampling (also called sketching) $\mathbf{A}$ to decrease the row dimension, addressed in Section 2.2; building a low rank approximation of $\mathbf{A}$, addressed in Section 2.3.

2.1 Context

Using randomness to extract information from a large system is a relatively recent research topic. In addition to validating a global error constraint, random projections preserve local properties of vectors such as the distance. Since the Johnson-Lindenstrauss theorem [71, 32], where a fundamental theorem bounding the distance between projected vectors to the distance between the original vectors was given, many projection mappings have been studied (e.g. the fast Johnson-Lindenstrauss transform [2]) and many applications have been pursued. Overall such projections enable to faster compute low rank approximations [49, 50, 98, 48], matrix multiplications, singular value decompositions, and faster solve linear systems of equations.

This section gives the basic theorem and the main projection operators, especially we introduce the subsampled randomized Hadamard transform which we will abundantly use for our novel method presented in the rest of the chapter.

For further details, randomization is reviewed widely in Woodruff [117], Martinsson and Tropp [85].

2.1.1 Subspace embeddings

The previous chapter defined orthogonal projections and their use to build low rank approximations. In this chapter we introduce projections specific to randomization. The next embedding properties characterize this kind of projections, and provides guarantees useful for its subsequent use. Theorem 5 shows that there exists a mapping from $\mathbb{R}^m$ to $\mathbb{R}^l$ with controlled distortion. This mapping projects all columns from a matrix A to a reduced size $l < m$. Originally formulated in Johnson et al. [71], the bound on the parameter l results from a more recent contribution [32].

Theorem 5 (Johnson-Lindenstrauss Theorem [71, 32])**.** *Let n be an integer, and ε be such that $0 \leq \varepsilon \leq 1$. Let l be a positive integer such that $l > 4(\varepsilon^2/2 - \varepsilon^3/3)^{-1} \ln n$. For any set V of n points in $\mathbb{R}^m$, there is a map $f : \mathbb{R}^m \to \mathbb{R}^l$ such that*

$$\forall (u, v) \in V^2, (1-\varepsilon)\|u-v\|^2 \leq \|f(u)-f(v)\|^2 \leq (1+\varepsilon)\|u-v\|^2.$$

This mapping is referenced in the literature as a subspace embedding.

In the following we take the map f as a matrix $\mathbf{\Omega} \in \mathbb{R}^{l \times m}$. If $\mathbf{\Omega}$ is independent of the data (f does not depend on V in the notation of Theorem 5) and drawn from random variables, it is called an (ε, δ, d)-*oblivious subspace embedding* (OSE), as defined in Definition 1.

Definition 1 (Oblivious subspace embedding)**.** *Let m and l be two positive integers such that $l < m$. Let $0 \leq \delta \leq 1$ and $0 \leq \varepsilon \leq 1$. The matrix $\mathbf{\Omega} \in \mathbb{R}^{l \times m}$ is an (ε, δ, d)-oblivious subspace embedding for the norm $\|.\|$ when for any fixed d-dimensional subspace $S \subset \mathbb{R}^m$,*

$$(1-\varepsilon)\|x\|^2 \leq \|\mathbf{\Omega}x\|^2 \leq (1+\varepsilon)\|x\|^2 \quad \text{for all } x \in S \tag{2.1}$$

holds with probability at least $1 - \delta$.

In practice, an oblivious subspace embedding is generally a linear operator mixing all coefficients of the operand vector, such that extracting a few coefficients from the result is sufficient to conserve the distance and orthogonality of the vectors in the projected space. We now present a few well-known embeddings.

2.1.2 Gaussian sampling

The reference random embedding is the *Gaussian embedding* consisting of a matrix $\mathbf{\Omega} \in \mathbb{R}^{l \times m}$ where each coefficient of $\mathbf{\Omega}$ is an independent uniform random variable of $\mathcal{N}(0, 1)$. In addition, the matrix is multiplied by a scaling factor $\sqrt{1/l}$ [85, Sec 8.3], such that

$$\forall \mathbf{x} \in \mathbb{R}^m, \mathbb{E}\left[\|\mathbf{\Omega}\mathbf{x}\|^2\right] = \|\mathbf{x}\|^2. \tag{2.2}$$

Theorem 8.4 and Analysis 8.7.2 from Martinsson and Tropp [85] lead to say that Gaussian sampling respects the oblivious subspace embedding property, for example by taking $\varepsilon = 0.8$, and $l = 2m$.

This mapping is easy to code and performs with a complexity similar to the multiplication of two matrices, and is widely used in the literature. Thus, it is used here as the reference embedding to evaluate the new method.

From Eq. (2.2) we have that $\mathbb{E}\left[\|\mathbf{\Omega A}\|_F\right] = \|\mathbf{A}\|_F$. It is not always true with the operator norm. Indeed, we have $\|\mathbf{\Omega A}\|_2^2 = \|\mathbf{\Omega A}\|_F^2 - \sum_{i=2}^{n} \sigma_i^2(\mathbf{\Omega A})$ and the tail plays an important role. In the specific case where the first singular value is dominant, i.e. if $\sigma_1(\mathbf{A}) \gg \sigma_2(\mathbf{A})$ and $\sigma_1(\mathbf{\Omega A}) \gg \sigma_2(\mathbf{\Omega A})$, then both norms are almost equal and $\mathbb{E}\left[\|\mathbf{\Omega A}\|_2\right] \approx \mathbb{E}\left[\|\mathbf{\Omega A}\|_F\right] = \|\mathbf{A}\|_F \approx \|\mathbf{A}\|_2$.

According to Tropp [110, Sec. 1.3], given a set of n vectors, $l \approx \log(n)$ is sufficient to conserve the distance between vectors.

2.1.3 Structured random embeddings

We can also build a subspace embedding by associating a mixing (structured and sometimes random step) with a sampling (random step). For the mixing part, the literature offers many options. We present a few in this section and Martinsson and Tropp [85, Sec. 9] provides a more refined list.

The discrete Fourier transform (DFT) uses trigonometric properties. Consider the mth root of 1: for $0 \leq k \leq m-1$

$$w_m^k = e^{ki2\Pi/m}.$$

Then the matrix $F_m \in \mathbb{R}^{m \times m}$ where

$$(F_m)_{ij} = w_m^{(i-1)(j-1)}$$

defines the DFT. It is specific for the complex case, and a clever algorithm, the Fast Fourier Transform (FFT), reduces the complexity dramatically. There also exists parallel implementations for the FFT.

The Walsh-Hadamard transform (WHT) is recursively defined for any m a power of 2 as

$$H(2) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

and for any $k > 1$

$$H(2^k) = \begin{bmatrix} H(2^{k-1}) & H(2^{k-1}) \\ H(2^{k-1}) & -H(2^{k-1}) \end{bmatrix} = H(2) \otimes H(2^{k-1}).$$

Examples of the Hadamard matrix:

$$H(4) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix},$$

$$H(8) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}.$$

This method is suited for real matrices whose size is a power of two, or using zero-padding. Associated with a random signs vector and a random permutation, it forms the subsampled randomized Hadamard transform (SRHT) introduced in Tropp [110]. This full-featured subspace embedding takes the form of the following $\mathbf{\Omega}$:

$$\mathbf{\Omega} = \sqrt{\frac{m}{l}}\mathbf{RHD},$$

where $\mathbf{R} \in \mathbb{R}^{l\times m}$ is a random sample selection matrix, $\mathbf{H} \in \mathbb{R}^{m\times m}$ is the Walsh-Hadamard matrix scaled by $1/\sqrt{m}$, and $\mathbf{D} \in \mathbb{R}^{m\times m}$ is a diagonal matrix of random coefficients in $\{-1, 1\}$.

In the continuity of Tropp [110, Thm. 1.3], Boutsidis and Gittens [20, Lem. 4.5] claim that the SRHT operator conserves the orthogonality of the input matrix. It is recollected in Theorem 6.

Theorem 6. *Fix an $m \times n$ matrix $\mathbf{V}$ with orthonormal columns, and let m be a power of 2. Let $0 < \varepsilon < 1$ and $0 < \delta < 1$. Draw a random $l \times m$ SRHT matrix $\mathbf{\Omega}$ where the embedding dimension satisfies*

$$\frac{8}{3}\varepsilon^{-1}\left[\sqrt{n} + \sqrt{8\log(m/\delta)}\right]^2 \log(n/\delta) \leq l \leq m.$$

Then, with probability at least $1 - 3\delta$, for all $i = 1, \ldots, n$,

$$\sqrt{1-\sqrt{\varepsilon}} \leq \sigma_i(\mathbf{\Omega V}) \leq \sqrt{1+\sqrt{\varepsilon}}.$$

It is shown in Tropp [110] that the SRHT is an oblivious subspace embedding, and this can be extended to other structured embeddings [85, Rem. 9.3]. The SRHT is a very handy tool because it can be applied implicitly, meaning that it is not necessary to store beforehand the matrix in memory to perform the product $\mathbf{\Omega A}$, and computing the product is cheaper than the matrix product. This gives the SRHT good chances to be more efficient than Gaussian sampling in practice.

The matrix $\mathbf{R}$ randomly selects rows. There is a significant discussion about whether $\mathbf{R}$ should select identical rows. The selection is then said to be *with replacement*. Tropp achieved to prove its version of Theorem 6 with replacement with the help of Gross and Nesme [58] [see 110, Sec. 2.2]. Boutsidis and Gittens [20] provides theorem for with and without replacement, as well as additional properties when sketching any matrix. Given an input matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, computing $\mathbf{B} = \mathbf{\Omega A} \in \mathbb{R}^{l\times n}$ including a selection with replacement has chances to produce duplicated rows in $\mathbf{B}$ and make the matrix singular, specifically if $l \lesssim n$. Therefore, it is important to keep $l \gg n$ if the resulting matrix is required to be nonsingular.

2.2 Sampling a matrix in parallel

We consider a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ distributed on a $P_r \times P_c$ grid of processors. We note $P = P_r P_c$ the total number of processors. If $P_r = 1$, $\mathbf{A}$ is *1D column-partitioned*, noted $\mathbf{A} = \begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2 & \ldots & \mathbf{A}_P\end{bmatrix}$, if $P_c = 1$, $\mathbf{A}$ is *1D row-partitioned*,

noted $\mathbf{A} = \begin{bmatrix}\mathbf{A}_1; \mathbf{A}_2; \ldots; \mathbf{A}_P\end{bmatrix}$, otherwise $\mathbf{A}$ is *2D partitioned*, noted

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \ldots & \mathbf{A}_{1P_c} \\ \vdots & & \vdots \\ \mathbf{A}_{P_r1} & \ldots & \mathbf{A}_{P_rP_c} \end{bmatrix}.$$

The first objective is to sample a matrix using a distributed architecture.

One can reduce the dimension of $\mathbf{A}$ by multiplying it from the left with a random short and wide matrix $\mathbf{\Omega}$. The resulting matrix $\mathbf{B} = \mathbf{\Omega}\mathbf{A}$ has fewer rows, while retaining similar inner product between columns, $\mathbf{\Omega}$ satisfying indeed the oblivious subspace embedding property described in Section 2.1.1. Subsequently to Section 2.1.1 are exposed different ways to build such an $\mathbf{\Omega}$. In this work we introduce adaptations of these methods to take advantage of a parallel architecture.

2.2.1 Block Gaussian sampling

The Gaussian sampling method is widely used, and it has a parallel version popular as well. Indeed, it can be applied to a 2D distributed matrix with only one reduce operation. This method is given in Algorithm 2 for a 1D row-partitioned matrix.

Algorithm 2 1D Block Gaussian Sampling

Require: $\mathbf{A} = \begin{bmatrix}\mathbf{A}_1; \mathbf{A}_2; \cdots; \mathbf{A}_P\end{bmatrix} \in \mathbb{R}^{mP \times n}$ distributed on P processors
1: **for** $i = 1 \ldots P$ **do**
2: Draw $\mathbf{\Omega}_i \in \mathbb{R}^{l \times m}$ a Gaussian sampling matrix
3: $\mathbf{B}_i \leftarrow \mathbf{\Omega}_i \mathbf{A}_i$
4: **end for**
5: $\mathbf{B} \leftarrow \sum_i \mathbf{B}_i$

The matrix $\mathbf{\Omega}_i$ from Algorithm 2 is drawn independently on each processor from the normal distribution $\mathcal{N}(0, 1)$ and scaled with $\sqrt{\frac{1}{lP}}$, such that $\forall \mathbf{x} \in \mathbb{R}^n, \mathbb{E}\|\mathbf{\Omega}\mathbf{x}\| = \|\mathbf{x}\|$ [85, Sec. 8.3]. The step 3 is the dominant cost in terms of floating point operations (flops). It requires a matrix multiplication having complexity $O(lmn)$. The step 5 implies a reduction step which is the dominant cost of communications, with $O(\log(P))$ messages and a total volume of $O(ln\log(P))$. The resulting matrix $\mathbf{B}$ is stored on the elected processor (typically the one of lowest global identification number). We consider two other cases: $\mathbf{A}$ is 1D column-partitioned and $\mathbf{A}$ is 2D-partitioned.

If $\mathbf{A} \in \mathbb{R}^{m \times nP}$ is 1D column-partitioned, the matrix $\mathbf{\Omega} \in \mathbb{R}$ with scaling factor $\sqrt{\frac{1}{lP}}$ has to be applied on each processor such that $\mathbf{B}_i \leftarrow \mathbf{\Omega}\mathbf{A}_i$. Then the result $\mathbf{B} = \begin{bmatrix}\mathbf{B}_1 & \mathbf{B}_2 & \cdots & \mathbf{B}_P\end{bmatrix}$ has the same distribution grid as $\mathbf{A}$. In this case it is not necessary to broadcast $\mathbf{\Omega}$ to all processors, and only the random number generator seed (a single integer) is shared. Thus, it does not require any significant communication.

If $\mathbf{A} \in \mathbb{R}^{mP_r \times nP_c}$ is 2D-partitioned, ideas from both 1D row and column-partitioned algorithms are combined. The matrices $\mathbf{\Omega}_i$ from Algorithm 2 with scaling factor $\sqrt{\frac{1}{lP_r}}$ are duplicated using the seed on their respective rows of

processors, then applied to each block and reduced to the first column. The resulting matrix $\mathbf{B}$ is distributed on the first column of processors.

The Gaussian sampling is simple to implement and lays out good performance. Therefore, it is the reference implementation, used here for comparison.

2.2.2 Block subsampled randomized Hadamard transform

We now introduce the parallel adaptation of the subsampled randomized Hadamard transform (SRHT) detailed in Section 2.1.3. Computing the exact SRHT is poorly suited for parallelisation. We introduce a derivated algorithm to obtain communication and computation efficiency in a distributed framework. Indeed, SRHT can be applied on each block of the distributed input matrix, as with block Gaussian sampling. We call this algorithm block SRHT (bSRHT). We present in this section first the 1D left sketching, meaning that for a given 1D or 2D distributed input matrix $\mathbf{A}$ we compute $\mathbf{\Omega A}$. Note that this part can be adapted to the 1D right sketching, i.e. to compute $\mathbf{A\Omega}^\intercal$. In a second part we present 2D sketching, where the goal is to compute $\mathbf{\Omega A\Theta}^\intercal$ where $\mathbf{\Theta}$ and $\mathbf{\Omega}$ are built with two different sets of independent random variables.

Applications can differ a lot depending on the input matrix distribution and the sketching direction. For instance in the case of 1D left sketching with a 1D row-partitioned matrix we can expect very large distributed vectors coming from a very tall and skinny matrix or from a streaming input of data. Changing the distribution or the sketching direction of this matrix results in a smaller dimension to sketch and thereby making the method less efficient. Consequently, 2D sketching requires both dimensions of the input matrix to be very large.

Algorithm 3 1D left block SRHT

Require: $\mathbf{A} = \left[\mathbf{A}_1; \mathbf{A}_2; \cdots; \mathbf{A}_P\right] \in \mathbb{R}^{mP \times n}$ distributed on P processors, l sampling size

1: Draw $\mathbf{R} \in \mathbb{R}^{l \times n}$ a truncated random permutation matrix
2: Broadcast $\mathbf{R}$
3: **for** $i = 1 \dots P$ **do**
4: Draw $\mathbf{d}_R^{(i)} \in \{\pm 1\}^n$ and $\mathbf{d}_L^{(i)} \in \{\pm 1\}^n$ Rademacher vectors
5: $\mathbf{B}_i \leftarrow diag(\mathbf{d}_R^{(i)})\mathbf{A}_i$
6: $\mathbf{B}_i \leftarrow \mathbf{H}\mathbf{B}_i$
7: $\mathbf{B}_i \leftarrow diag(\mathbf{d}_L^{(i)})\mathbf{B}_i$
8: $\mathbf{B}_i \leftarrow \sqrt{\frac{m}{l}}\mathbf{R}\mathbf{B}_i$
9: **end for**
10: $\mathbf{B} \leftarrow \sum_i \mathbf{B}_i$

Ensure: $\mathbf{B}$

Given a 1D row partitioned matrix $\mathbf{A} \in \mathbb{R}^{mP \times n}$, computing the left random projection $\mathbf{\Omega A}$ of $\mathbf{A}$ using bSRHT is described in Algorithm 3. It consists of a local SRHT applied on each block $\mathbf{A}_i$ followed by a sum of all the blocks, i.e. a reduction with the summation operator. The operator $\mathbf{\Omega} \in \mathbb{R}^{l \times mP}$ is written in explicit matrix form as

$$\mathbf{\Omega} = \begin{bmatrix} \mathbf{\Omega}_1 & \mathbf{\Omega}_2 & \dots & \mathbf{\Omega}_P \end{bmatrix}. \tag{2.3}$$

Each block $\mathbf{\Omega}_i \in \mathbb{R}^{l \times m}$ details as

$$\mathbf{\Omega}_i = \mathbf{S}\mathbf{D}_i^{(1)}\mathbf{H}\mathbf{D}_i^{(2)},$$

where $\mathbf{S} \in \mathbb{R}^{l \times m}$ randomly selects rows, $\mathbf{D}_i^{(1)}$ and $\mathbf{D}_i^{(2)} \in \{-1, 1\}^{m \times m}$ are sign matrices, equivalently referred to as diagonal matrix with a Rademacher vector as the diagonal, and $\mathbf{H} \in \mathbb{R}^{m \times m}$ is the Walsh-Hadamard matrix. $\mathbf{S}$ and $\mathbf{H}$ are identical in all blocks yet the sign matrices are built with different random variables for each block, and for each of the two. Note that it can happen that $l \leq m$, then the local SRHT would increase the size of the input blocks. This is only feasible if the random selection is done *with replacement* as mentioned in Section 2.1.3.

The oblivious subspace embedding property is defined in Section 2.1.1. It provides guarantees on the distance between columns of the sketched matrix with regard to the original matrix. Theorem 7 gives a value for l for which Ω is an oblivious subspace embedding, when the selection is done with replacement. In particular, this is important to make sure that we can perform an efficient subspace selection on the sketched matrix, mirroring the behavior of the original matrix. By doing this, an economic operation can be executed on the sketch rather than a full expensive operation on the original matrix.

Theorem 7 (Balabanov et al. [12])**.** *Let* $0 \leq \varepsilon \leq 1$ *and* $0 \leq \delta \leq 1$. *Let* $\mathbf{\Omega} \in \mathbb{R}^{l \times m}$ *defined as in Eq.* (2.3). *If*

$$l \geq 3\varepsilon^{-2}(\sqrt{d} + \sqrt{8\log(6mP/\delta)})^2 3d/\delta,$$

then $\mathbf{\Omega}$ *is an* (ε, δ, d) *oblivious subspace embedding.*

The proof of Theorem 7, derived by Oleg Balabanov, is provided in Balabanov et al. [12]. The given bound shows that the required number of rows in $\mathbf{\Omega}$ is similar to the sequential algorithm counterpart, given in Theorem 6.

This algorithm can be adapted for other distribution layouts, and the theory still applies. If the input matrix is 1D column-partitioned, then the local SRHTs are computed in parallel without communication, and the same output matrix as with a global SRHT is obtained. If the input matrix is 2D-partitioned, then bSRHT is performed on each column of the processor grid without extra communication, obtaining the same result as with one column of processors. However, in both cases the result is 1D column-partitioned.

It can also be adapted to right sketching, with any distribution layout, by left sketching the transpose of the input matrix. Beware that the processor grid is also transposed in this case.

We now present how to implement 2D sketching, i.e. from left and right. To compute the 2D sketch of a 2D-partitioned matrix, one can perform local SRHT from left and right locally on each processor, and then sum all blocks of size $l \times l$ on an elected process. This is described in Algorithm 4.

This algorithm requires twice more flops than the 1D version, and $O(\log(P))$ messages, for a total communication volume of $O(l^2 \log(p))$. In Section 2.3 2D sketching is discussed further and applied to the Nyström approximation.

2.2.3 Numerical experiments

In this section we provide numerical experiments of 1D right sketching with bSRHT.

Algorithm 4 2D block SRHT.

Require: $\mathbf{A} \in \mathbb{R}^{mP_r \times nP_c}$ distributed on a $P_r \times P_c$ processor grid
1: $\mathbf{B} \leftarrow \mathbf{O}$
2: **for** $j = 1 \dots P_c$ **do**
3: **for** $i = 1 \dots P_r$ **do**
4: $\mathbf{\Omega}_i \leftarrow$ generate $l \times m$ random matrix
5: $\mathbf{\Theta}_j \leftarrow$ generate $l \times n$ random matrix
6: $\mathbf{B}_j \leftarrow \mathbf{B} + \mathbf{\Omega}_i \mathbf{A}_{i,j} \mathbf{\Theta}_j^\intercal$
7: **end for**
8: **end for**
Ensure: $\mathbf{B} \in \mathbb{R}^{l \times l}$ on processor 1

RBS kernel

For numerical experiments, all-purpose test matrix generation methods are given in Section 1.5. An additional method specifically intended for this chapter is presented here. The radial basis function enables to build a dense positive definite matrix $\mathbf{A}$ of size $n \times n$ from a dataset containing n records.

We denote by $\mathbf{x}_i$ the ith record of the dataset. The matrix $\mathbf{A}$ is built with the formula

$$a_{ij} = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / \sigma^2}.$$

In this work we limit ourselves to numerical datasets, but it could be easily extended to any type of data, as long as the distance between two records exists and provides a real number.

The parameter σ is a real number greater than zero. The larger it is, the steeper the singular values. If σ is very small, $\mathbf{A}$ is close to the identity.

We use the datasets described in Section 2.2.3.

Table 2.1: List of input datasets for the RBS kernel method.

Name	Record length	Maximum record number	description
`mnist`	784	8,100,000	70000 images of handwritten materials from 500 different writers. Each record corresponds to an image is converted to a vector.
`year`	90	515,345	Prediction of the release year of a song from audio features.
`xor`	129	6,000,000	Physical Unclonable Functions (PUFs) simulation, specifically XOR Arbiter PUFs. PUFs are used for authentication purposes.
`botnet`	115	7,062,606	Network traffic simulating a botnet attack.

To generate a distributed matrix, each processor only needs to read the

records corresponding to its row as well as column indices. Therefore, if a processor generates a submatrix of size $m \times n$, it only needs reading (and parsing) $m + n$ records. Figure 2.1 displays the singular values of the matrix generated from each dataset.

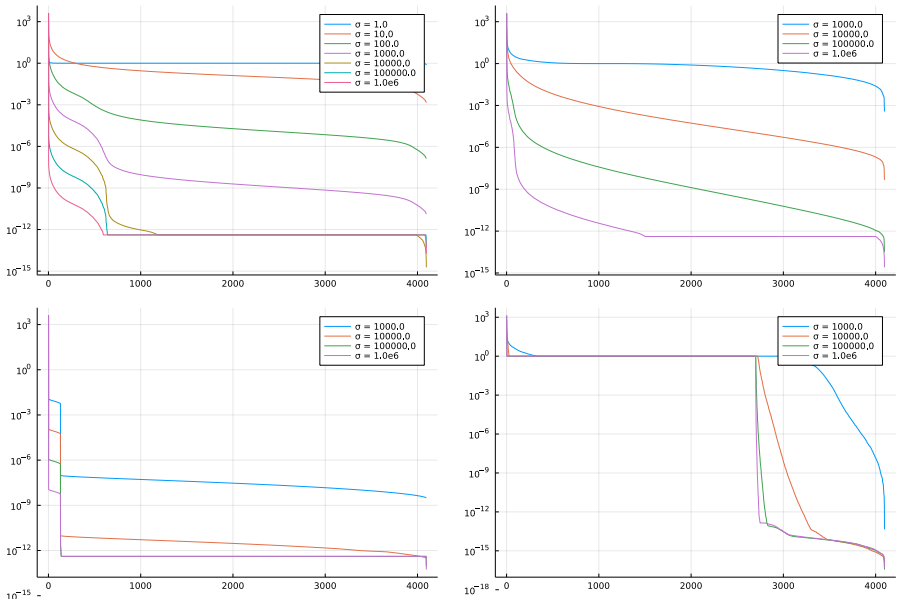
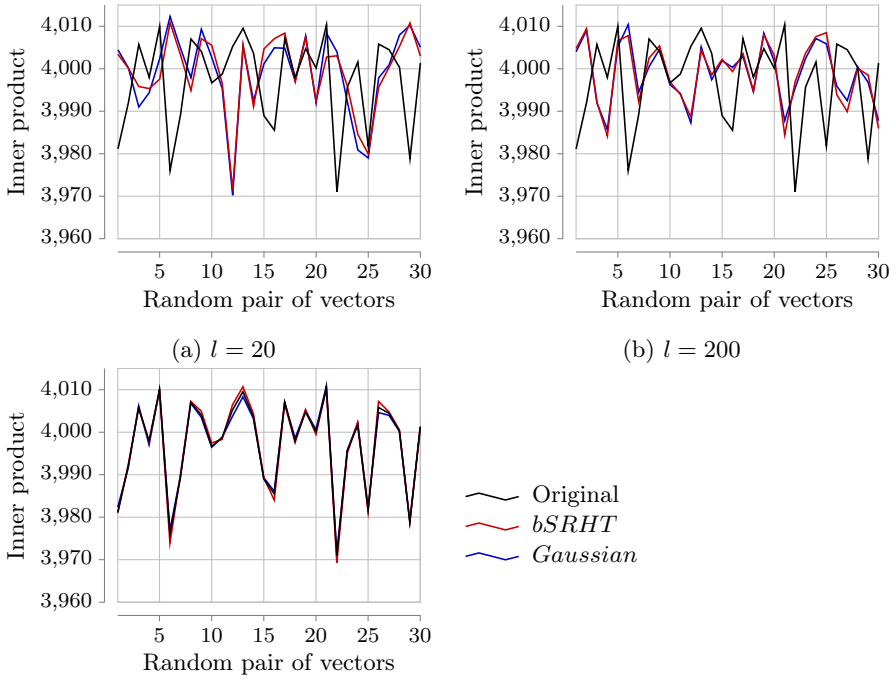


Figure 2.1: Singular values for the matrix formed with the RBS kernel method applied to the first 4096 records of datasets `mnist`, `year`, `xor` and `botnet` for different values of σ.

Accuracy

According to Theorem 7 the bSRHT is a subspace embedding, meaning that it preserves inner products with high probability. In the following we study how inner products between columns of a matrix are preserved. Indeed, validating this property is paramount for many use cases in low rank approximation, for instance to select pivots in the QRCP decomposition (see Sections 1.2.4 and 3.1.2). Considering an input matrix of size 4096×4096 1D-row partitioned on 4 processors, 30 pairs of vectors are randomly drawn, and their inner product is compared with the inner product of their sketch. The sketch is performed with either bSRHT or Gaussian sampling. Two different input matrices were chosen. The first one is built using the RBS kernel method applied to the `mnist` dataset. The results are displayed in Figure 2.2. The second input matrix is drawn from random variables, and its results are displayed in Figure 2.3. Each subfigure corresponds to a different sampling size: 20, 200 and 500.

The figures show clearly that increasing the sampling size makes the sketch more reliable. Indeed, sketched vectors better mimic the original behavior when a larger sampling size is used. In the case of structured data, here the `mnist` dataset, large sampling size results in very reliable preservation of angles. On the other hand, for random data, it does not completely eliminate discrepancies.

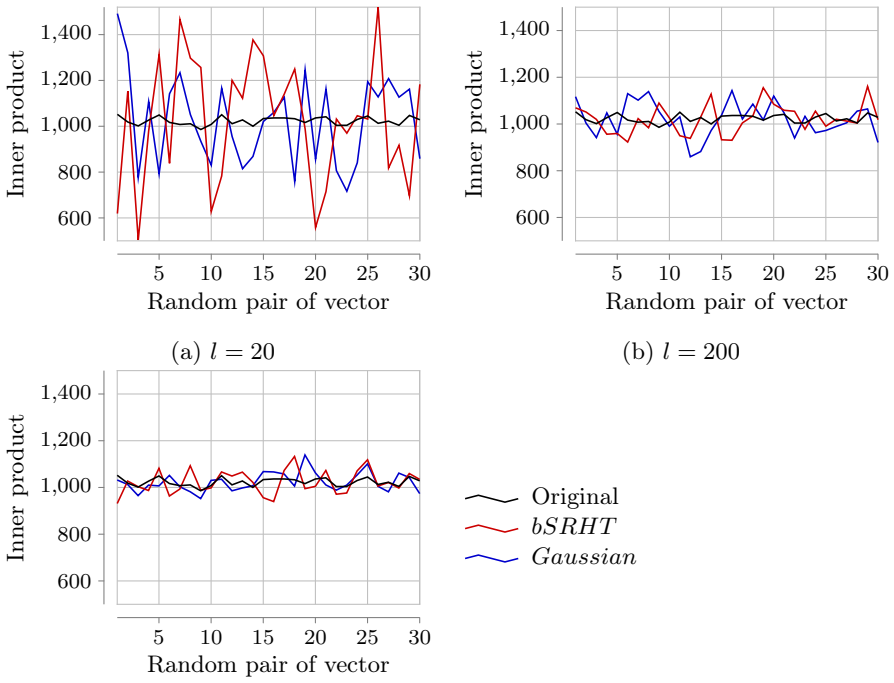


(a) $l = 20$

(b) $l = 200$

(c) $l = 500$

Figure 2.2: Angle conservation after randomization of the `mnist` dataset. Using 4 processors, and using 4096 records of the dataset.



(a) $l = 20$

(b) $l = 200$

(c) $l = 500$

Figure 2.3: Angle conservation after randomization of a random matrix of size 4096×4096. Using 4 processors.

Comparing bSRHT and Gaussian, for the `mnist` dataset they are very close, almost coinciding, whereas for random data both behaviors differ entirely from one another. Furthermore, the distributed method bSRHT does not exhibit worst performance than the classical sampling, pledging for its use in real life applications.

Execution time

Previous section illustrated that bSRHT is very close in accuracy to Gaussian sampling. The focus is now on the computational performance, comparing the runtime of both methods. Gaussian sampling and SRHT are implemented with Julia programming language version 1.5.3 along with `Hadamard.jl` and `Statistics.jl` packages. The source code for SRHT and Gaussian sampling is available in Appendix A. Consider first the sequential Gaussian sampling and SRHT. The Gaussian sampling $\mathbf{\Omega A}$ where $\mathbf{\Omega} \in \mathbb{R}^{l \times m}$ and $\mathbf{A} \in \mathbb{R}^{m \times n}$ is equivalent to a matrix product, hence a complexity of $O(mnl)$. SRHT's most expensive operation is the fast Walsh-Hadamard transform (step 6 of Algorithm 3) with complexity $O(mn \log(n))$. Note that the complexity of SRHT is independent of the sampling size l, and the ratio $l/\log(n)$ represents how SRHT and Gaussian sampling execution times compare. If this ratio is large, SRHT will be faster than Gaussian sampling. On the contrary, if this ratio is small, it suggests that Gaussian sampling is faster. Figure 2.4 displays numerical results, taking $m = 65536, n = 256$ or 512 and l varying from 2 to 4096.

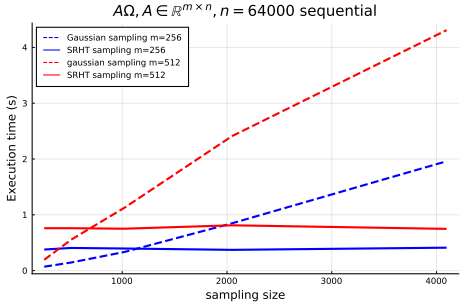


Figure 2.4: Block Gaussian and bSRHT execution time with varying sampling sizes.

For large values of l, the runtime of Gaussian sampling becomes large, making the use of SRHT advantageous.

SRHT is also interesting when it comes to memory usage, because the sampling matrix is never formed. In our use case, the sampling matrix can be very large, for instance given a matrix of size 256×65536 and a sampling size of 4096 the sampling matrix will have 268 million coefficients and take up to 17 gigabytes of memory.

Consider the bSRHT method, on a distributed architecture. The experimental environment is made of 8 nodes, each being a 2x Cascade Lake Intel Xeon 5218 2.4GHz of 32 cores. The input matrix is distributed using the `Distributed.jl` and `DistributedArrays.jl` Julia packages. Figure 2.5 shows the scaling execution time while increasing the number of cores, each core owning a submatrix of size 4096×256 (corresponding to the case of Figure 2.4 where

$l = 4096$), distributed on a column of processors. The green line stands for the communication part of the algorithm, the reduction. Note that the reduction takes an insignificant part of the execution time, even for a large number of cores.

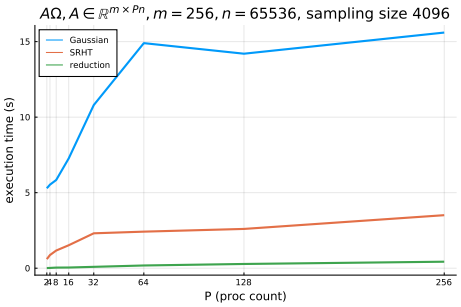


Figure 2.5: Weak scaling of Algorithm 3 with local submatrix size $256 \times 65,536$.

Notice that the slope is steeper (meaning scalability is worse) when there are less than 64 cores. There is up to 32 cores per node, so from 1 to 32 we increase the number of processes on the same node. A possible explanation is that some resources are shared amongst the cores of a same node (for example the L3 cache is shared amongst all cores of a node). Increasing the number of cores will overload these resources and slow down the execution of each process.

2.3 Low-rank approximation with parallel sampling

2.3.1 Nyström approximation

The Nyström approximation aims to obtain a low rank approximation from a symmetric positive semi-definite (SPSD) matrix. It involves sketching from both right and left of the matrix. The output decomposition reads as

$$\mathbf{A}_k = (\mathbf{\Omega A})(\mathbf{\Omega A \Omega}^\intercal)^+(\mathbf{\Omega A})^\intercal. \tag{2.4}$$

The parentheses delimit 3 different terms in the decomposition, pictured in Figure 2.6, such that $\mathbf{A}_k = \mathbf{B}^\intercal\mathbf{C}\mathbf{B}$ where $\mathbf{B} = \mathbf{\Omega A}$ is the left sketch, and $\mathbf{C} = \mathbf{B\Omega}^\intercal$ is the left and right sketch of the input matrix. The matrix $\mathbf{\Omega}$ corresponds to applying a random sampling to a matrix on the right, and $\mathbf{\Omega}^\intercal$ on the left. Similarly to other low rank approximations, this decomposition makes it possible to store and use the matrix with less resources.

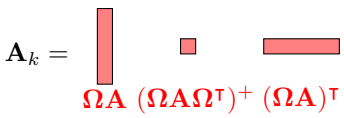


Figure 2.6: Low rank approximation with the Nyström method.

The Nyström approximation involves 3 steps, summarized in Algorithm 5. It is not necessary here to use directly the 2D sketching presented in Algorithm 4, as sketching in two steps makes use of the intermediate 1D sketch $\mathbf{B}$ needed to build the decomposition.

Algorithm 5 Nyström approximation

Require: A matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$
1: Left sketch $\mathbf{A}$: $\mathbf{B} = \mathbf{\Omega A}$
2: Right sketch $\mathbf{B}$: $\mathbf{C} = \mathbf{B\Omega}^\intercal$
3: Compute the pseudo-inverse $\mathbf{C}^+$
Ensure: $\mathbf{B}$ and $\mathbf{C}^+$ forming the Nyström decomposition $\mathbf{A}_k = \mathbf{B}\mathbf{C}^+\mathbf{B}^\intercal$.

The 2D-sketched matrix $\mathbf{C}$ is usually singular, due to SRHT with replacement repeating columns and rows. Therefore, it is necessary to apply pretreatment. The next section presents different ways to deal with this.

2.3.2 Inversion of the middle matrix

Inverting a matrix is not computationally efficient. Therefore, the concatenation is better obtained by combining factors $\tilde{\mathbf{B}} = \mathbf{B}\mathbf{C}^+$ or $\bar{\mathbf{B}} = \mathbf{B}\mathbf{C}^{-\frac{1}{2}}$. There are several ways to apply this matrix inverse. The non-exhaustive list includes the least-square system resolution, GMRES and Conjuguate gradient. Alternatively the matrix inversion can be performed after decomposing the matrix with Cholesky, QR or SVD. The latter proves useful when $\mathbf{\Omega A\Omega}^\intercal$ is numerically singular, i.e. a subset of its singular values are close to machine precision. In the extreme case the matrix is so ill-conditioned that the Cholesky decomposition is not applicable. To circumvent this, the smallest singular values can be ignored by using the SVD.

In more details, to compute the inverse of a rank-deficient matrix one has to perform the truncated SVD (see Section 1.2.3). The input rank k is determined with a threshold ε such that $\forall i > k, \sigma_i(\mathbf{\Omega A\Omega}^\intercal) \leq \varepsilon$. The singular vectors are equal because $\mathbf{A}$ is SPSD, and therefore $\mathbf{\Omega A\Omega}^\intercal$ also. We obtain a replacement matrix $\mathbf{\Omega A\Omega}^\intercal = \mathbf{U}_k\mathbf{\Sigma}\mathbf{U}_k^\intercal$ and $\mathbf{\Omega A\Omega}^\intercal = \mathbf{U}_k\mathbf{\Sigma}^{-1}\mathbf{U}_k$. We can also work with the matrix square root such that

$$\begin{aligned}
\mathbf{\Omega A}(\mathbf{\Omega A\Omega}^\intercal)^+(\mathbf{\Omega A})^\intercal &= \mathbf{\Omega A}\mathbf{U}_k\mathbf{\Sigma}_k^{-\frac{1}{2}}{\mathbf{\Sigma}_k^{-\frac{1}{2}}}^\intercal\mathbf{U}_t^\intercal(\mathbf{\Omega A})^\intercal \\
&= \mathbf{\Omega A}\mathbf{U}_k\mathbf{\Sigma}_k^{-\frac{1}{2}}(\mathbf{\Omega A}\mathbf{U}_k\mathbf{\Sigma}_k^{-\frac{1}{2}})^\intercal \\
&= \bar{\mathbf{B}}\bar{\mathbf{B}}^\intercal,
\end{aligned}$$

converting the decomposition of Eq. (2.4) to a 2-factor decomposition $\mathbf{A}_k = \bar{\mathbf{B}}\bar{\mathbf{B}}^\intercal$.

With this in mind, we present a refined version of the randomized Nyström approximation in Algorithm 6 optimized for parallel processing. Because $\tilde{\mathbf{B}}$ is tall and skinny, its SVD is efficiently obtained by using the R factor from the QR decomposition computed with the TSQR algorithm.

2.3.3 Numerical results

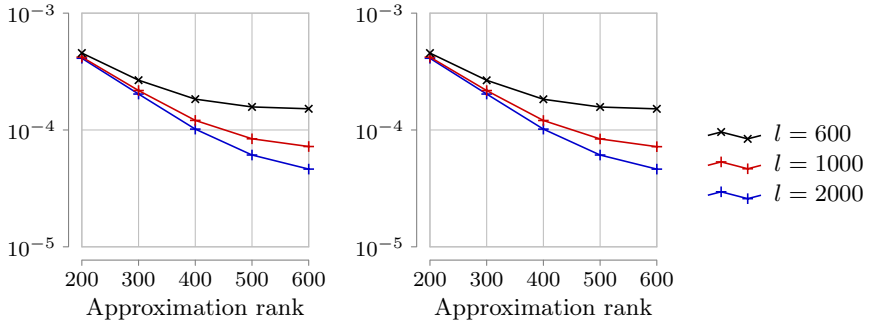
To illustrate the practical usability of block SRHT we now present numerical experiments. Its efficiency and accuracy are characterized through compari-

Algorithm 6 Randomized Nyström approximation, suited for distributed computing.

Require: $n \times n$ matrix $\mathbf{A}$, $l \times n$ matrix $\mathbf{\Omega}$ with $l \ll n$, the target rank k.

1: Compute $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}^\intercal$.
2: Obtain a Cholesky factor $\mathbf{C}$ of $\mathbf{\Omega}\mathbf{Y}$.
3: Compute $\tilde{\mathbf{B}} = \mathbf{Y}\mathbf{C}^{-1}$ with backward substitution.
4: Obtain the R factor $\mathbf{B}$ of $\tilde{\mathbf{B}}$ (with TSQR or similar).
5: Use SVD to compute the best rank-k approximation $\tilde{\mathbf{U}}_k\tilde{\mathbf{\Sigma}}_k\tilde{\mathbf{V}}_k^\intercal$ of $\mathbf{B}$.
6: Compute $\mathbf{U}_k = (\mathbf{Y}\tilde{\mathbf{V}}_k)\tilde{\mathbf{\Sigma}}_k^{-1}$.
7: Output factorization $\mathbf{A}_k^{(\mathrm{Nyst})} = \mathbf{U}_k\tilde{\mathbf{\Sigma}}_k^2\mathbf{U}_k^\intercal$.

son with the Gaussian embeddings. We picked the Nyström approximation as the representative application. The experiments were executed with Julia programming language version 1.7.2 along with the `Distributed.jl` and `DistributedArrays.jl` packages for parallelism. We used 2 nodes Intel Skylake 2.7GHz (AVX512) having 48 available cores and 180 MB of RAM each. In this experiment we used only 32 cores on each node. As input data we used the datasets `mnist`, `year`, `xor` and `botnet`, mentioned in Section 2.2.3. They are input to the RBS kernel method, generating a distributed matrix. The parameter σ was respectively chosen as 10000 for `mnist`, 1000 or 10000 for `year`, and 1000 for `xor` and `botnet`, and the dimension n of the input matrix is chosen as 32768 or 65536. The matrix $\mathbf{A}$ has been uniformly distributed on a square grid of 8×8 processors. In all the experiments, the local matrices $\mathbf{\Omega}^{(i)}$ on each processor were generated with a seeded random number generator with a low communication cost.



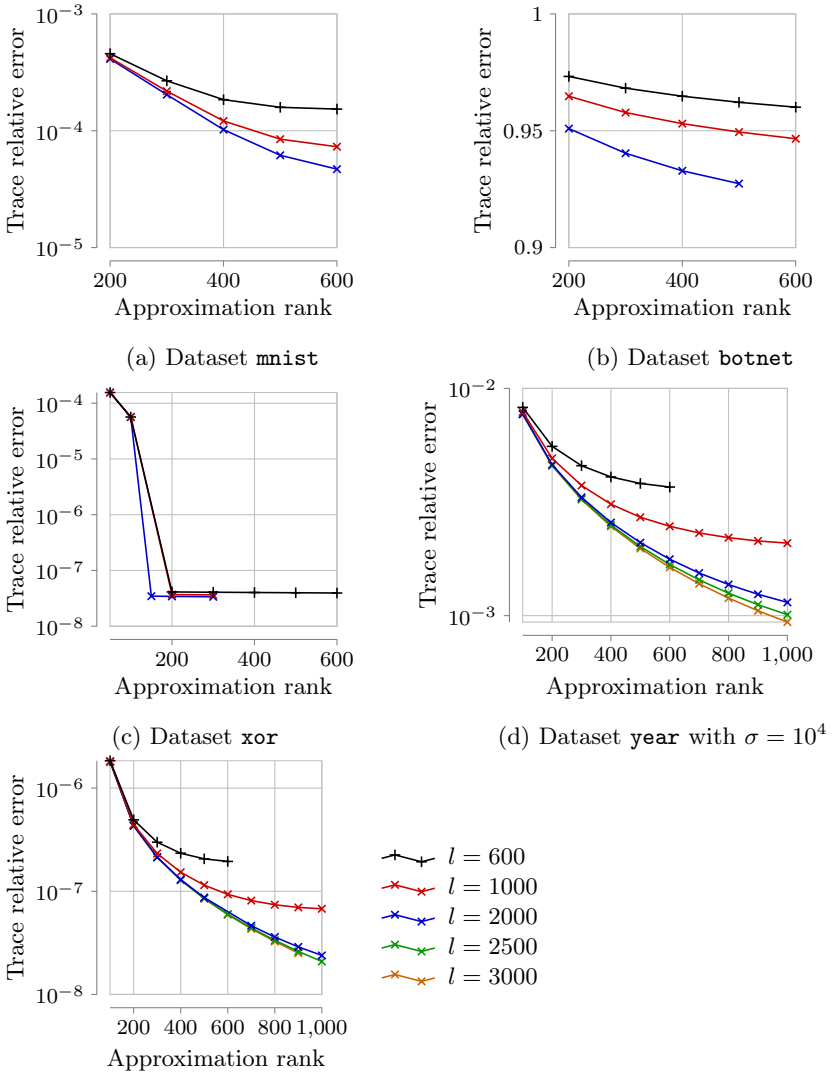
(a) Block SRHT (b) Gaussian matrix

Figure 2.7: Trace error $\|\mathbf{A} - \mathbf{A}_k^{(\mathrm{Nyst})}\|_*/\|\mathbf{A}\|_*$ of the Nyström approximation computed with bSRHT and Gaussian sampling. The input matrix is built with dataset `mnist` and has dimension $n = 32768$.

Figure 2.7 depicts the convergence of the error of the low-rank approximation obtained with Algorithm 5 taking $\mathbf{\Omega}$ as either block SRHT or Gaussian sampling, and $n = 32768$. In this numerical experiment, the error is measured with the trace norm. Different sketching sizes l were tested. For each pair of parameters (l, k) 20 different approximations were computed for each type of $\mathbf{\Omega}$, in order to

have the 95% confidence interval. Nevertheless, this interval is not displayed as it is too small to be visible, that particularly implies the stability of Algorithm 6. This figure shows that block SRHT and Gaussian embedding give very similar results.

Figure 2.8 extends this analysis for larger input matrices by taking $n = 65536$, and other datasets. Similarly to the case $n = 32768$, the Gaussian sampling and bSRHT exhibit almost identical results for all datasets. Therefore, to save space the Gaussian sampling plot is not displayed here.



(a) Dataset `mnist`

(b) Dataset `botnet`

(c) Dataset `xor`

(d) Dataset `year` with $\sigma = 10^4$

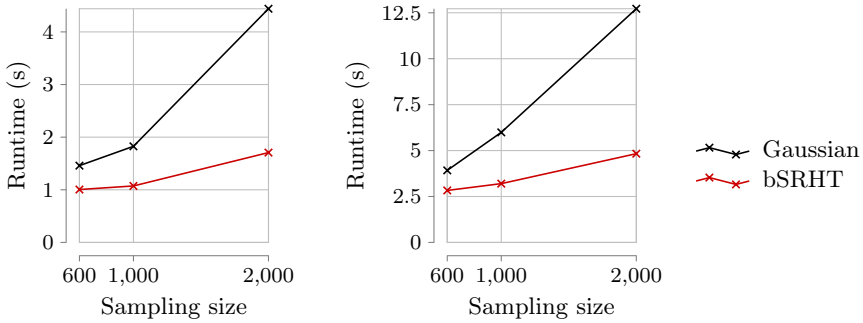
(e) Dataset `year` with $\sigma = 10^5$

Figure 2.8: Trace error $\|\mathbf{A} - [\![\mathbf{A}]\!]_k^{(\mathrm{Nyst})}\|_*/\|\mathbf{A}\|_*$ using bSRHT. The input matrix dimension is $n = 65536$.

For datasets `mnist` and `year` we observe the same behavior as in the previous figure. For datasets `botnet` and `xor` the sampling parameter does not have much

impact. Indeed, the approximation for the `xor` dataset is very precise, whereas the dataset `botnet` is approximated with a large error, for any value of l. It can be explained by looking at Figure 2.1 with the guess that the singular values profile is similar when including more records (which is what we experience in practice). In Figure 2.1 `botnet` is very low rank, explaining a very accurate approximation for any sampling size. On the contrary `xor` has a numerical rank around 2500, which is much higher than the maximum approximation of 600 in the experiment, hence a large error.

Figure 2.9 gives runtime characterization. In particular, we depict the time spent on computing $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}^{\mathrm{T}}$ and $\mathbf{\Omega}\mathbf{Y}$ in step 1 and 2 of Algorithm 6. These operations will dominate the overall computational cost, when the submatrix size is large enough. Nevertheless, the reader should be aware that TSQR and the SVD of $\mathbf{B}$ (step 4 and 5) are also important, especially when the sampling size is close to the submatrix size. The parameter k is not involved in steps 1 and 3 hence not mentioned in Figure 2.9.



(a) Matrix 32768 x 32768

(b) Matrix 65536 x 65536

Figure 2.9: Runtimes of computing $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}^{\mathrm{T}}$ and $\mathbf{\Omega}\mathbf{Y}$ in Algorithm 6 for different sampling sizes.

According to Figure 2.9, the runtime of the Gaussian sampling is up to 2.5 times higher and grows faster with l than the runtime taken by SRHT. Note that for SRHT the local computation cost is independent of l, hence the slope comes only from the reductions in steps 1 and 2. On the other hand, the Gaussian sampling involves local computations with linear dependency in l, in addition to these reductions.

2.4 Conclusion

This chapter presented a distributed adaptation of the subsampled randomized Hadamard transform. Backed up with a theorem from our paper [12], it is numerically illustrated to have the same accuracy as its sequential counterpart, as well as Gaussian sampling, the standard method. Compared to the parallel version of Gaussian sampling, it exposes a speedup of more than 2, increasing with the sampling size. Other contributions in this work present deterministic algorithms with similar goals, and the randomization is again used as comparison in the next chapter.

Chapter 3

Distributed QR decomposition with tournament pivoting

This chapter is dedicated to the computation of the truncated QRCP decomposition (see Section 1.2.4) on a distributed architecture. Unlike the randomized methods presented in the preceding chapters, the focus is now on deterministic methods. Computing this decomposition leads to obtaining a low rank approximation of the input matrix, hence, as before, improving the storage and manipulation efficiency of this matrix. Based on a previous algorithm, CARRQR [36], this work extends and generalizes this method as well as it provides numerical experiments. The reader should also note that another recent thesis implements the block truncated QRCP decomposition (see Korkmaz [79, Alg. 4]), based on the modified `dgeqp3` routine from MUMPS block low-rank solver [5, 6].

The chapter is organized as follows. Section 3.1 gives background and past work leading to 1Dc-TP, which generalizes the bounds of tournament pivoting from Demmel et al. [36] to more general reduction trees. Section 3.2 introduces 1Dr-TP, an analogous strategy that allows to select k columns from a matrix partitioned into blocks of rows. Section 3.3 introduces QRTP (2D tournament pivoting), the main algorithm of this chapter, which relies on a combination of 1Dc-TP and 1Dr-TP. We show that QRTP provides a spectrum revealing and kernel approximation of the original matrix and works well in practice. Section 3.5 presents the parallel design of QRTP as well as its estimated parallel cost. Parallel performance results from Section 3.6 compare the runtimes of QRTP and randomized QRCP [42, 86] as implemented in Xiao et al. [118], an algorithm that uses randomization to select columns during the QR factorization. We show that QRTP scales well on up to 16384 processors for a matrix of size 131072×131072.

3.1 Earlier work

This section presents the preceding work, CARRQR, a method on which our novel algorithm is based. Additionally, we present the randomized QRCP algo-

rithm used for comparison.

3.1.1 Tournament pivoting for 1D block column partitioned matrices

In the context of computing a low rank approximation of a matrix while also minimizing communication, a communication avoiding version of strong RRQR factorization is introduced in Demmel et al. [36]. It relies on a technique referred to as tournament pivoting for selecting k columns from the columns of the input matrix $\mathbf{A}$, which proceeds as following. Consider that the matrix $\mathbf{A}$ is partitioned into 4 column blocks, $\mathbf{A} = \begin{bmatrix}\mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} & \mathbf{A}_{14}\end{bmatrix}$. From each column block $\mathbf{A}_{1i}, i = 1, \ldots, 4$, k columns are selected by using strong RRQR, and their indices are given in I_{i0}.

$$\begin{array}{cccc} [\ \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} & \mathbf{A}_{14}\ ] \\ \| & \| & \| & \| \\ [\ \mathbf{Q}_{00}\mathbf{R}_{00}\mathbf{\Pi}_{00}^{\intercal} & \mathbf{Q}_{10}\mathbf{R}_{10}\mathbf{\Pi}_{10}^{\intercal} & \mathbf{Q}_{20}\mathbf{R}_{20}\mathbf{\Pi}_{20}^{\intercal} & \mathbf{Q}_{30}\mathbf{R}_{30}\mathbf{\Pi}_{30}^{\intercal}\ ] \\ \downarrow & \downarrow & \downarrow & \downarrow \\ I_{00} & I_{10} & I_{20} & I_{30} \end{array}$$

From these 4 sets of columns, the final k columns are selected through a reduce like operation, with the operator being the strong RRQR factorization. Thus, 2 sets of k columns are concatenated, and a new set of k columns is selected with strong RRQR.

$$\begin{array}{cc} [\ \mathbf{A}[:, I_{00} \cup I_{10}] & \mathbf{A}[:, I_{20} \cup I_{30}];\ ] \\ \| & \| \\ [\ \mathbf{Q}_{01}\mathbf{R}_{01}\mathbf{\Pi}_{01}^{\intercal} & \mathbf{Q}_{11}\mathbf{R}_{11}\mathbf{\Pi}_{11}^{\intercal}\ ] \\ \downarrow & \downarrow \\ I_{01} & I_{11} \end{array}$$

The final set of k columns is obtained by concatenating the two sets of k columns whose indices are in I_{01} and I_{11}, and performing strong RRQR on the obtained matrix.

$$\mathbf{A}[:, I_{01} \cup I_{11}] = \mathbf{Q}_{02}\mathbf{R}_{02}\mathbf{\Pi}_{02}^{\intercal} \ \rightarrow I_{02}$$

The indices of the selected columns are in I_{02}.

In the original algorithm, each step of the reduction operation is performed on 2 sets of columns. We extend now the analysis to the case where each reduction involves a number of p sets of columns, as presented in Algorithm 7.

Algorithm 7 1Dc-TP: Tournament pivoting for 1D column partitioned matrices (one reduction step, selection of k columns).

Require: Input matrices $\mathbf{A}_1, \ldots, \mathbf{A}_p$, approximation rank k

1: For each $\mathbf{A}_i$, compute strong RRQR to select k columns, store indices in I_i

2: Concatenate selected columns $\bar{\mathbf{A}} = \begin{bmatrix}\mathbf{A}_1[:, I_1] & \cdots & \mathbf{A}_p[:, I_p]\end{bmatrix}$

3: Compute strong RRQR of $\bar{\mathbf{A}}$ to select k columns

Ensure: indices of k rank revealing columns of $\mathbf{A}$

Theorem 8 generalizes Lemma 2.5 from Demmel et al. [36] to any number p of matrices from which k columns are selected.

Theorem 8. *Let p be a positive integer. Let $f > 1$ and $1 \leq k \leq \min(m, n, p)$. For any $1 \leq \ell \leq p$, let $\mathbf{A}_\ell \in \mathbb{R}^{m \times n}$ and $f_\ell > 1$. We note $\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \cdots & \mathbf{A}_p \end{bmatrix}$ and the strong RRQR decomposition of any $\mathbf{A}_\ell$ as*

$$\mathbf{A}_\ell \mathbf{\Pi}_\ell = \mathbf{Q}_\ell \begin{bmatrix} \mathbf{R}_{11}^\ell & \mathbf{R}_{12}^\ell \\ & \mathbf{R}_{22}^\ell \end{bmatrix},$$

where $\mathbf{R}_{11}^\ell \in \mathbb{R}^{k \times k}$ and where Π_ℓ is such that for any $1 \leq j \leq n - k$

$$\gamma_j^2({\mathbf{R}_{11}^\ell}^{-1} \mathbf{R}_{12}^\ell) + \gamma_j^2(\mathbf{R}_{22}^\ell)/\sigma_{\min}^2(\mathbf{R}_{11}^\ell) \leq k f_\ell^2.$$

Then the horizontal concatenation $\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{Q}_1 \begin{bmatrix} \mathbf{R}_{11}^1 \\ \end{bmatrix} & \cdots & \mathbf{Q}_p \begin{bmatrix} \mathbf{R}_{11}^p \\ \end{bmatrix} \end{bmatrix}$ factorized as $\tilde{\mathbf{Q}} \begin{bmatrix} \tilde{\mathbf{R}}_{11} & \bar{\mathbf{R}}_{12} \\ & \bar{\mathbf{R}}_{22} \end{bmatrix}$ with $\tilde{\mathbf{R}}_{11} \in \mathbb{R}^{k \times k}$, such that $(\tilde{\mathbf{R}}_{11}^{-1} \bar{\mathbf{R}}_{12})_{ij}^2 + \omega_i^2(\tilde{R}_{11}) \gamma_j^2(\bar{R}_{22}) \leq f^2$, verifies

$$\gamma_j^2(\tilde{\mathbf{R}}_{11}^{-1} \tilde{\mathbf{R}}_{12}) + \gamma_j^2(\tilde{\mathbf{R}}_{22})/\sigma_{\min}^2(\tilde{\mathbf{R}}_{11}) \leq 2k^3 f^2 \max(f_1^2, \ldots, f_p^2).$$

Here $\tilde{\mathbf{R}}_{12}$ and $\tilde{\mathbf{R}}_{22}$ are such that $\mathbf{A}\tilde{\mathbf{\Pi}} = \tilde{\mathbf{Q}} \begin{bmatrix} \tilde{\mathbf{R}}_{11} & \tilde{\mathbf{R}}_{12} \\ & \tilde{\mathbf{R}}_{22} \end{bmatrix}$ and $\tilde{\mathbf{\Pi}}$ is a permutation matrix.

Proof. Let $p > 1$. Consider Section 2.3.1 of Demmel et al. [36] which proves the theorem for $p = 2$. We now give the main differences to generalize it for $p > 2$. Start with p matrices $\mathbf{A}_1 \ldots \mathbf{A}_p$ and $\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_1 & \cdots & \mathbf{A}_p \end{bmatrix}$. For $1 \leq \ell \leq p$ we define for $\mathbf{A}_\ell$ the following matrices the same ways $\mathcal{N}$, $\mathcal{C}$, N, C_1 and C_2 are defined for B and $\hat{B}$ in section 2.3.1 of Demmel et al. [36].

$$\begin{aligned}
\mathbf{N}_\ell &= \left(\mathbf{R}_{11}^\ell\right)^{-1} \mathbf{R}_{12}^\ell \\
\tilde{\mathbf{N}} &= \tilde{\mathbf{R}}_{11}^{-1} \bar{\mathbf{R}}_{12} \\
\mathbf{M}_\ell &= \begin{bmatrix} \mathbf{I}_k & \tilde{\mathbf{N}} \end{bmatrix} \bar{\Pi}^\intercal[:, 1:k] \\
\mathbf{D}_\ell &= \begin{bmatrix} \mathbf{O} & \bar{\mathbf{R}}_{22} \end{bmatrix} \bar{\Pi}^\intercal[:, 1:k] \\
\begin{bmatrix} \mathbf{C}_1^\ell \\ \mathbf{C}_2^\ell \end{bmatrix} &= \tilde{\mathbf{Q}}_\ell^\intercal \mathbf{Q}_\ell \begin{bmatrix} \\ \mathbf{R}_{22}^\ell \end{bmatrix}
\end{aligned}$$

With the same reasoning as in Demmel et al. [36] we obtain

$$\begin{aligned}
\tilde{\mathbf{R}}_{12} &= \begin{bmatrix} \bar{\mathbf{R}}_{12} & \tilde{\mathbf{R}}_{11}(\mathbf{M}_1 \mathbf{N}_1 + \tilde{\mathbf{R}}_{11}^{-1} \mathbf{C}_1^1) & \cdots & \mathbf{R}_{11}(\mathbf{M}_p \mathbf{N}_p + \tilde{\mathbf{R}}_{11}^{-1} \mathbf{C}_1^p) \end{bmatrix}, \\
\tilde{\mathbf{R}}_{22} &= \begin{bmatrix} \bar{\mathbf{R}}_{22} & \mathbf{D}_1 \mathbf{N}_1 + \mathbf{C}_2^1 & \cdots & \mathbf{D}_p \mathbf{N}_p + \mathbf{C}_2^p \end{bmatrix}.
\end{aligned}$$

By adapting the second part of the proof of Demmel et al. [36] we obtain for $1 \leq \ell \leq p$,

$$\gamma_j^2(\mathbf{M}_\ell \mathbf{N}_\ell + \mathbf{R}_{11}^{-1} \mathbf{C}_1^\ell) + \gamma_j^2(\mathbf{D}_\ell \mathbf{N}_\ell + \mathbf{C}_2^\ell)/\sigma_{\min}^2(\mathbf{R}_{11}) < 2f^2 k^3 f_\ell^2.$$

By combining these two relations, we obtain the result of the theorem. □

3.1.2 Randomized QRCP

A randomized version of QRCP was introduced in the recent years in Duersch and Gu [42], Martinsson et al. [86]. This algorithm, that selects through a randomized process a set of pivot columns during the QR factorization, will be later compared to our QRTP algorithm in terms of both accuracy and performance. Randomized QRCP [118], presented in Algorithm 8, selects k pivot columns from a large matrix distributed over a grid of $P_r \times P_c$ processors in two steps. (1) Given an input matrix $\mathbf{A} \in \mathbb{R}^{mP_r \times nP_c}$, $\mathbf{A}$ is *sketched* into a matrix $B = \Omega A$, where $\Omega \in \mathbb{R}^{l \times m}$ is a $\sqrt{1/l}$ scaled matrix of independent random variables drawn from $\mathbf{M}(0, 1)$. (2) The selection of k pivot columns is obtained by computing the QRCP factorization of B involving P_c processors.

Algorithm 8 Randomized QRCP to find pivots columns.

Require: Input matrix $\mathbf{A}$ distributed on a $P_r \times P_c$ processor grid, approximation rank k, sampling parameter l

1: $B = \frac{1}{\sqrt{l}}\Omega A$, and $\Omega \in \mathbb{R}^{l \times m}$ coefficients i.i.d. from $\mathbf{M}(0, 1)$
2: Distributed truncated QRCP on $\mathbf{B}$ to select k columns

Ensure: indices of k pivot columns of $\mathbf{A}$

In the section with numerical experiments we compare the accuracy and the runtime of QRTP with RQRCP. To this end we used the Fortran/C++/MPI code provided along with Xiao et al. [118]. Several characteristics of this code are important to mention with respect to our comparison. First, RQRCP uses a block cyclic distribution, based on a block size parameter `NB`. In order to compare with QRTP, which does not use a block cyclic distribution, we set the block size to be equal to the size of the submatrix on each processor. Second, to select columns of a distributed matrix, the authors of Xiao et al. [118] modified ScaLAPACK's `pdgeqpf` to truncate the QRCP decomposition. Hence, the performance of the second step of RQRCP is directly related to the performance of ScaLAPACK. We note that the full RQRCP algorithm, as presented in Algorithm 9, computes the low rank approximation $\mathbf{A}_k = \mathbf{Q}_k\mathbf{R}_k$, where $\mathbf{Q}_k$ and $\mathbf{R}_k$ are the factors of the truncated QR decomposition from Eq. (1.1). This requires two extra steps: (3) swap selected columns in the leading positions of the distributed matrix, and (4) perform a partial QR factorization. It can also proceed with the QR factorization to some larger rank k. For the comparison presented in this work we limit ourselves to Algorithm 8.

Algorithm 9 Randomized QRCP to compute a low rank approximation.

Require: Input matrix $\mathbf{A}$ distributed on a $P_r \times P_c$ processor grid, approximation rank k, sampling parameter l

1: $\mathbf{B} = \frac{1}{\sqrt{l}}\mathbf{\Omega}\mathbf{A}$, and $\mathbf{\Omega} \in \mathbb{R}^{l \times m}$ coefficients i.i.d. from $\mathbf{M}(0, 1)$
2: Distributed truncated QRCP on $\mathbf{B}$ to select k columns
3: Swap the k to the front in $\mathbf{A}$
4: Distributed truncated QR on the pivoted $\mathbf{A}$ to compute $\mathbf{Q}_k\mathbf{R}_k$

Ensure: Low rank approximation $\mathbf{A}_k = \mathbf{Q}_k\mathbf{R}_k$

We now recall some error guarantees of RQRCP derived in Xiao et al. [118]. Considering the factorization given in Eq. (1.1), it is shown in Xiao et al. [118]

that for any permutation matrix Π and $1 \leq j \leq k$ we have:

$$\sigma_j^2(\mathbf{A}) \leq \sigma_j^2(\begin{bmatrix}\mathbf{R}_{11} & \mathbf{R}_{12}\end{bmatrix}) + \|\mathbf{R}_{22}\|_2^2. \tag{3.1}$$

In addition, when Π is obtained from RQRCP or QRCP, the following upper bound on $\|\mathbf{R}_{22}\|_2$ holds:

$$\|\mathbf{R}_{22}\|_2 \leq g_1 g_2 \sqrt{(l+1)(n-l)}\sigma_{l+1}(\mathbf{A}), \tag{3.2}$$

where $g_1 = \frac{\|\mathbf{R}_{22}\|_{1,2}}{|\alpha|}$, $g_2 = |\alpha|\|\hat{\mathbf{R}}^{-\intercal}\|_{1,2}$, $\|.\|_{1,2}$ denotes the largest column norm, and $\hat{\mathbf{R}} = \begin{pmatrix}\mathbf{R}_{11} & \mathbf{a} \\ & \alpha\end{pmatrix}$ is a leading submatrix of $\mathbf{R}$. It is also shown in Xiao et al. [118] that for QRCP, $g_1 \leq 1$ and $g_2 \leq 2^l$, while for RQRCP $g_1 \leq \sqrt{\frac{1+\varepsilon}{1-\varepsilon}}$ and $g_2 \leq \frac{\sqrt{2(1+\varepsilon)}}{1-\varepsilon}(1+\sqrt{\frac{1+\varepsilon}{1-\varepsilon}})^{l-1}$ with high probability. Here $l = k$ for QRCP, $l \geq k$ for RQRCP because of oversampling, and $\varepsilon \in (0,1)$. In other words, both QRCP and RQRCP have exponential terms in the error bounds of the singular values. Furthermore the authors indicate that computing a Π such that $\|R_{22}\|_2 \leq O(\sigma_{l+1}(\mathbf{A}))$ will reveal well the singular values of $\mathbf{A}$ in $\begin{pmatrix}\mathbf{R}_{11} & \mathbf{R}_{12}\end{pmatrix}$. Since RQRCP does not guarantee that $\|\mathbf{R}_{22}\|_2 \leq O(\sigma_{l+1}(\mathbf{A}))$, the authors introduce SRQR, an algorithm that relies on efficiently estimating g_2 and performing extra column permutations until g_2 is small. However, the extra permutations are performed on the large matrix $\mathbf{A}$ and thus this algorithm becomes less efficient. Hence, we do not consider further SRQR in our work.

3.2 Tournament pivoting for 1D block row partitioned matrices

In this section we present a new algorithm to compute a spectrum preserving and kernel approximation factorization of a block row partitioned matrix by using tournament pivoting. We use this algorithm in Section 3.3 to extend tournament pivoting to the case when a matrix is distributed over a set of processors by using a 2D partitioning of both rows and columns.

Tournament pivoting for 1D block row partitioned matrices, referred to as 1Dr-TP, selects k columns of a matrix from the selections performed on its blocks of rows using a reduction tree. It has thus similarities with the column partitioned version (1Dc-TP cf. Section 3.1.1), however now the selections are performed from subcolumns of the matrix. We present the algebra of 1Dr-TP by using a simple example in which $\mathbf{A}$ is partitioned into 4 blocks of rows, $\mathbf{A} = [\mathbf{A}_{11}; \ldots; \mathbf{A}_{41}]$. First k columns are selected from each block of rows $\mathbf{A}_{i1}, i = 1, \ldots, 4$ and their indices are stored in $I_{i0}, i = 1, \ldots, 4$,

$$\mathbf{A} = \begin{pmatrix}\mathbf{A}_{11} \\ \mathbf{A}_{21} \\ \mathbf{A}_{31} \\ \mathbf{A}_{41}\end{pmatrix} = \begin{pmatrix}\mathbf{Q}_{00}\mathbf{R}_{00}\mathbf{\Pi}_{00}^{-1} \\ \mathbf{Q}_{10}\mathbf{R}_{10}\mathbf{\Pi}_{10}^{-1} \\ \mathbf{Q}_{20}\mathbf{R}_{20}\mathbf{\Pi}_{20}^{-1} \\ \mathbf{Q}_{30}\mathbf{R}_{30}\mathbf{\Pi}_{30}^{-1}\end{pmatrix} \begin{matrix}\rightarrow \text{select k cols } I_{00} \\ \rightarrow \text{select k cols } I_{10} \\ \rightarrow \text{select k cols } I_{20} \\ \rightarrow \text{select k cols } I_{30}\end{matrix}$$

Then the sets of k indices are combined two by two to select each time a new set of k indices. For example, for the first two sets, I_{00} and I_{10}, the selection

is performed as following. The columns of the first two block rows $[\mathbf{A}_{11}; \mathbf{A}_{21}]$ whose indices belong to $I_{00} \cup I_{10}$ are concatenated together to form a new matrix, $[\mathbf{A}_{11}; \mathbf{A}_{21}][:, I_{00} \cup I_{10}]$. Strong RRQR is applied to this matrix to select a new set of its columns I_{01},

$$\left[\begin{array}{c} \begin{bmatrix}\mathbf{A}_{11}\\ \mathbf{A}_{21}\end{bmatrix}[:, I_{00} \cup I_{10}] \\ \hline \begin{bmatrix}\mathbf{A}_{31}\\ \mathbf{A}_{41}\end{bmatrix}[:, I_{20} \cup I_{30}] \end{array}\right] = \begin{bmatrix}\mathbf{Q}_{01}\mathbf{R}_{01}\Pi_{01}^{-1}\\ \mathbf{Q}_{11}\mathbf{R}_{11}\Pi_{11}^{-1}\end{bmatrix} \begin{array}{l}\rightarrow I_{01}\\ \rightarrow I_{11}\end{array}.$$

In the last step, the columns of $\mathbf{A}$ whose indices belong to $I_{20} \cup I_{30}$ are concatenated together, and the final k columns are selected through strong RRQR from $\mathbf{A}[:, I_{01} \cup I_{11}]$,

$$\mathbf{A}[:, I_{01} \cup I_{11}] = \mathbf{Q}_{02}\mathbf{R}_{02}\mathbf{\Pi}_{02}^{-1} \rightarrow I_{02}.$$

Algorithm 10 describes one reduction of 1Dr-TP, i.e. selects k columns from $\mathbf{A} = [\mathbf{A}_1; \ldots; \mathbf{A}_p]$ through local selections of k indices from subcolumns of $\mathbf{A}$ in $\mathbf{A}_1, \ldots, \mathbf{A}_p$.

Algorithm 10 1Dr-TP: Tournament pivoting for row partitioned matrix (one reduction step, selection of k columns).

Require: Input matrices $\mathbf{A}_1 \ldots \mathbf{A}_p$, rank of approximation b
1: For each $\mathbf{A}_i$, compute strong RRQR to select k columns, store indices in I_i
2: Concatenate selected columns $\bar{\mathbf{A}} = \begin{bmatrix}\mathbf{A}[:, I_1] & \cdots & \mathbf{A}[:, I_p]\end{bmatrix}$
3: Compute strong RRQR of $\bar{\mathbf{A}}$ to select k columns
Ensure: indices of k rank revealing columns of $\mathbf{A}$

Similarly to Theorem 8 we derive bounds on the quality of the approximation in the sense of Theorem 3 from bounds corresponding to the selection at each step of the tournament.

Theorem 9. *Let p be any strictly positive integer. Let $f > 1$ and $1 \leq k \leq \min(m, n/p)$. For any $1 \leq \ell \leq p$, let $\mathbf{A}_\ell \in \mathbb{R}^{m \times n}$ and $f_\ell > 1$. We note $\mathbf{A} = \begin{bmatrix}\mathbf{A}_1; \ldots; \mathbf{A}_p\end{bmatrix}$ and the strong RRQR decomposition of any $\mathbf{A}_\ell$ as,*

$$\mathbf{A}_\ell\mathbf{\Pi}_\ell = \mathbf{Q}_\ell \begin{bmatrix}\mathbf{R}_{1,\ell}^\ell & \mathbf{C}_\ell\\ & \mathbf{D}_\ell\end{bmatrix}, \tag{3.3}$$

where $\mathbf{R}_{1,\ell}^\ell \in \mathbb{R}^{k \times k}$ and where $\mathbf{\Pi}_\ell$ is such that for any $1 \leq j \leq n-k$,

$$\gamma_j^2({\mathbf{R}_{1,\ell}^\ell}^{-1}\mathbf{C}_\ell) + \gamma_j^2(\mathbf{D}_\ell)/\sigma_{\min}^2(\mathbf{R}_{1,\ell}^\ell) \leq k f_\ell^2. \tag{3.4}$$

Then the horizontal concatenation $\bar{\mathbf{A}} = \begin{bmatrix}\mathbf{A}\mathbf{\Pi}_1[:, 1:k] & \cdots & \mathbf{A}\mathbf{\Pi}_p[:, 1:k]\end{bmatrix}$ factorized as $\tilde{\mathbf{Q}}\begin{bmatrix}\tilde{\mathbf{R}}_{11} & \bar{\mathbf{R}}_{12}\\ & \bar{\mathbf{R}}_{22}\end{bmatrix}$ with $\tilde{\mathbf{R}}_{11} \in \mathbb{R}^{k \times k}$, such that $(\tilde{\mathbf{R}}_{11}^{-1}\bar{\mathbf{R}}_{12})_{ij}^2 + \omega_i^2(\tilde{\mathbf{R}}_{11})\gamma_j^2(\bar{\mathbf{R}}_{22}) \leq f^2$, verifies

$$\gamma_j^2(\tilde{\mathbf{R}}_{11}^{-1}\tilde{\mathbf{R}}_{12}) + \gamma_j^2(\tilde{\mathbf{R}}_{22})/\sigma_{\min}^2(\tilde{\mathbf{R}}_{11}) \leq 2k^3 f^2 \sum_{i=1}^{p} f_i^2,$$

where $\tilde{\mathbf{R}}_{12}$ *and* $\tilde{\mathbf{R}}_{22}$ *are such that* $\mathbf{A}\tilde{\mathbf{\Pi}} = \tilde{\mathbf{Q}} \begin{bmatrix} \tilde{\mathbf{R}}_{11} & \tilde{\mathbf{R}}_{12} \\ & \tilde{\mathbf{R}}_{22} \end{bmatrix}$ *and* $\tilde{\mathbf{\Pi}}$ *is a permutation matrix.*

Proof. Under the same hypothesis as Theorem 9 and $\mathbf{\Pi}$ a permutation matrix such that

$$\mathbf{\Pi} = \begin{bmatrix} \mathbf{\Pi}_1[:, 1:k] & \cdots & \mathbf{\Pi}_p[:, 1:k] & \mathbf{\Pi}_{p+1} \end{bmatrix},$$

where $\mathbf{\Pi}_{p+1}$ selects the remaining columns so that $\mathbf{\Pi}$ is orthogonal. For the sake of clarity, blocks are supposed to reveal non overlapping sets of columns, *i.e.* $\forall \ell' \neq \ell, \mathbf{\Pi}_\ell[:, 1:k]^\intercal \mathbf{\Pi}_{\ell'}[:, 1:k] = \mathbf{O}$. Using $\mathbf{\Pi}$ we can rewrite the strong rank revealing factorization 3.3 as,

$$\mathbf{A}_\ell \mathbf{\Pi} = \mathbf{Q}_\ell \begin{bmatrix} \mathbf{R}^\ell_{1,1} & \cdots & \mathbf{R}^\ell_{1,p} & \mathbf{R}^\ell_{1,p+1} \\ \mathbf{R}^\ell_{2,1} & \cdots & \mathbf{R}^\ell_{2,p} & \mathbf{R}^\ell_{2,p+1} \end{bmatrix},$$

with $\mathbf{R}^\ell_{2,\ell} = \mathbf{O}$, $\mathbf{R}^\ell_{1,1}, \ldots, \mathbf{R}^\ell_{1,p} \in \mathbb{R}^{k \times k}$, $\mathbf{R}^\ell_{2,1}, \ldots, \mathbf{R}^\ell_{2,p} \in \mathbb{R}^{(m-k) \times k}$, $\mathbf{R}^\ell_{1,p+1} \in \mathbb{R}^{k \times (n-kp)}$ and $\mathbf{R}^\ell_{2,p+1} \in \mathbb{R}^{(m-k) \times (n-kp)}$.

The condition 3.4 on the factorization of each $\mathbf{A}_\ell$ can be reformulated with the alternative permutation $\mathbf{\Pi}$ as the following. For $1 \leq \ell' \leq p+1$ so that $\ell' \neq \ell$ and j is a column index of $\mathbf{R}^\ell_{1,\ell'}$,

$$\gamma_j^2({\mathbf{R}^\ell_{1,\ell}}^{-1} \mathbf{R}^\ell_{1,\ell'}) + \gamma_j^2(\mathbf{R}^\ell_{2,\ell'}) / \sigma^2_{\min}(\mathbf{R}^\ell_{1,\ell}) \leq k f_\ell^2.$$

Recalling the hypothesis of Theorem 9, $\bar{\mathbf{A}} = \mathbf{A} \begin{bmatrix} \mathbf{\Pi}_1[:, 1:k] & \cdots & \mathbf{\Pi}_p[:, 1:k] \end{bmatrix} = \mathbf{A}\mathbf{\Pi}[:, 1:kp] \in \mathbb{R}^{mp \times kp}$ contains the columns of $\mathbf{A}$ selected on each block $\mathbf{A}_\ell$.

The RRQR factorization of $\bar{\mathbf{A}}$ is defined as,

$$\bar{\mathbf{A}}\bar{\mathbf{\Pi}} = \tilde{\mathbf{Q}} \begin{bmatrix} \tilde{\mathbf{R}}_{11} & \bar{\mathbf{R}}_{12} \\ & \bar{\mathbf{R}}_{22} \end{bmatrix},$$

where $\tilde{\mathbf{Q}} \in \mathbb{R}^{mp \times mp}$ and $\tilde{\mathbf{R}}_{11} \in \mathbb{R}^{k \times k}$ such that for any $1 \leq i \leq k$ and $1 \leq j \leq k(p-1)$

$$(\tilde{\mathbf{R}}_{11}^{-1} \bar{\mathbf{R}}_{12})_{ij}^2 + \omega_i^2(\tilde{\mathbf{R}}_{11}) \gamma_j^2(\bar{\mathbf{R}}_{22}) \leq f^2 \tag{3.5}$$

Finally, the global approximation can be expressed as

$$\mathbf{A}\tilde{\mathbf{\Pi}} = \tilde{\mathbf{Q}} \begin{bmatrix} \begin{bmatrix} \tilde{\mathbf{R}}_{11} & \bar{\mathbf{R}}_{12} \\ & \bar{\mathbf{R}}_{22} \end{bmatrix}, & \tilde{\mathbf{Q}}^\intercal \begin{bmatrix} \mathbf{Q}_1 \begin{bmatrix} \mathbf{R}^1_{1,p+1} \\ \mathbf{R}^1_{2,p+1} \end{bmatrix} \\ \vdots \\ \mathbf{Q}_p \begin{bmatrix} \mathbf{R}^p_{1,p+1} \\ \mathbf{R}^p_{2,p+1} \end{bmatrix} \end{bmatrix} \end{bmatrix} =_{def} \tilde{\mathbf{Q}} \begin{bmatrix} \tilde{\mathbf{R}}_{11} & \tilde{\mathbf{R}}_{12} \\ & \tilde{\mathbf{R}}_{22} \end{bmatrix}$$

We will now express $\tilde{\mathbf{R}}_{12}$ and $\tilde{\mathbf{R}}_{22}$. Let $\tilde{\mathbf{Q}} = \begin{bmatrix} \tilde{\mathbf{Q}}_1; \ldots; \tilde{\mathbf{Q}}_p \end{bmatrix}$, where $\forall \ell \leq p, \tilde{\mathbf{Q}}_\ell \in \mathbb{R}^{m \times mp}$ leading to,

$$\tilde{\mathbf{Q}}^\intercal \begin{bmatrix} \mathbf{Q}_1 \begin{bmatrix} \mathbf{R}^1_{1,p+1} \\ \mathbf{R}^1_{2,p+1} \end{bmatrix} \\ \vdots \\ \mathbf{Q}_p \begin{bmatrix} \mathbf{R}^p_{1,p+1} \\ \mathbf{R}^p_{2,p+1} \end{bmatrix} \end{bmatrix} = \sum_{\ell=1}^{p} \tilde{\mathbf{Q}}_\ell^\intercal \mathbf{Q}_\ell \begin{bmatrix} \mathbf{R}^\ell_{1,p+1} \\ \mathbf{R}^\ell_{2,p+1} \end{bmatrix}.$$

We can express, with $\mathbf{N}_\ell = {\mathbf{R}^\ell_{1,\ell}}^{-1}\mathbf{R}^\ell_{1,p+1}$,

$$\tilde{\mathbf{Q}}_\ell^\intercal \mathbf{Q}_\ell \begin{bmatrix} \mathbf{R}^\ell_{1,p+1} \\ \mathbf{R}^\ell_{2,p+1} \end{bmatrix} = \tilde{\mathbf{Q}}_\ell^\intercal \mathbf{Q}_\ell \begin{bmatrix} \mathbf{R}^\ell_{1,\ell} \\ \end{bmatrix} \mathbf{N}_\ell + \tilde{\mathbf{Q}}_\ell^\intercal \mathbf{Q}_\ell \begin{bmatrix} \\ \mathbf{R}^\ell_{2,p+1} \end{bmatrix}. \tag{3.6}$$

To ease the selection notation, let $I_\ell = [(\ell-1)k+1 : \ell k]$ be the indices of the columns of $\bar{\mathbf{A}}$ corresponding to the selected columns of $\mathbf{A}_\ell$. From the definition of $\bar{\mathbf{A}}$,

$$\begin{aligned} \mathbf{Q}_\ell \begin{bmatrix} \mathbf{R}^\ell_{1,\ell} \\ \end{bmatrix} &= \tilde{\mathbf{Q}}_\ell \begin{bmatrix} \tilde{\mathbf{R}}_{11} & \bar{\mathbf{R}}_{12} \\ & \bar{\mathbf{R}}_{22} \end{bmatrix} \bar{\mathbf{\Pi}}^\intercal[:, I_\ell] \\ &= \tilde{\mathbf{Q}}_\ell \begin{bmatrix} \tilde{\mathbf{R}}_{11}\mathbf{M}_\ell \\ \mathbf{D}_\ell \end{bmatrix}, \end{aligned}$$

with

$$\begin{aligned} \mathbf{M}_\ell &= \begin{bmatrix} \mathbf{I}_k & \tilde{\mathbf{N}} \end{bmatrix} \bar{\mathbf{\Pi}}^\intercal[:, I_\ell], \\ \mathbf{D}_\ell &= \begin{bmatrix} \mathbf{O} & \bar{\mathbf{R}}_{22} \end{bmatrix} \bar{\mathbf{\Pi}}^\intercal[:, I_\ell], \\ \tilde{\mathbf{N}} &= \tilde{\mathbf{R}}_{11}^{-1}\bar{\mathbf{R}}_{12}. \end{aligned}$$

Note that the new introduced matrices verify for $1 \le i \le k$ and $1 \le j \le k$,

$$(\mathbf{M}_\ell)^2_{i,j} + \gamma_j^2(\mathbf{D}_\ell)\omega_i^2(\tilde{\mathbf{R}}_{11}) \le f^2. \tag{3.7}$$

We also define,

$$\tilde{\mathbf{Q}}_\ell^\intercal \mathbf{Q}_\ell \begin{bmatrix} \\ \mathbf{R}^\ell_{2,p+1} \end{bmatrix} =_{def} \begin{bmatrix} \mathbf{C}^\ell_1 \\ \mathbf{C}^\ell_2 \end{bmatrix}.$$

Plugging everything using the global approximation expression, we obtain,

$$\tilde{\mathbf{Q}}^\intercal \begin{bmatrix} \mathbf{Q}_1 \begin{bmatrix} \mathbf{R}^1_{1,p+1} \\ \mathbf{R}^1_{2,p+1} \end{bmatrix} \\ \vdots \\ \mathbf{Q}_p \begin{bmatrix} \mathbf{R}^p_{1,p+1} \\ \mathbf{R}^p_{2,p+1} \end{bmatrix} \end{bmatrix} = \sum_{\ell=1}^{p} \left(\begin{bmatrix} \tilde{\mathbf{R}}_{11}\mathbf{M}_\ell \\ \mathbf{D}_\ell \end{bmatrix} \mathbf{N}_\ell + \begin{bmatrix} \mathbf{C}^\ell_1 \\ \mathbf{C}^\ell_1 \end{bmatrix} \right),$$

and then

$$\begin{aligned} \tilde{\mathbf{R}}_{12} &= \begin{bmatrix} \bar{\mathbf{R}}_{12} & \tilde{\mathbf{R}}_{11} \sum_{\ell=1}^{p} \left(\mathbf{M}_\ell\mathbf{N}_\ell + \tilde{\mathbf{R}}_{11}^{-1}\mathbf{C}^\ell_1 \right) \end{bmatrix}, \\ \tilde{\mathbf{R}}_{22} &= \begin{bmatrix} \bar{\mathbf{R}}_{22} & \sum_{\ell=1}^{p} \left(\mathbf{D}_\ell\mathbf{N}_\ell + \mathbf{C}^\ell_2 \right) \end{bmatrix}. \end{aligned}$$

We are looking for an upper bound of the following expression

$$\gamma_j^2(\tilde{\mathbf{R}}_{11}^{-1}\tilde{\mathbf{R}}_{12}) + \gamma_j^2(\tilde{\mathbf{R}}_{22})/\sigma^2_{\min}(\tilde{\mathbf{R}}_{11}). \tag{3.8}$$

Case 1 If $1 \le j \le k$, then $\gamma_j^2(\tilde{\mathbf{R}}_{11}^{-1}\tilde{\mathbf{R}}_{12}) = \gamma_j^2(\tilde{\mathbf{R}}_{11}^{-1}\bar{\mathbf{R}}_{12})$ and $\gamma_j^2(\tilde{\mathbf{R}}_{22}) = \gamma_j^2(\bar{\mathbf{R}}_{22})$. So we can use (3.5) which gives (3.8) $< kf^2$.

Case 2 Assume $k+1 \leq j \leq n-kp$. We have,

$$\begin{aligned}(3.8) &= \gamma_j^2\left(\sum_{\ell=1}^{p}\left(\mathbf{M}_\ell\mathbf{N}_\ell+\tilde{\mathbf{R}}_{11}^{-1}\mathbf{C}_1^\ell\right)\right)+\gamma_j^2\left(\sum_{\ell=1}^{p}\left(\mathbf{D}_\ell\mathbf{N}_\ell+\mathbf{C}_2^\ell\right)/\sigma_{\min}^2(\tilde{\mathbf{R}}_{11})\right)\\ &= \gamma_j^2\left(\sum_{\ell=1}^{p}\left(\begin{bmatrix}\mathbf{M}_\ell\mathbf{N}_\ell\\ \mathbf{D}_\ell\mathbf{N}_\ell/\sigma_{\min}(\tilde{\mathbf{R}}_{11})\end{bmatrix}+\begin{bmatrix}\tilde{\mathbf{R}}_{11}^{-1}\mathbf{C}_1^\ell\\ \mathbf{C}_2^\ell/\sigma_{\min}(\tilde{\mathbf{R}}_{11})\end{bmatrix}\right)\right)\\ &\leq 2p\sum_{\ell=1}^{p}\left(\gamma_j^2\left(\begin{bmatrix}\mathbf{M}_\ell\mathbf{N}_\ell\\ \mathbf{D}_\ell\mathbf{N}_\ell/\sigma_{\min}(\tilde{\mathbf{R}}_{11})\end{bmatrix}\right)+\gamma_j^2\left(\begin{bmatrix}\tilde{\mathbf{R}}_{11}^{-1}\mathbf{C}_1^\ell\\ \mathbf{C}_2^\ell/\sigma_{\min}(\tilde{\mathbf{R}}_{11})\end{bmatrix}\right)\right).\end{aligned}$$

On one hand we can say using the relaxed form (cf. Eq. (1.3)) of (3.7) that for any $1 \leq \ell \leq p$,

$$\gamma_j^2\left(\begin{bmatrix}\mathbf{M}_\ell\mathbf{N}_\ell\\ \mathbf{D}_\ell\mathbf{N}_\ell/\sigma_{\min}(\tilde{\mathbf{R}}_{11})\end{bmatrix}\right) \leq \left\|\begin{bmatrix}\mathbf{M}_\ell\\ \mathbf{D}_\ell/\sigma_{\min}(\tilde{\mathbf{R}}_{11})\end{bmatrix}\right\|_F^2\gamma_j^2(\mathbf{N}_\ell) \leq f^2k^2\gamma_j^2(\mathbf{N}_\ell). \tag{3.9}$$

On other hand using in a similar manner as Eq. (3.9) that for any $x \in \mathbb{R}^k$, $\|\tilde{\mathbf{R}}_{11}^{-1}x\| \leq \sigma_{\max}(\tilde{\mathbf{R}}_{11}^{-1})\|x\| = \sigma_{\min}(\tilde{\mathbf{R}}_{11})\|x\|$,

$$\gamma_j^2\left(\begin{bmatrix}\tilde{\mathbf{R}}_{11}^{-1}\mathbf{C}_1^\ell\\ \mathbf{C}_2^\ell/\sigma_{\min}(\tilde{\mathbf{R}}_{11})\end{bmatrix}\right) \leq \gamma_j^2\left(\begin{bmatrix}\mathbf{C}_1^\ell\\ \mathbf{C}_2^\ell\end{bmatrix}\right)/\sigma_{\min}^2(\tilde{\mathbf{R}}_{11}).$$

Furthermore, using definition of $\mathbf{C}_1^\ell$ and $\mathbf{C}_2^\ell$,

$$\sum_{\ell=1}^{p}\gamma_j^2\left(\begin{bmatrix}\mathbf{C}_1^\ell\\ \mathbf{C}_2^\ell\end{bmatrix}\right)=\gamma_j^2\left(\begin{bmatrix}\begin{bmatrix}\mathbf{C}_{1,1}^1\\ \mathbf{C}_{2,1}^1\end{bmatrix}\\ \vdots\\ \begin{bmatrix}\mathbf{C}_{1,p}^p\\ \mathbf{C}_{2,p}^p\end{bmatrix}\end{bmatrix}\right)=\gamma_j^2\left(\begin{bmatrix}\mathbf{R}_{2,p+1}^1\\ \vdots\\ \mathbf{R}_{2,p+1}^p\end{bmatrix}\right)=\sum_{\ell=1}^{p}\gamma_j^2\left(\mathbf{R}_{2,p+1}^\ell\right). \tag{3.10}$$

We can use Eq. (3.9) and Eq. (3.10) to derive further Eq. (3.8).

$$(3.8) \leq 2p\sum_{\ell=1}^{p}\left(f^2k^2\gamma_j^2(\mathbf{N}_\ell)+\frac{\gamma_j^2(\mathbf{R}_{2,p+1}^\ell)}{\sigma_{\min}^2(\tilde{\mathbf{R}}_{11})}\right).$$

Firstly considering that for any $1 \leq \ell \leq p$,

$$\sigma_{\min}^2\left(\begin{bmatrix}\tilde{\mathbf{R}}_{11}\mathbf{M}_\ell\\ \mathbf{D}_\ell\end{bmatrix}\right) \leq \sigma_{\min}^2\left(\tilde{\mathbf{R}}_{11}\right)\|\mathbf{M}_\ell\|_2^2+\|\mathbf{D}_\ell\|_2^2 \leq f^2k^2\sigma_{\min}^2(\tilde{\mathbf{R}}_{11}),$$

secondly with the interlacing singular values theorem (see Theorem 1),

$$\sigma_{\min}^2(\mathbf{R}_{1,\ell}^\ell) \leq \sigma_{\min}^2\left(\bar{\mathbf{A}}[:,I_\ell]\right),$$

and the fact that

$$\begin{bmatrix}\mathbf{Q}_1 & & \\ & \ddots & \\ & & \hat{\mathbf{Q}}_p\end{bmatrix}\mathbf{A}[:,I_\ell]=\tilde{\mathbf{Q}}\begin{bmatrix}\tilde{\mathbf{R}}_{11}\mathbf{M}_\ell\\ \mathbf{D}_\ell\end{bmatrix},$$

we obtain

$$\sigma_{\min}^2(\mathbf{R}_{1,\ell}^{\ell}) \leq f^2 k^2 \sigma_{\min}^2(\tilde{\mathbf{R}}_{11}).$$

Using these two relations we continue to derive Eq. (3.8),

$$\begin{aligned} (3.8) &\leq 2pf^2k^2 \sum_{\ell=1}^{p} \left(\gamma_j^2(\mathbf{N}_\ell) + \frac{\gamma_j^2(\mathbf{R}_{2,p+1}^{\ell})}{\sigma_{\min}^2(\mathbf{R}_{1,\ell}^{\ell})} \right) \\ &\leq 2pf^2k^3 \sum_{\ell=1}^{p} f_\ell^2. \end{aligned}$$

□

3.3 QR factorization with 2D tournament pivoting

In this section we introduce an algorithm for computing a low rank approximation of a matrix distributed over a 2D grid of processors by using tournament pivoting. We refer to this pivoting strategy as 2D tournament pivoting, or 2D TP, which combines the two pivoting strategies introduced in the previous sections, 1Dc-TP for matrices partitioned into blocks of columns and 1Dr-TP for matrices partitioned into blocks of rows.

3.3.1 QRTP algorithm

We consider a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ distributed on a $P_r \times P_c$ grid of processors. We explain the algorithm considering a 2×4 grid of processors, that is $\mathbf{A}$ is partitioned as,

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{A}_{13} & \mathbf{A}_{14} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{A}_{23} & \mathbf{A}_{24} \end{pmatrix}.$$

We consider here that tournament pivoting relies on a binary tree. The general case will be covered in a following section. First, k columns are selected from each column block by using binary 1Dr-TP.

$$\begin{array}{cccc} \begin{pmatrix} \mathbf{A}_{11} \\ \mathbf{A}_{21} \end{pmatrix} & \begin{pmatrix} \mathbf{A}_{12} \\ \mathbf{A}_{22} \end{pmatrix} & \begin{pmatrix} \mathbf{A}_{13} \\ \mathbf{A}_{23} \end{pmatrix} & \begin{pmatrix} \mathbf{A}_{14} \\ \mathbf{A}_{24} \end{pmatrix} \\ \downarrow & \downarrow & \downarrow & \downarrow \\ I_{00} & I_{10} & I_{20} & I_{30} \end{array}$$

Second, binary 1Dc-TP is applied on the sets of k selected columns to obtain the final k columns.

$$\begin{array}{cccc} \mathbf{A}(:, I_{00}) & \mathbf{A}(:, I_{10}) & \mathbf{A}(:, I_{20}) & \mathbf{A}(:, I_{30}) \\ & & \downarrow & \\ & & I_{02} & \end{array}$$

Figure 3.1 illustrates this algebra and Algorithm 11 gives the formal procedure for computing the QR factorization of a matrix with 2D tournament pivoting, referred to as QRTP.

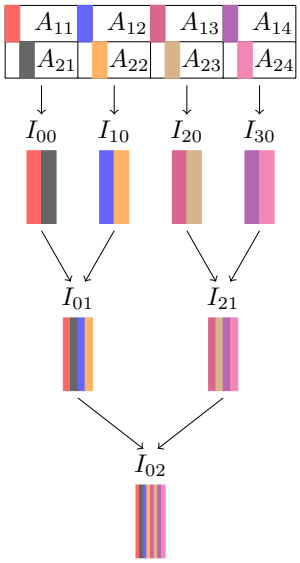


Figure 3.1: Column selection using Algorithm 11. A set of k columns are selected for each submatrix $\mathbf{A}_{ij}$, then these columns are combined along a reduction tree to find k columns for the complete matrix $\mathbf{A}$. I represents a set of indices.

Algorithm 11 QRTP: QR factorization with 2D tournament pivoting.

Require: $P_r \times P_c$ processor grid, $\mathbf{A}$ distributed matrix, k approximation rank

1: **for** $1 \leq i \leq P_r$ in parallel **do**
2: $\quad I_i \leftarrow$ select k columns from each block column i using binary 1Dr-TP
3: **end for**
4: $I \leftarrow$ Select k columns from $\mathbf{A}(:, \cup_{i=1}^{P_r} I_i)$ using binary 1Dc-TP

Ensure: I indices of k rank revealing columns of $\mathbf{A}$

We now give the bounds for the approximations of the singular values obtained by QRTP. We assume that each selection of k columns in the algorithm is performed such that the bounds in Eq. (1.2) are satisfied with a constant f. Let $P = P_r P_c$. Then with

$$f_{TP} = \sqrt{P} P_r k^{\log_2(P)} f^{\log_2(P)+1}, \tag{3.11}$$

and by using Theorem 4, it can be shown that the singular values of $\tilde{\mathbf{R}}_{11}$ and $\tilde{\mathbf{R}}_{22}$ approximate the singular values of $\mathbf{A}$ in the following sense: for $1 \leq j \leq n-k$ and $1 \leq i \leq k$,

$$1 \leq \frac{\sigma_i(\mathbf{A})}{\sigma_i(\tilde{\mathbf{R}}_{11})} \leq \sqrt{1 + k f_{TP}^2 (n-k)}, 1 \leq \frac{\sigma_j(\tilde{\mathbf{R}}_{22})}{\sigma_{j+k}(\mathbf{A})} \leq \sqrt{1 + k f_{TP}^2 (n-k)}. \tag{3.12}$$

The steps to find these bounds are detailed in Section 3.3.2 in a more general setting.

We compare now the bounds of QRTP with the bounds of RQRCP from Eq. (3.2). For a given $i \leq n-k$, the bound of $\|\mathbf{R}_{22}\|_2$ can be compared by contrasting $g_1^2 g_2^2 (l+1)(n-l)$ for RQRCP to $k f_{TP}^2 (n-k)$ for QRTP. The terms g_1 and g_2 are defined in Xiao et al. [118] and recalled in Section 3.1.2. In particular, g_2 depends exponentially on l for the RQRCP bound, whereas the QRTP bound depends exponentially on $\log_2 P$.

3.3.2 Spectrum preserving and kernel approximation properties of QRTP

We discuss in this section the spectrum preserving and kernel approximation properties of QRTP by considering more general reduction trees. We also compare the reduction strategy used in QRTP with an alternative one in terms of approximation bounds.

Spectrum preserving and kernel approximation properties of 1Dc-TP and 1Dr-TP

We discuss first these two properties for both 1Dc-TP and 1Dr-TP. Let $\mathbf{A}$ be partitioned into P_c blocks of columns and let $\mathbf{A}_i$ be the ith block. Let k be the rank of all approximations. Consider that the selection of k columns from each block $\mathbf{A}_i$ satisfies inequality Eq. (1.3) with the upper bound being F_i. Consider the h^{th} reduction step out of D reduction steps of the algorithm and let f_i^h be the bound corresponding to the selection of k columns at this reduction step, and let x_h be the number of column sets implied in this reduction. Using induction on Theorem 8, we obtain that the selection of k columns from $\mathbf{A}$ leads to a factorization based on 1Dc-TP that satisfies Eq. (1.3) with the upper bound being

$$F_c^2 = (2k^2)^D \max_{1 \leq i \leq P_c} \left(F_i^2 \prod_{h=1}^{D} \left(f_i^h\right)^2 \right). \tag{3.13}$$

This expression relates to Corollary 2.6 and 2.7 in Demmel et al. [36] by assuming that all approximations enforce the same bound f in inequality Eq. (1.3), i.e. for $1 \leq i \leq P_c$ and $1 \leq h \leq D$, $F_i = \sqrt{k} f$ and $f_i^h = f$.

Now assume that $\mathbf{A}$ is partitioned into P_r blocks of rows. With the same conventions as for Eq. (3.13), using induction on Theorem 9, the approximation obtained by selecting k columns from $\mathbf{A}$ using 1Dr-TP satisfies Eq. (1.3) with the upper bound being

$$F_r^2 = (2k^2)^D \sum_{1 \leq i \leq N} \left(F_i^2 \prod_{h=1}^{D} x_h (f_i^h)^2 \right). \tag{3.14}$$

Row-first and column-first strategies

When considering a matrix partitioned into $P_r \times P_c$ blocks, the choice of reduction tree impacts the accuracy and the performance of QRTP. In Section 3.3 we described 2D TP by using first 1Dr-TP followed by 1Dc-TP, we refer to this strategy as row-first strategy. Additionally, we define the column-first strategy as 1Dc-TP followed by 1Dr-TP, and we show in this section that given any partitioning of $\mathbf{A}$, the row-first strategy has better theoretical approximation bounds than the column-first strategy. There can be more complex ways to combine row and column reductions, but we do not explore them in this work.

Let $\mathbf{A}$ be a matrix partitioned into $P_r \times P_c$ blocks and $\mathbf{A}_{i,j}$ be the block at indices (i, j) in the partitioned matrix. Let k be the rank of all approximations. Let $F_{i,j}$ be the bound associated with the selection of k columns from $\mathbf{A}_{i,j}$ in the sense of Eq. (1.3). We assume that 1Dr-TP is composed of D_r reduction steps and 1Dc-TP has D_c reduction steps. Let $f_{i,j}^h$ be the bounds associated with the h^{th} reduction step of the partition $\mathbf{A}_{i,j}$. Replacing each F_i in Eq. (3.13) with F_r from Eq. (3.14), we obtain the following bounds in the sense of Eq. (1.2) for the row-first strategy,

$$F_{rc}^2 = (2k^2)^{D_r + D_c} \max_{1 \leq j \leq P_c} \left(\sum_{i=1}^{P_r} \left[F_{i,j}^2 \prod_{h=1}^{D_r + D_c} (f_{i,j}^h)^2 \prod_{h=1}^{D_r} x_i \right] \right). \tag{3.15}$$

Replacing each F_i in Eq. (3.14) with F_c from Eq. (3.13), we have the following bounds in the sense of Eq. (1.2) for the column-first strategy,

$$F_{cr}^2 = (2k^2)^{D_r + D_c} \sum_{i=1}^{P_r} \left[\max_{1 \leq j \leq P_c} \left(F_{i,j}^2 \prod_{h=1}^{D_c + D_r} (f_{i,j}^h)^2 \prod_{h=D_c+1}^{D_c + D_r} x_i \right) \right]. \tag{3.16}$$

We conclude with the following corollary.

Corollary 1. *Let $\mathbf{A}$ be a matrix partitioned into $P_r \times P_c$ blocks for which a low rank approximation based on QRTP is computed using 1Dr-TP and 1Dc-TP with given reduction trees. Let F_{rc} resp. F_{cr} be the bounds in the sense of Eq.* (1.2) *obtained when executing QRTP with 1Dr-TP followed by 1Dc-TP resp. 1Dc-TP followed by 1Dr-TP. We have the following relation on the bounds,*

$$F_{rc} \leq F_{cr}.$$

Corollary 1 shows that once the reduction trees are fixed for 1Dr-TP and 1Dc-TP, the row-first strategy always has a smaller lower bound than the column-first strategy. In more details, it means that given a matrix partitioned into $P_r \times P_c$ blocks for which a low rank approximation is computed using QRTP,

the guarantees of the low rank approximations in the sense of Theorem 3 are better when reducing first along the first dimension (1Dr-TP to select P_c subsets of columns, then 1Dc-TP to select k columns) than along the second dimension (1Dc-TP to select P_r subsets of columns, then 1Dr-TP to select k columns).

3.4 Numerical results

In this section we study the numerical behavior of QRTP on matrices of small size. The parallel performance of the algorithm on large matrices is studied later in Section 3.6. We consider first the following two matrices,

- `heat` is a 1000×1000 matrix modeling an inverse heat equation[22],
- `gravity` is a 1000×1000 matrix modeling a gravity problem[116].

Let $\mathbf{A}$ be one of these two matrices and $\mathbf{A}_{QRTP}$ its low rank approximation computed with QRTP. Figure 3.2 displays the first 50 singular values of the matrix `heat` and their approximations computed with QRTP with the matrix being partitioned into 8×8 blocks (which corresponds to executing the algorithm on 64 processors). The red dots give the ratio between the singular values of the QRTP approximation and the singular values of $\mathbf{A}$, $\sigma_i(\mathbf{A}_{QRTP})/\sigma_i(\mathbf{A})$. The approximations $\sigma_i(\mathbf{A}_{QRTP})$ are computed by applying the SVD to $\mathbf{A}_{QRTP}$. Figure 3.2 shows that the QRTP approximation gives a very accurate approximation of the 40 largest singular values of $\mathbf{A}$ with a ratio larger than 0.975. For the singular values 41 to 48, the ratio decreases to 0.9 and reaches 0.8 for singular values 49 and 50. It also shows that QRTP is close to the QRCP approximation, and for some singular values QRTP is more accurate, see for example the 48th singular values.

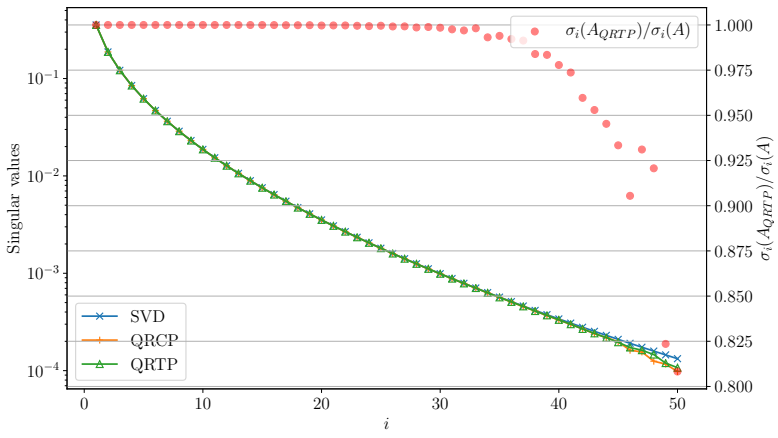


Figure 3.2: Singular values of the matrix `heat` and its approximation with QRCP and QRTP on 64 processors. The rank of the approximation is 50. The red dots give the ratio of the singular values of the approximation with QRTP to the original singular values.

Figure 3.3 presents the same analysis for the matrix `gravity`. The first 25 singular values of this matrix decay rapidly, while the following ones decay slowly. This figure shows that the QRTP approximation gives a very accurate approximation for the 22 largest singular values of the matrix `gravity`, with a ratio larger than 0.99. From the 23rd singular value the ratio decreases to reach 0.7 for the 26th singular value and is between 0.58 and 0.75 for the singular values 27 to 50. In addition, we see that QRCP and QRTP give an almost identical approximation. We also compute the relative error between QRCP and QRTP as $\frac{\|\mathbf{A}-\mathbf{A}_{QRTP}\|_F - \|\mathbf{A}-\mathbf{A}_{QRCP}\|_F}{\|\mathbf{A}-\mathbf{A}_{QRCP}\|_F} = -4.9 \times 10^{-5}$, showing that both algorithms have indeed very close results, with QRTP performing slightly better than QRCP in terms of Frobenius norm. Nevertheless, the two algorithms select different columns of $\mathbf{A}$, with only 5 columns selected by both QRCP and QRTP. For the matrix `heat` the relative error between QRCP and QRTP is -0.06 and there are 4 common columns selected by both algorithms, leading to the same conclusion for the matrix `heat`: both approximations have very close results even though most of the columns selected are different.

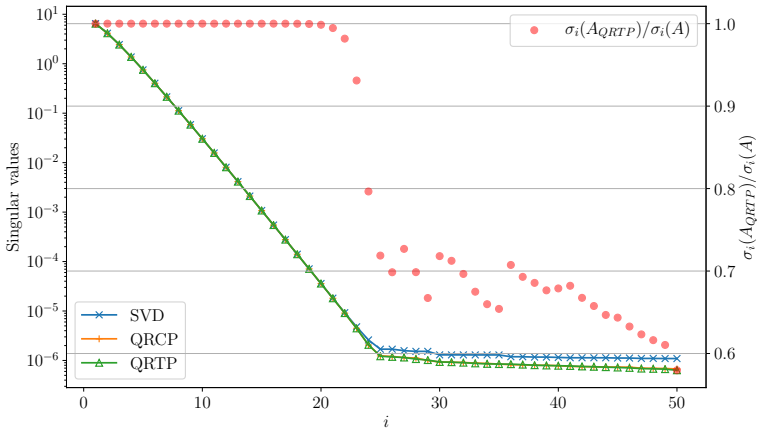


Figure 3.3: Singular values of the matrix `gravity` and its approximation with QRCP and QRTP on 64 processors. The rank of the approximation is 50. The red dots give the ratio of the singular values of the approximation with QRTP over the original singular values.

We now consider a set of 13 matrices arising from different application domains. They are presented in Table 3.1, where we also display the ratio $\frac{\sigma_1(\mathbf{A})}{\sigma_{k+1}\mathbf{A}}$ as an indication of the decay of their singular values, where $k = 50$. Figure 3.4 displays the minimum, maximum and average over $1 \leq i \leq 50$ of the ratio $\frac{\sigma_i(\mathbf{A}_{QRTP})}{\sigma_i(\mathbf{A})}$ where $\mathbf{A}$ corresponds to the input matrix, and $\mathbf{A}_{QRTP}$ to the compression of $\mathbf{A}$ distributed on an 8×8 processor grid using QRTP with a rank of 50. We can see that the worst ratio is 0.58 for the matrix `gravity`, and the smallest average is 0.72 for the matrix `ursell`. These values are far greater than the theoretical lower bound given in Eq. (3.12), which is $\frac{1}{\sqrt{1+50*\sqrt{64}*8*50^6(1000-50)}} = 5 \times 10^{-9}$. This lower bound is calculated using

#	Name	Size	$\frac{\sigma_1(\mathbf{A})}{\sigma_{k+1}(\mathbf{A})}$	Description
1	`baart`	1000×1000	2×10^7	Fredholm integral equation of the first kind
2	`blur`	900×900	1×10^0	Digital image deblurring
3	`deriv2`	1000×1000	3×10^3	Computation of the second derivative
4	`foxgood`	1000×1000	8×10^6	Severely ill-posed problem
5	`gravity`	1000×1000	6×10^6	1-D gravity surveying model problem
6	`heat`	1000×1000	3×10^3	Inverse heat equation
7	`parallax`	28×1000	N/A	Stellar parallax problem with 28 fixed, real observations
8	`phillips`	1000×1000	1×10^4	Phillips' "famous" problem
9	`shaw`	1000×1000	6×10^6	One-dimensional image restoration model
10	`spikes`	1000×1000	2×10^7	Test problem with a "spiky" solution
11	`tomo`	900×900	3×10^0	Create a 2D tomography test problem
12	`ursell`	1000×1000	3×10^7	Integral equation with no square integrable solution
13	`wings`	1000×1000	3×10^7	Test problem with a discontinuous solution

Table 3.1: Matrices corresponding to various modelisation problems.

$f = 1$ for all executions of strong RRQR.

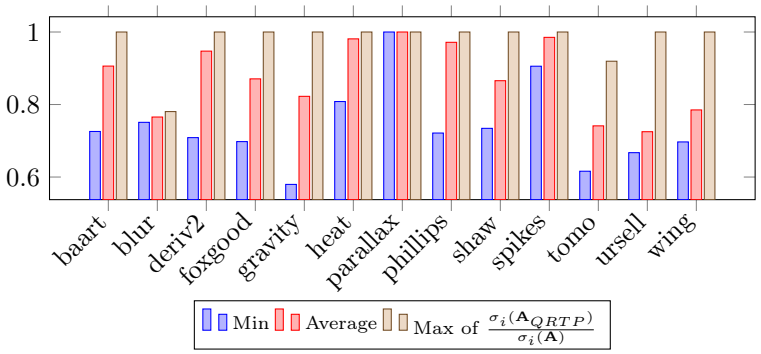


Figure 3.4: Singular values of the approximation of the 13 matrices.

3.4.1 Influence of the reduction tree used during tournament pivoting

We study now the influence of the structure of the reduction tree on the accuracy of QRTP. We define the degree of the tree as the number of children of each node. We assume the degree is constant for a given tree, and we study the impact of this parameter along with the type of reduction executed first, 1Dr-TP versus 1Dc-TP.

We compare the following strategies for QRTP. We set the number of processors to 64 such that the matrix is partitioned into 8×8 blocks and we compute approximations of rank $k = 50$.

- Row-first of degree 2: 1Dr-TP with reduction tree of degree 2, followed by 1Dc-TP with reduction tree of degree 2. Therefore, each 1D tournament executes three reductions.

- Row-first of degree 8: 1Dr-TP with a reduction tree of degree 8, followed by 1Dc-TP with a reduction tree of degree 8. Therefore, each 1D tournament executes one reduction.

- Column-first of degree 2: 1Dc-TP with reduction tree of degree 2, followed by 1Dr-TP with reduction tree of degree 2. Therefore, each 1D tournament executes three reductions.

- Column-first of degree 8: 1Dc-TP with reduction tree of degree 8, followed by 1Dr-TP with reduction tree of degree 8. Therefore, each 1D tournament executes one reduction.

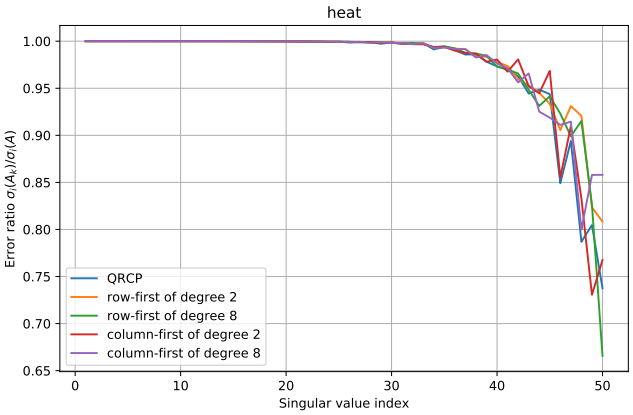


Figure 3.5: Error on the singular values $\sigma_i(\mathbf{A}_k)/\sigma_i(\mathbf{A})$ of the matrix `heat` distributed on an 8×8 processor grid using generalized QRTP with a rank 50 and different reduction trees. Degree 2 means that the 64 blocks are combined 2 by 2, degree 8 means that the 64 blocks are combined 8 by 8. Column-first and row-first refer to the reduction strategies defined in Section 3.3.2.

The results, displayed in Figure 3.5 for the matrix `heat` and Figure 3.6 for the matrix `gravity`, show the ratios of the approximated leading singular values with respect to the singular values as computed by SVD, i.e. $\frac{\sigma_i(\mathbf{A}_k)}{\sigma_i(\mathbf{A})}$, where $\sigma_i(\mathbf{A})$ is the ith singular value of the input matrix and $\sigma_i(\mathbf{A}_k)$ is the singular value of the approximation computed as explained previously. In this experiment the number of cores is always 64, what changes is the strategy to combine the selected columns. Corollary 1 suggests that the row-first strategy is more accurate than the column-first strategy, and the relations Eq. (3.13) and Eq. (3.14) lead to the conclusion that a lower degree strategy is less accurate (it has indeed more intermediate reductions). We use these two claims to state the following relation on the theoretical bounds of the approximations computed with QRTP using the differents strategies.

$$F_{\text{column\_first,degree 2}} \leq F_{\text{column\_first,degree 8}}, F_{\text{row\_first,degree 2}} \leq F_{\text{row\_first,degree 8}} \tag{3.17}$$

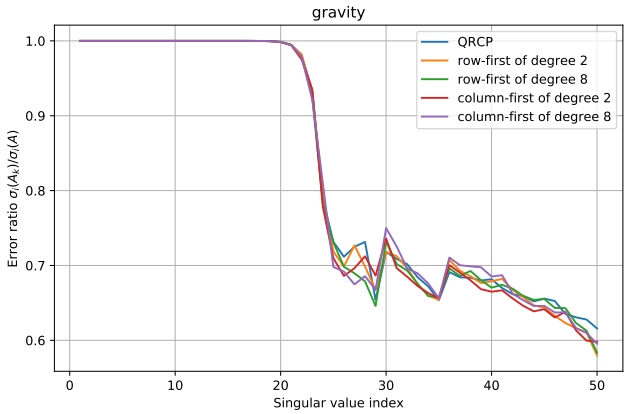


Figure 3.6: Error on the singular values $\sigma_i(\mathbf{A}_k)/\sigma_i(\mathbf{A})$ of the matrix `gravity` distributed on an 8×8 processor grid using generalized QRTP with a rank 50 and different reduction trees. Degree 2 means that the 64 blocks are combined 2 by 2, degree 8 means that the 64 blocks are combined 8 by 8. Column-first and row-first refer to the reduction strategies defined in Section 3.3.2.

Note that the bounds in Eq. (3.17) are only lower bounds for the ratios of the singular values and do not give the exact precision of the algorithm, because a relation between two lower bounds does not give any conclusion on the relation between the two actual values. We mention it here to merely link the theoretical and practical conclusions.

Going back to Figure 3.5 and Figure 3.6 we see that all strategies give very similar results. It invalidates in this case the expectations behind the relation Eq. (3.17), that there is an inequality relation between the accuracies of the different strategies. This leads to conclude that for these matrices the strategy does not impact the accuracy of QRTP, meaning that an approximation can be computed with the cheapest reduction tree without much loss of accuracy.

3.4.2 QRTP for image compression

We discuss here the results obtained by QRTP when used to compress images. While we do not claim that these methods should be used for image compression, we use them to visually display and interpret the selection of columns done by tournament pivoting on partitioned matrices. We use black and white images as matrices. After applying a low rank approximation, the factors are multiplied back such that the approximation matrix can be displayed as an image and can be compared with the original image.

Figure 3.7 presents an image and its compressions obtained using truncated SVD, QRCP, and QRTP with a truncation rank of 10. The last image shows the 10 columns selected by QRTP. Figure 3.8 gives the singular values of this image and their approximations obtained by the different algorithms. Performing a rank-10 approximation brings a lot of distortion to the original image. But only few differences can be seen between the images obtained by different approximation algorithms. QRCP and QRTP have less information for the left and

the right sides of the picture (note that the selected columns are mainly in the middle of the picture), leading to the presence of lines in this area. Moreover, QRCP and QRTP have very close results, showing that the subset selected by QRTP is close to the one selected by QRCP. The singular values represented in Figure 3.8 confirm this, with an error ratio for QRTP from 1 to 0.6. We see that the matrix corresponding to the image is close to low rank and the approximated singular values are very close to the original ones.

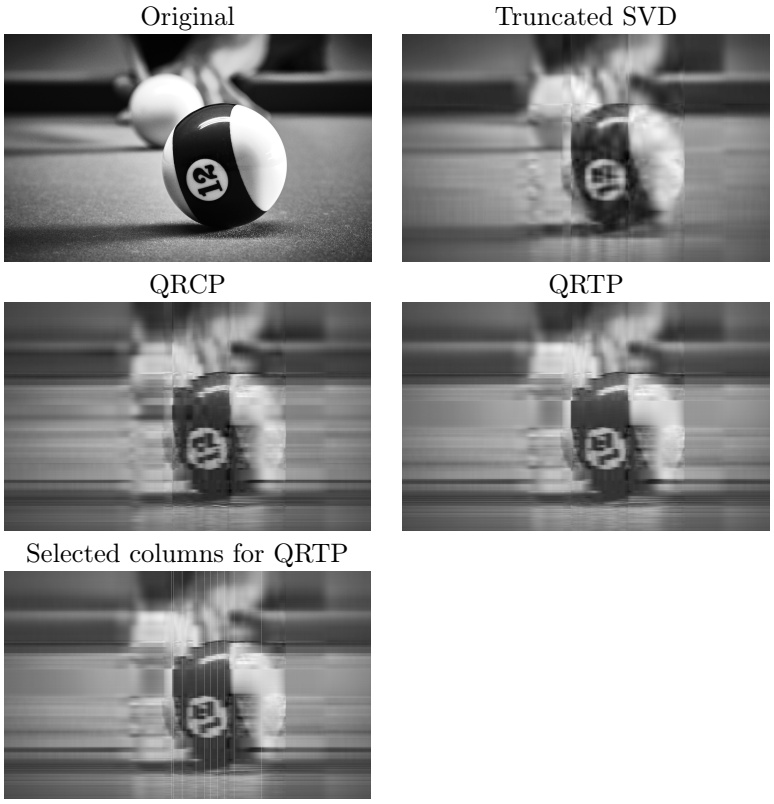


Figure 3.7: Compression of a 1190×1920 image for different algorithms with an approximation rank 10. QRTP uses a binary row-first reduction tree on an 8×8 processor grid.

Additional results obtained for four other images are displayed in Figure 3.9, where the results of QRCP and QRTP use truncation ranks 10 and 50. We see that these images have a more complex structure than the first one, which explains the need of a higher rank to capture more details in the compressed versions. We also see that the images compressed with QRTP and QRCP are very similar.

3.4.3 Accuracy comparison with RQRCP

The accuracy of QRTP is further illustrated in Figure 3.10. In this figure we compare the Frobenius norm error of the low rank approximation for three matrices and three algorithms. To the two matrices already presented in this

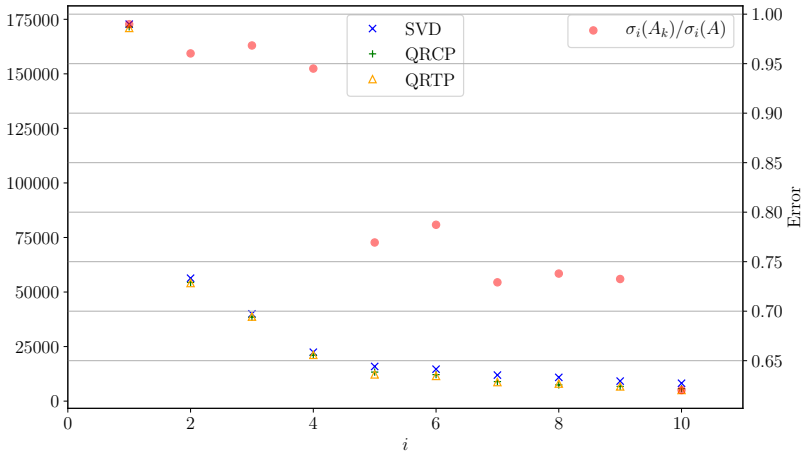


Figure 3.8: Singular values of the billiard ball image, original and compressed with QRCP and QRTP. The scale on the right gives the ratio $\sigma_i(\mathbf{A}_k)/\sigma_i(\mathbf{A})$ where $\mathbf{A}$ is the matrix corresponding to the picture and $\mathbf{A}_k$ is its approximation with QRTP.

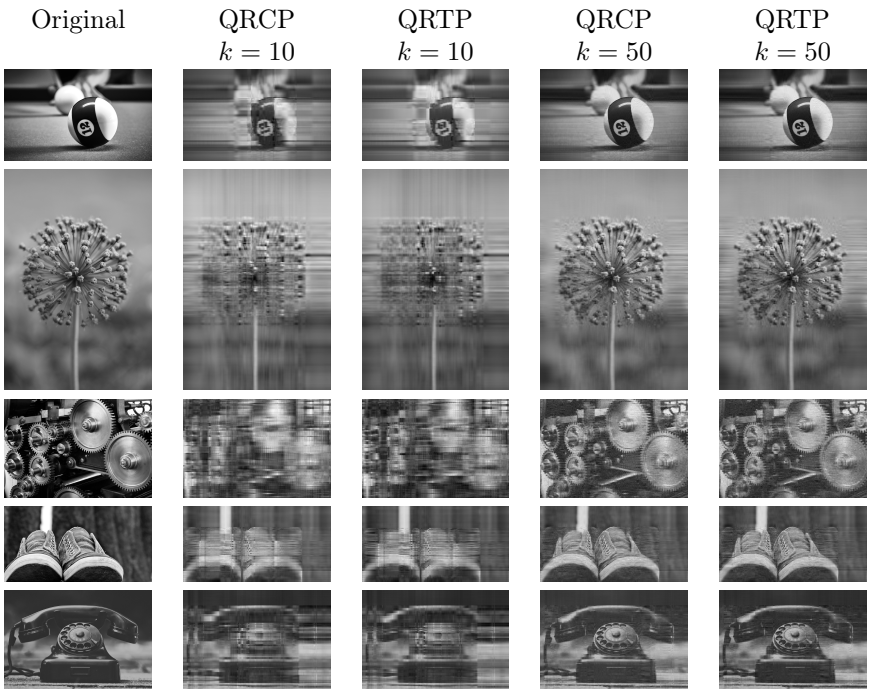
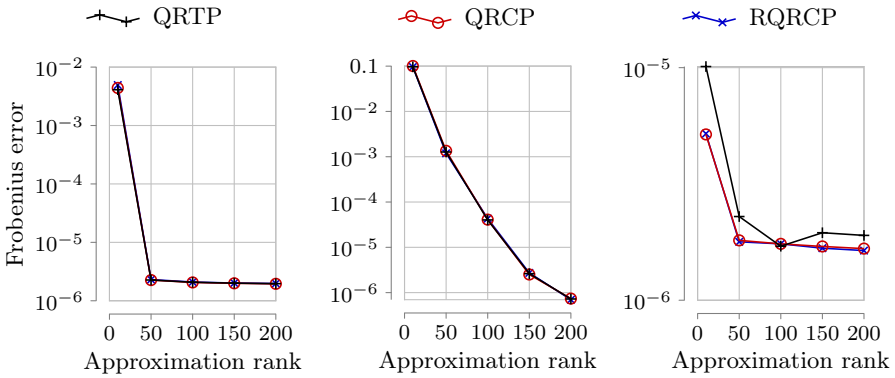


Figure 3.9: Compression of different images using QRCP and QRTP. QRTP uses a binary row-first division tree on an 8×8 processor grid.



(a) `gravity` (b) `heat` (c) `shaw`

Figure 3.10: Error when approximating with the truncated QRCP decomposition of three 10 000 × 10 000 matrices. Different distributed algorithms are compared for pivoting 50 columns of the QRCP decomposition.

section, we add the matrix `shaw` coming from a 1-D image restoration model, forming a test set of matrices of dimension 10000 × 10000 with rapidly decaying singular values, distributed on an 8×8 processor grid. Each of the three algorithms, QRTP, RQRCP and truncated QRCP, selects 10 to 200 columns from each matrix to produce the decomposition Eq. (1.1). RQRCP has a fixed oversampling of 10. To compute QRCP we used the routine `pdgeqpf` from ScaLAPACK, however the routine `pdgeqp3` and the truncated implementations (`partial_pdgeqpf` and `partial_pdgeqp3`) provided in Xiao et al. [118] give identical results. The Frobenius norm error is computed as $\frac{\|\mathbf{A}-\mathbf{A}_k\|_F}{\|\mathbf{A}\|_F}$ where $\mathbf{A}$ and $\mathbf{A}_k$ are respectively the input matrix and its approximation. This figure illustrates that both algorithms QRTP and RQRCP provide very close accuracy to QRCP, the reference sequential algorithm. RQRCP and QRTP are always almost equal to QRCP, except for matrix `shaw` for which QRTP provides slightly less accuracy.

3.5 Parallel design of QRTP

In this section we present the parallel design of Algorithm 11 and then we determine its cost in terms of computation, number of messages and volume of communication.

The parallel implementation of QRTP is described in Algorithm 12 with the following notation. For the coordinate (i, j) on the grid, $\mathbf{A}_{ij}$ is the submatrix of $\mathbf{A}$ on processor P_{ij}. Let H be the depth of the reduction tree and for $0 \leq h \leq H$, I_{ij}^h is the set of columns selected by the h^{th} reduction of $\mathbf{A}_{ij}$ (in particular I_{ij}^0 is the set of columns selected locally for $\mathbf{A}_{ij}$ and I_{ij}^H is the set of final columns, output of the algorithm). Let C_{ij}^h be the MPI communicator containing processors in the subtree having root I_{ij}^h. Considering that there is a processor in each communicator elected to own the selected columns at the end of the

reduction (for example MPI rank 0), let $\mathcal{C}_{ij}^h$ be the communicator containing processors $P_{i'j'}$ such that $P_{i'j'} \in C_{ij}^h$ and $P_{i'j'}$ is elected for the reduction $I_{i'j'}^{h-1}$. Figure 3.11 illustrates how the two kinds of communicators are composed. In Algorithm 12 the processors are not descheduled during binary 1Dr-TP, corresponding to the first iterations, but half of them are idle at each iteration of binary 1Dc-TP, corresponding to the last iterations of QRTP.

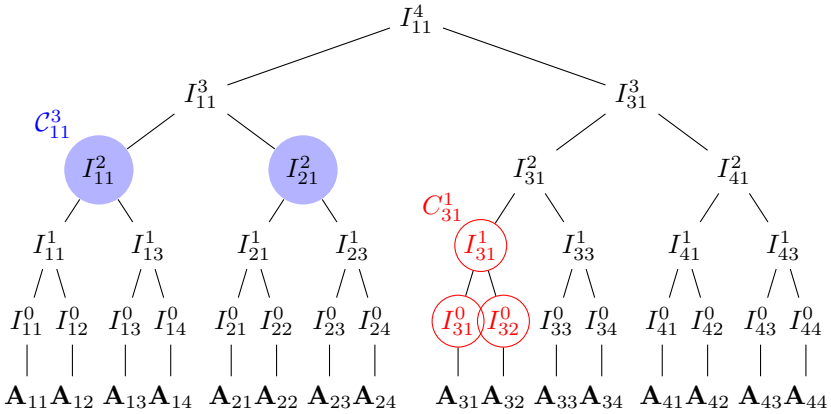


Figure 3.11: Reduction tree of QRTP on a 4×4 processor grid. Each node of the tree represents a reduction to select k columns from $2k$ columns concatenated from its children. Sets of processors are represented to illustrate the notation of MPI communicators. The communicator $\mathcal{C}_{11}^3$ regroups the elected processors on the children subtrees of the reduction I_{11}^3 *i.e.* processors P_{11} and P_{21}. The communicator C_{31}^1 regroups all processors in the subtree of reduction I_{31}^1 *i.e.* P_{31} and P_{32}.

3.5.1 Computational and communication cost

We now determine the computation and communication cost of QRTP as described in Algorithm 12 for a row-first strategy (binary 1Dr-TP followed by binary 1Dc-TP) and $\mathbf{A} \in \mathbb{R}^{mP_r \times nP_c}$. The maximum computation cost per processor is,

$$\begin{aligned} \#flops(QRTP) &= \texttt{QRCP}(m \times n, k) + \log_2(P_r)\texttt{QRCP}(2k \times 2k, k) \\ &+ \sum_{i=1}^{\log_2(P_r)} \texttt{TSQR}(m \times 2k, 2^i) + \log_2(P_c)\texttt{TSQR}(m \times 2k, P_r) \\ &+ \log_2(P_c)\texttt{QRCP}(2k \times 2k, k), \end{aligned}$$

where $\texttt{QRCP}(m \times n, k)$ represents the computational cost of the rank k QRCP algorithm on an $m \times n$ matrix, which we use in practice to substitute strong RRQR, and $\texttt{TSQR}(m \times n, P_r)$ represents the cost of TSQR [35] applied on P_r blocks of size $m\times n$. Considering that $\texttt{QRCP}(m\times n, k) = 4mnk$ [54] and $\texttt{TSQR}(m\times n, P_r) = 2n^2(m + (5\log_2 P_r - 1)\frac{n}{3})$, we obtain

$$\#flops(QRTP) = 4mnk + K_1k^3 + K_2mk^2,$$

Algorithm 12 Parallel QRTP

Require: input matrix $\mathbf{A}$, reduction tree S, rank of the approximation k, current processor P_{ij}

Compute strong RRQR of $\mathbf{A}_{ij}$ to select k columns $\rightarrow I_{ij}^0$
for h from 1 to H **do**
  if $\{\mathbf{A}_{ij}, P_{ij} \in \mathcal{C}_{ij}^h\}$ is a row partition **then**
    if $P_{ij} \in \mathcal{C}_{ij}^h$ **then**
      Merge I_{ij}^{h-1} on communicator $\mathcal{C}_{ij}^h \rightarrow \bar{I}_{ij}^h$
      Broadcasts $\bar{I}_{ij}^h$ on communicator C_{ij}^{h-1}
    else
      Receive broadcast from communicator C_{ij}^{h-1}
    end if
    `TSQR` of columns $\bar{I}_{ij}^h$ of $\{\mathbf{A}_{ij}, P_{ij} \in C_{ij}^h\} \rightarrow \mathbf{R}$ factor on elected processor of C_{ij}^h
    Compute strong RRQR of $\mathbf{R} \rightarrow I_{ij}^h$
  else
    if $\exists i', P_{i'j} \in \mathcal{C}_{ij}^h$ **then**
      Elected processor of $\mathcal{C}_{ij}^h \rightarrow P_{i'y}$
      Gather subcolumns I_{ij}^{h-1} of $\mathbf{A}_{ij}$ on $P_{iy}^h \rightarrow \bar{I}_{ij}^h$
      if $j = y$ ($P_{i'j}$ is elected on $\mathcal{C}_{ij}^h$) **then**
        `TSQR` of columns $\bar{I}_{ij}^h \rightarrow \mathbf{R}$ factor on elected processor of C_{ij}^h
        Compute strong RRQR of $\mathbf{R} \rightarrow I_{ij}^h$
      end if
    end if
  end if
end for

Ensure: I_{ij}^H indices of k columns of $\mathbf{A}$

where

$$\begin{aligned} K_1 &= 16\log_2(P_rP_c) + 8\log_2(P_r)(1 + 5\log_2(P_r)/3) + \frac{16}{3}(5\log_2(P_r) - 1), \\ K_2 &= 8(\log_2(P_r) + 1). \end{aligned}$$

The number of messages is,

$$\#messages = (\log_2 P_c + \log_2 P_r)(1 + \log_2 P_r),$$

and the volume of communication is

$$\#words = (2k^2 + \frac{k}{2})(\log_2 P_r)(1 + \log_2 P_r) + 2k(\log_2 P_c)(m + 2k\log_2 P_r).$$

3.6 Parallel performance of QRTP

In this section we study the parallel performance of Algorithm 11 and also compare it with that of RQRCP. We use a parallel machine formed by 342 nodes, each composed of two Intel Skylake 2.7GHz (AVX512) with 24 physical cores each. Each node has 180 GB of shared memory. This machine thus has a total of 16384 cores with 60.117 TB of distributed memory. The algorithm is implemented in C++ and built with Intel C Compiler 20.0.0, MKL 20.0.0 and IntelMPI 2018.0.3.222. For these parallel performance results, to be able to use large dense matrices, we generate for each run a double precision matrix using C++'s pseudo-random double precision number generator[1]. For P a given number of processors, the matrix is distributed over a $\sqrt{P} \times \sqrt{P}$ grid of processors and QRTP is applied. Experiments compare only runtimes to obtain a set of column indices. The subsequent steps to obtain a low rank approximation, i.e. computing $\mathbf{Q}_1$ and $\begin{bmatrix}\mathbf{R}_{11} & \mathbf{R}_{12}\end{bmatrix}$ from Eq. (1.1) are identical in both algorithms.

Figure 3.12 displays the weak scaling of QRTP and RQRCP with the submatrices owned by each processor being of constant dimensions of 1024×1024 while the number of processors increases. Starting with a 2×2 processor grid, each dimension of the global matrix is doubled until the processor grid reaches a size of 128×128 and the global matrix has dimensions 131072×131072. In weak scaling experiments scalable algorithms runtimes are expected to follow a straight line. In our case QRTP is more scalable than RQRCP and especially for a large number of processors the runtime of RQRCP is at least 6 times larger than the runtime of QRTP.

The computational cost of QRTP essentially comes from sequential truncated QRCP factorizations. The first QRCP is performed locally on each initial submatrix, and subsequent QRCPs are performed on $k \times k$ matrices along the reduction tree (step 11 of Algorithm 12), where k is the approximation rank. Therefore, the size of the input submatrices and the approximation rank have a significant impact on the runtime. To illustrate this Figure 3.13 and Figure 3.14 display runtimes when increasing the approximation rank, for different matrix sizes, and a 64×64 processor grid. In the first figure the input matrix has dimensions 32768×32768, hence the local submatrices have dimensions 512×512. In the second figure the input matrix has dimensions 65536×65536, hence the

[1] initialized with `std::uniform_real_distribution<double> dist(-32.768, 32.768)`, c.f. `https://en.cppreference.com/w/cpp/numeric/random/uniform_real_distribution`

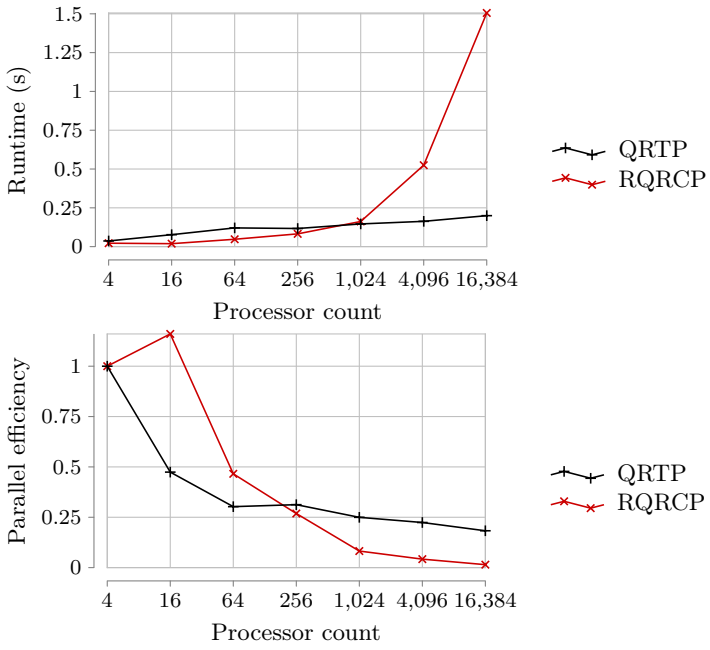


Figure 3.12: Weak scaling for a random matrix with a fixed processor submatrix size of 1024×1024 and an approximation rank of 50. We compare the column selection steps of QRTP and RQRCP when increasing together the number of processors and the global matrix size. The processor grid is $\sqrt{P} \times \sqrt{P}$ where P is the processor count. The cyclic block size of RQRCP is 1024 and the oversampling is 10.

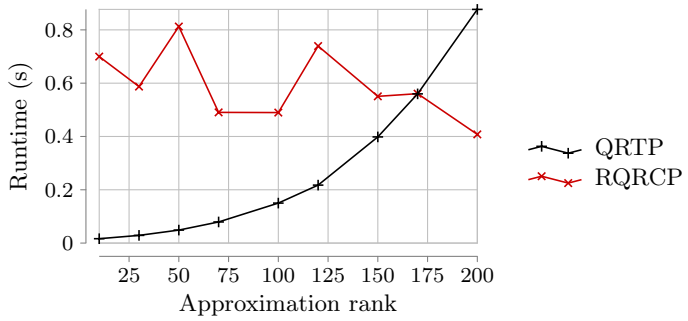


Figure 3.13: Random matrix of size $32\,768 \times 32\,768$ distributed on a 64×64 grid of processors. The cyclic block size of RQRCP is 512 and the oversampling size is 10.

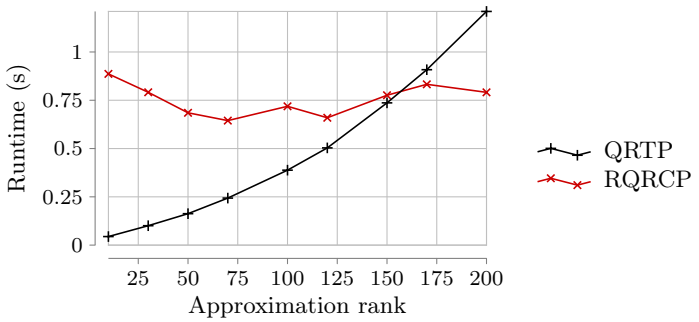


Figure 3.14: Random matrix of size 65 536 × 65 536 distributed on a 64 × 64 grid of processors. The cyclic block size of RQRCP is 1024 and the oversampling size is 10.

local submatrices have dimensions 1024 × 1024. With regard to the impact of the approximation rank, QRTP is faster than RQRCP for a lower approximation rank. Implementing the block cyclic distribution in QRTP may allow a similar performance for larger values. In terms of submatrix size, 1024 and 512 submatrix sizes give reasonable results. However, more extensive tests with larger block size, not produced here, showed that if the submatrix size increases beyond 2048 × 2048, the first QRCP tends to become a bottleneck in QRTP.

3.7 Conclusion

In this chapter we introduced QRTP, a scalable communication avoiding algorithm using QR with tournament pivoting to compute a low rank approximation. QRTP guarantees bounds on the singular values of the resulting low rank approximation and provides accurate results in practice. QRTP for image compression shows very close results to the sequential QRCP algorithm. We also presented an MPI implementation of QRTP that is shown to scale well on up to several thousand cores. Future work may address the possibility of using non-binary reduction trees and the associated trade-off between performance and accuracy, as well as the possibility of combining the selection of subsets of columns and of rows during the same tournament to obtain a CUR approximation.

Chapter 4

Distributed Tucker decomposition using QR with tournament pivoting

Many scientific problems involve multilinear operators. As an example, solving the electronic Schrödinger equation of an N-electrons molecule can be rewritten as finding the smallest eigenvalue of an operator belonging to $\mathbb{R}^{4^N}$. To this end matrix approximations presented in previous chapters are not sufficient, and higher order approximations are to be considered. The Tucker decomposition presented in Section 4.1.1 is a widely recognized generalization of the truncated SVD. For this reason it is a good candidate to fulfill this goal. In this section we consider a very large tensor, typically larger than the memory of a single computer, and present how to compute a Tucker decomposition using a distributed architecture. First, several existing solutions are recollected, then a new method taking advantage of the QRTP algorithm presented in Chapter 3 is introduced. After producing error bounds for this method, the practical runtime and accuracy is evaluated and compared with the state-of-the-art.

4.1 Context

This section introduces the tensor decomposition at stakes in this chapter, the Tucker decomposition, as well as several state-of-the-art implementations.

4.1.1 Tucker decomposition

Considering a d-order tensor, each mode (dimension) of the tensor can be reduced by means of projection matrices, called *factor matrices*, converting each initial *observational* modes into new, preferably smaller, *derivational* modes. The tensor is then represented as the contracted product of this core tensor with the factor matrices.

Formally, the *Tucker decomposition* [111] of a d-order tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \dots n_d}$ and ranks $(r_1 \dots r_d)$ is composed of a *core tensor* $\mathcal{C} \in \mathbb{R}^{r_1 \times \dots \times r_d}$ and factor matrices $(\mathbf{U}_1, \dots, \mathbf{U}_d)$ such that

$$\tilde{\mathcal{A}} = \mathcal{C} \times_1 \mathbf{U}_1^\intercal \ldots \times_n \mathbf{U}_n^\intercal. \tag{4.1}$$

The factor matrices have orthonormal columns and for a given i, $\mathbf{U}_i \in \mathbb{R}^{n_i \times r_i}$. The core tensor is then the projected representation of $\mathcal{A}$ in $\mathbb{R}^{r_1 \times \ldots \times r_d}$, obtained from $\mathbb{R}^{n_1 \times \ldots \times n_d}$ by applying orthogonal transformations in every dimension. The factor matrices are projection matrices on this new vector space, applying also the change of basis.

This decomposition was first introduced by Tucker [111] along with several methods to compute it: method I addresses balanced modes, methods II and III address the particular cases of one mode being much larger. The first method recollects as the following for each mode i:

1. Compute the Gram matrix of the unfolding $\mathbf{P} = \mathcal{A}_i \mathcal{A}_i^\intercal$.

2. Select the eigenvectors corresponding to the r_i largest eigenvalues of $\mathbf{P}$, forming the matrix $\mathbf{U}_i$.

Because the i-mode unfolding is usually short and wide, it is adequate to rather compute the truncated SVD on the Gram matrix, i.e. $\mathcal{A}_i \mathcal{A}_i^\intercal$, as the left singular vectors of $\mathcal{A}_i$ are a set of orthonormal eigenvectors of $\mathcal{A}_i \mathcal{A}_i^\intercal$.

To complete the picture, it is important to mention a similar method. Carroll and Chang [23] introduced the alternating least squares (ALS) algorithm, called CANDECOMP and at the same time Harshman [61] introduced the equivalent PARAFAC algorithm. ALS, further studied in Kroonenberg and de Leeuw [81], Sands and Young [100], is another method to find factor matrices in the Tucker decomposition. Because resolving all factor matrices at once is not computationally feasible, this method suggests initializing all factor matrices but one and performing a least squares resolution limited to this factor matrix. All factor matrices are updated subsequently in this manner. These steps are repeated until the error requirement is met. More recently, Kaya and Uçar [75] proposed a faster distributed version of the CANDECOMP algorithm with sparse matrices.

Kroonenberg and de Leeuw [81] show that the full Tucker decomposition, i.e. with $(n_1 \ldots n_d) = (r_1 \ldots r_d)$, is unique. Furthermore they prove that the minimization problem for a given $(r_1 \ldots r_d)$,

$$\min \|\mathcal{A} - \tilde{\mathcal{A}}\|_F$$

where the factor matrices have orthonormal columns, always has a solution, which is the factor matrix being the left singular vectors corresponding to respective largest singular values (the largest components) of each unfolding.

Later, De Lathauwer et al. [34] further formalized the Tucker decomposition with contemporary terminology around the singular value decomposition, renaming it high order SVD (HOSVD). This algorithm is recollected in Algorithm 13. It is directly linked to the steps given by Tucker, and provides many additional properties derived from the SVD.

Algorithm 13 HOSVD - High Order Singular Value Decomposition.

Require: A d-order tensor $\mathcal{A}$, a set of ranks $(k_1, \ldots, k_d)$

1: **for** i from 1 to d **do**
2: Compute the Gram matrix $\mathcal{A}_i\mathcal{A}_i^\intercal$ of the i-mode unfolding of $\mathcal{A}$
3: Compute the k_i truncated SVD $\mathcal{A}_i = \mathbf{U}_i\mathbf{\Sigma}\mathbf{V}$
4: **end for**
5: $\mathcal{C} = \mathcal{A}$
6: **for** i from 1 to d **do**
7: $\mathcal{C} = \mathcal{C} \times_i \mathbf{U}_i$
8: **end for**

Ensure: A core tensor $\mathcal{C}$ and d factor matrices $(\mathbf{U}_1, \ldots, \mathbf{U}_d)$ with orthonormal columns.

Kolda and Bader [78] observe that "the truncated HOSVD is not optimal in terms of giving the best fit as measured by the norm of the difference, but it is a good starting point for an iterative ALS algorithm". Indeed De Lathauwer et al. [33] provides a method to initialize the ALS with an HOSVD. This method is called High Order Othogonal Iteration (HOOI).

Various notable contributions make use of randomization to compute the Tucker decomposition [47, 97, 91, 90, 28], while others use the cross approximation [102, 101].

Vannieuwenhoven et al. [114] more recently produced a faster version of the HOSVD, called sequentially truncated high order SVD (ST-HOSVD), given in Algorithm 14.

Algorithm 14 ST-HOSVD - Sequentially Truncated High Order Singular Value Decomposition.

Require: A d-order tensor $\mathcal{A}$, a set of ranks $(k_1, \ldots, k_d)$

1: $\mathcal{C} = \mathcal{A}$
2: **for** i from 1 to d **do**
3: Compute the Gram matrix $\mathcal{C}_i\mathcal{C}_i^\intercal$ of the i-mode unfolding of $\mathcal{C}$
4: Compute the k_i truncated SVD $\mathcal{C}_i = \mathbf{U}_i\mathbf{\Sigma}\mathbf{V}$
5: $\mathcal{C} = \mathcal{C} \times_i \mathbf{U}_i$
6: **end for**

Ensure: A core tensor $\mathcal{C}$ and d factor matrices $(\mathbf{U}_1, \ldots, \mathbf{U}_d)$ with orthonormal columns.

The following theorem is provided in Vannieuwenhoven et al. [114], and proves correct for both HOSVD and ST-HOSVD. The error bound sums the error lost when truncating each unfolding. Even though in the case of ST-HOSVD, the truncation on the i-mode unfolding is done on the projection of the unfolding, the error bound relates to the original tensor unfolding truncation error. As said in the paper, "when truncating the T-HOSVD and ST-HOSVD to a given multilinear rank, both approximation errors are bounded by the same quantity".

Theorem 10 (HOSVD and ST-HOSVD error bound)**.** *Let $\mathcal{A} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ be a d-order tensor, and $\tilde{\mathcal{A}}$ its $(k_1, \ldots, k_d)$ HOSVD or ST-HOSVD approximation.*

Then the following holds,

$$\|\mathcal{A} - \tilde{\mathcal{A}}\|_F^2 \leq \sum_{i=1}^{d} \sum_{j=k_i+1}^{n_i} \sigma_j^2(\mathcal{A}_i).$$

This bound does not depend on the order of processing the modes of $\mathcal{A}$.

The question of finding an approximate column space for matrices, discussed in previous sections, strongly echoes in this context. Consider the matrix $\mathbf{A}$, a 2-order tensor, to compute $\mathbf{U}_1$ and $\mathbf{U}_2$ in Eq. (4.1) is to compute two partially orthogonal matrices mapping respectively the row and column indices (modes) to smaller indices. This is the exact purpose of the truncated SVD, the truncated QR decomposition and the cross approximation to name a few. For this reason the Tucker decomposition can be considered as a high order low rank approximation.

4.1.2 Existing solutions to compute the Tucker decomposition in parallel

The HOSVD algorithm, extensively introduced in Section 4.1.1, aims at producing a Tucker decomposition of a tensor. It consists of computing the factor matrix for each dimension i from the truncated SVD of the i-mode unfolding. HOSVD can be parallelized in two ways. First, each factor matrix can be computed independently. Indeed, the input data, the initial tensor, is available from the beginning. Therefore, each factor matrix can be computed on a different set of nodes. Second, the operations for computing a factor matrix can be performed using parallel implementations, and especially the matrix product to compute the Gram matrix is very well parallelizable.

The alternative algorithm ST-HOSVD requires to compute the factor matrices sequentially. For this, only the second type of parallelisation (computing a factor matrix) can be performed.

Parallel implementations of ST-HOSVD include TuckerMPI [13, 8], Kaya and Uçar [74] specific for sparse tensors, and more recent ones such as Choi et al. [30] using GPUs and Cobb et al. [31] decreasing memory usage.

TuckerMPI stands out as the reference modern implementation of ST-HOSVD. The code is openly available, thus it is our choice in the following for comparison.

4.2 Partitioned unfolding and applying QRTP

In the cited literature, computing the factor matrices is done with the truncated SVD. In the following QRTP (see Chapter 3) is used instead to compute the factor matrices of the Tucker decomposition.

Consider a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ distributed on a set of processors. Each mode is partitioned, hence the processor grid becomes a processor tensor. An example is given in Figure 4.1, where a $4 \times 4 \times 4$ tensor is distributed on a $2 \times 2 \times 2$ processor tensor. Each processor owns a $2 \times 2 \times 2$ subtensor, identified with a particular color.

When unfolding this tensor, the resulting matrix has a spread block distribution, as displayed in Figure 4.2, such that it is not distributed on a typical

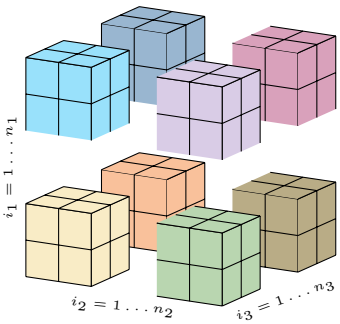


Figure 4.1: Distribution of a 3-order tensor on a $2 \times 2 \times 2$ processor tensor.

processor grid. We call this distribution a *partitioned unfolding*. Yet, a regular distribution is necessary if this matrix is to be approximated with QRTP.

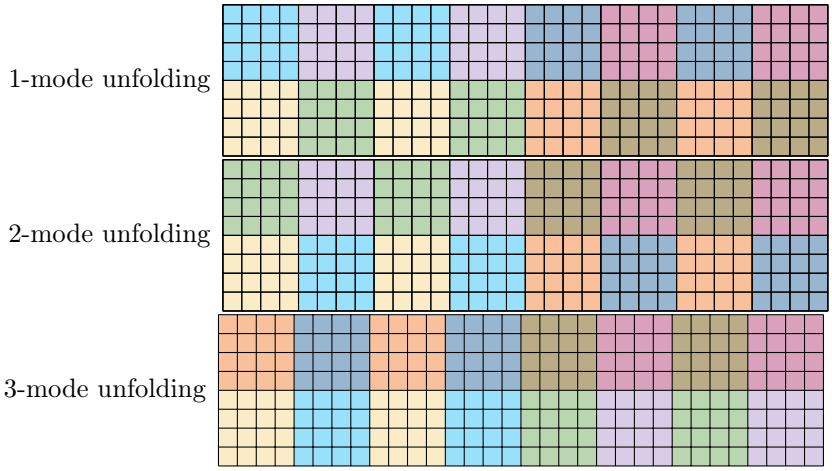


Figure 4.2: Processor repartition of the different unfolding of the tensor from Figure 4.1.

Austin et al. [8, Sec. IV.C] exposes an alternative via duplicating the subtensor along the unfolding direction and considering the result in a different data layout. Performing communications to gather continuous blocks on the same processor, as mentioned in Zhou et al. [123, Sec. IV.B], is another method to unfold a distributed tensor. Nevertheless, in order to avoid communication as well as storage and alternative data layouts, we propose a different approach. First we show that unfolding local subtensors on each processor, and concatenating the resulting matrices, leads to a column permutation from the actual unfolding. In addition, we show that QRTP gives the same output when applied to a column-permuted matrix and the original one. We detail this method in the following.

Theorem 11 (Partitioned unfolding [16])**.** *For each $i \in (1, \ldots, d)$, the $i^{\backslash\backslash}$-mode partitioned unfolding $\mathcal{A}_{i^{\backslash\backslash}}$ of the tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is equal to a column*

permutation of the corresponding matrix of the direct i-mode unfolding of $\mathcal{A}$. In other words, $\mathcal{A}_{i\backslash\backslash} = \mathcal{A}_i\mathbf{\Pi}_i \quad \forall i \in (1,\ldots,d)$, where each $\mathbf{\Pi}_i$ is a permutation matrix, $\mathcal{A}_{i\backslash\backslash}$ is the $i^{\backslash\backslash}$-mode partitioned unfolding, and $\mathcal{A}_i$ is the ordinary i-mode unfolding of $\mathcal{A}$.

Theorem 11 shows that a partitioned unfolding of a tensor is only a column-wise permutation of the actual tensor unfolding. See Beaupère et al. [16] for a formal proof of this theorem, mostly written by co-authors. For a deeper explanation, consider on the one hand that each processor unfolds its subtensor locally, and that processors are organized in a grid computed by unfolding the processor tensor. The resulting distributed matrix is formed from each of these local unfoldings. On the other hand imagine the complete tensor located on a single processor, unfolded on this processor, and then distributed on the same processor grid. Theorem 11 claims that these two distributed matrices are only different by a permutation of columns. Nevertheless, the first method is very advantageous as it generates unfoldings for all modes without communication or extra storage.

To use QRTP on partitioned unfoldings, it remains to prove that it is compatible with the permutation of columns. We now compare the executions of QRTP on a matrix and its column-wise permutation. The following shows that both executions comply to the same error bound.

Let $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{\Pi}$ a permutation matrix, $1 \leq k \leq \min(m,n)$. We note the RRQR decomposition of $\mathbf{A\Pi}$ computed with QRTP as

$$(\mathbf{A\Pi})\bar{\mathbf{\Pi}} = \mathbf{Q}\begin{bmatrix}\mathbf{R}_{11} & \mathbf{R}_{12}\\ & \mathbf{R}_{22}\end{bmatrix}, \tag{4.2}$$

where $\mathbf{R}_{11} \in \mathbb{R}^{k\times k}$ and where $\bar{\mathbf{\Pi}}$ is such that for any $1 \leq j \leq n-k$

$$\gamma_j^2(\mathbf{R}_{11}^{-1}\mathbf{R}_{12}) + \gamma_j^2(\mathbf{R}_{22})/\sigma_{\min}^2(\mathbf{R}_{11}) \leq k f_{TP}^2 \tag{4.3}$$

and $f_{TP} \geq 1$. We suppose that the QRTP process does not encounter identical column norms, hence no arbitrary choice of pivot is made. Then

1. $\mathbf{\Pi}\bar{\mathbf{\Pi}}$ leads to the same RRQR decomposition for the matrix $\mathbf{A}$, with the same bound f_{TP},

2. $\mathbf{A}$ computed with QRTP also observes the same bound f_{TP}.

The first point is obvious by changing the parenthesis $(\mathbf{A\Pi})\bar{\mathbf{\Pi}}$ with $\mathbf{A}(\mathbf{\Pi}\bar{\mathbf{\Pi}})$ in Eq. (4.2). It shows that the columns selected with QRTP on the permuted matrix are valid pivots for the original matrix, producing the same decomposition. From Eq. (3.11) it is clear that the upper bound f_{TP} does not depend on the order of the columns of the input matrix, hence the second point. It suggests that truncating the permuted decomposition will produce a similar error as executing QRTP on the original matrix.

By considering the theorem and subsequent affirmations of this section, it is now explicit that QRTP can be applied directly on the partitioned unfoldings of the input tensor.

4.3 High-Order QR with tournament pivoting

The previous section revealed the feasibility to apply QRTP on the partitioned unfolding of a tensor. For each unfolding a set of pivot columns is computed, and subsequently, with additional parallel operations, the factor matrices of the Tucker decomposition. The complete algorithm, the High Order QRTP decomposition (HOQRTP), appears in Algorithm 15. Section 4.3 was made in collaboration with David Frenkiel (in addition to Laura Grigori), a former Phd student at Inria, and the figures of Section 4.3 are due to him. Section 4.3 summarizes the results presented in Beaupère et al. [16, Sec. 3,5.1-5.3], initially written by D. Frenkiel.

Algorithm 15 HOQRTP: Low-rank approximation of a tensor

Require: A tensor $\mathcal{A}$ and a set of ranks $\{k_i\}_{1\leq i\leq d}$
1: Distribute the sub-tensors of $\mathcal{A}$ amongst the processors
2: **for** i in $1:d$ **do**
3: Extract i-mode unfolding of sub-tensor on each processor
4: **if** unfolding matrix $\mathcal{A}_i$ is tall **then**
5: Unfold processors according to the i-mode unfolding so that $\mathcal{A}_{i\backslash\backslash}$ is logically distributed on the processor grid
6: Run QRTP (for rank k_i) on $\mathcal{A}_{i\backslash\backslash}$ to get $\mathcal{A}_{i\backslash\backslash}\boldsymbol{\Pi} = \mathbf{QR}$
7: Truncate $\mathbf{Q}$ to get the factor matrix $\mathbf{U}_i = \mathbf{Q}[:, 1:k_i]$
8: **else if** /* $\mathcal{A}_i$ is wide */ **then**
9: Unfold processors according to the i-mode unfolding so that $\mathcal{A}^\intercal_{i\backslash\backslash}$ is logically distributed on the processor grid
10: Run QRTP (for rank k_i) on $\mathcal{A}^\intercal_{i\backslash\backslash}$ to get $\mathcal{A}^\intercal_{i\backslash\backslash}\boldsymbol{\Pi} = \mathbf{QR}$
11: Run SVD on $\boldsymbol{\Pi}_1\mathbf{R}_{11}^\intercal + \boldsymbol{\Pi}_2\mathbf{R}_{12}^\intercal$ to get $\boldsymbol{\Pi}_1\mathbf{R}_{11}^\intercal + \boldsymbol{\Pi}_2\mathbf{R}_{12}^\intercal = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\intercal$
12: Let $\mathbf{U}_i = \mathbf{U}[:, 1:k_i]$ be the factor matrix associated with the i-mode unfolding
13: **end if**
14: **end for**
15: Compute the core tensor: $\mathcal{C} = \mathcal{A} \times_1 \mathbf{U}_1^\intercal \times_2 \mathbf{U}_2^\intercal \cdots \times_d \mathbf{U}_d^\intercal$
Ensure: A Tucker decomposition (core tensor $\mathcal{C}$ and factor matrices $\{\mathbf{U}_i\}_{1\leq i\leq d}$ such that $\widetilde{\mathcal{A}} = \mathcal{C} \times_1 \mathbf{U}_1 \times_2 \cdots \times_d \mathbf{U}_d$)

For each unfolding, Algorithm 15 lays out two different cases. If the unfolding is tall, QRTP is applied directly. If the unfolding is short and wide, QRTP is applied on its transpose. In this way QRTP is always applied to a tall and skinny matrix. This is particularly interesting because the error of QRTP grows with the width of the input matrix (see Eq. (3.12)).

In the second case, QRTP approximates the row space, instead of the column space, of the input matrix such that

$$\mathbf{A}^\intercal\boldsymbol{\Pi} = \mathbf{Q}\begin{pmatrix}\mathbf{R}_{11} & \mathbf{R}_{12}\\ & \mathbf{R}_{22}\end{pmatrix}.$$

To find an approximate column space, we propose the following. First transpose both sides such that

$$\mathbf{A} = \boldsymbol{\Pi}\begin{pmatrix}\mathbf{R}_{11}^\intercal & \\ \mathbf{R}_{12}^\intercal & \mathbf{R}_{22}^\intercal\end{pmatrix}\mathbf{Q}^\intercal$$

and furthermore

$$\mathbf{A} = \left[\mathbf{\Pi} \begin{pmatrix} \mathbf{R}_{11}^\intercal \\ \mathbf{R}_{12}^\intercal \end{pmatrix} \quad \mathbf{\Pi} \begin{pmatrix} \\ \mathbf{R}_{22} \end{pmatrix} \right] \mathbf{Q}^\intercal.$$

By computing the economic SVD $\mathbf{\Pi} \begin{pmatrix} \mathbf{R}_{11}^\intercal \\ \mathbf{R}_{12}^\intercal \end{pmatrix} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\intercal$, we can use $\mathbf{U}$, a partial orthogonal matrix, as the factor matrix, such that the residual of the approximation is

$$\mathbf{A} - \mathbf{U}\mathbf{U}^\intercal\mathbf{A} = (\mathbf{I} - \mathbf{U}\mathbf{U}^\intercal)\mathbf{\Pi} \begin{pmatrix} \\ \mathbf{R}_{22}^\intercal \end{pmatrix} \mathbf{Q}^\intercal, \tag{4.4}$$

corresponding to $\mathbf{R}_{22}$ arranged in different, reduced, row and column basis sets.

4.3.1 Error bound

The HOQRTP algorithm uses several QRTP approximations. An error bound was given above for QRTP, in Eq. (3.11), in the form of RRQR. In addition bounds on the singular values were given in Eq. (3.12). The following theorem uses these inequalities to produce a Frobenius norm of the residual, when applying HOQRTP to a tensor.

Theorem 12. *Given a set of ranks $\{k_i\}_{1 \leq i \leq d}$ and any $f > 1$, the QRTP factorizations of the unfolding matrices of the tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ are such that the low-multilinear rank tensor approximation $\widetilde{\mathcal{A}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ generated by Algorithm 15 satisfies the error bound,*

$$\|\mathcal{A} - \widetilde{\mathcal{A}}\|_F^2 \leq \sum_{i=1}^{d} \sum_{j=k_i+1}^{r_i} \sigma_j^2(\mathcal{A}_i)[1 + \tilde{f}_i^2(n_i - k_i)],$$

where

$$\tilde{f}_i = \sqrt{P} P_r^{(i)} k_i^{\log_2(P)} f^{\log_2(P)+1},$$

and r_i and $\sigma_j(\mathcal{A}_i)$ are respectively the rank and singular values of the i-mode unfolding $\mathcal{A}_i$, and where P is the total number of processors and $P_r^{(i)}$ is the height of the processor grid related to the i^{th} unfolding.

In the following Corollary this residual is compared to the residual of $\mathcal{A}_{\text{best}}$, the best rank-$(k_1, \ldots, k_d)$ approximation. The existence of $\mathcal{A}_{\text{best}}$ is proved in Kroonenberg and de Leeuw [81], as mentioned in Section 4.1.1.

Corollary 2. *Given a set of ranks $k_1, \ldots, k_d$ and any $f > 1$, the QRTP factorizations of the unfoldings matrices of the tensor $\mathcal{A}$ are such that the low-multilinear rank tensor approximation $\widetilde{\mathcal{A}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ generated by Algorithm 15 satisfies the error bound,*

$$\|\mathcal{A} - \tilde{\mathcal{A}}\|_F^2 \leq d(1 + \max_i[\tilde{f}_i^2(n_i - k_i)])\|\mathcal{A} - \mathcal{A}_{best}\|_F^2, \tag{4.5}$$

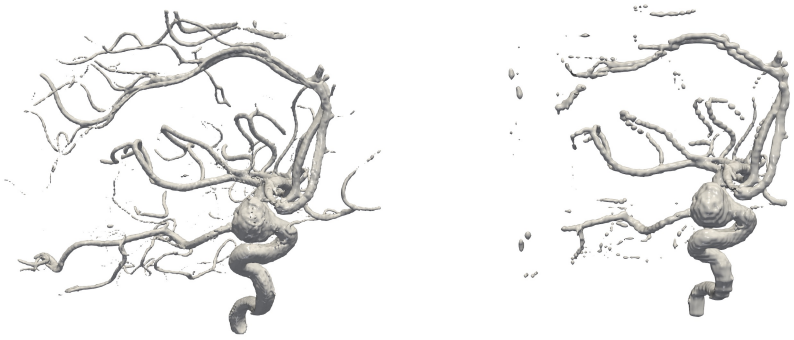
where

$$\tilde{f}_i = \sqrt{P} P_r^{(i)} k_i^{\log_2(P)} f^{\log_2(P)+1}.$$

Proofs for both propositions, mainly derived by co-authors, can be found in Beaupère et al. [16].

4.3.2 Numerical experiments

The following presents several numerical experiments using the HOQRTP. We first apply it to a 3D medical image representing an aneurysm. Each figure reveals the isosurface view of the tensor, note that empty spaces do not indicate sparsity in the tensor. Figure 4.3 compares the original $256 \times 256 \times 256$ tensor on the left to the rank-$(64 \times 64 \times 64)$ compressed tensor on the right.



(a) Uncompressed.

(b) Rank-$(64, 64, 64)$ approximation on 64 processors.

Figure 4.3: Isosurface views of 256×256×256 aneurysm 3D image from `https://tc18.org/3D_images.html`.

Figure 4.4 displays the singular values of the 1-mode unfolding of the original aneurysm tensor, as well as compressed tensor with different number of processors. The associated Table 4.1 provides the minimum, maximum and average ratio of the approximated singular values with respect to the original one. Note that this layout was already used in Figure 3.2 and Figure 3.3.

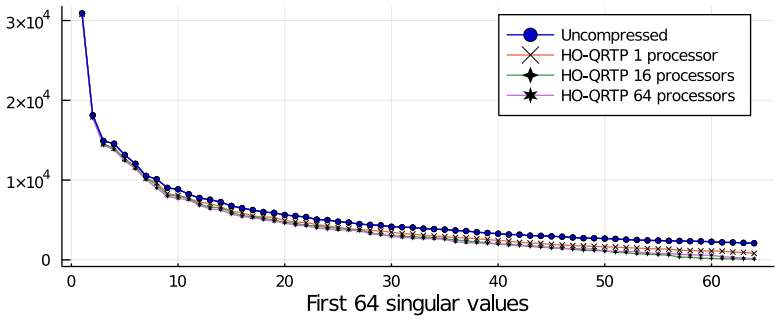


Figure 4.4: Singular values of the 1-mode unfolding matrix of the uncompressed and compressed aneurysm 3D image.

Executing HOQRTP with only one processor is equivalent to using the truncated QRCP. We observe that the average singular value ratio is reasonably high, meaning that the singular values are quite well approximated with HOQRTP. It is confirmed by the image, the major parts of the isosurface being conserved by the compression. We note that the accuracy decreases with the number of processors, in compliance with the trend of the error bound.

Proc count	$\max_i \frac{\sigma_i(\tilde{A}_1)}{\sigma_i(A_1)}$	$\min_i \frac{\sigma_i(\tilde{A}_1)}{\sigma_i(A_1)}$	$\frac{1}{64}\sum_i \frac{\sigma_i(\tilde{A}_1)}{\sigma_i(A_1)}$
1	0.996	0.385	0.763
16	0.996	0.0440	0.636
64	0.995	0.0715	0.642

Table 4.1: Maximum, minimum and mean ratio $\frac{\sigma_i(\tilde{A}_1)}{\sigma_i(A_1)}$ where $\sigma_i(A_i)$ (*resp* $\sigma_i(\tilde{A}_i)$) is the ith largest singular value of the 1-mode unfolding of the uncompressed (*resp* compressed) aneurysm 3D image (see Figure 4.4).

We now analyse the runtime of HOQRTP on a cluster. Consider a tensor of size $1024 \times 1024 \times 1024$. Figure 4.5 distributes the strong scaling runtime of the rank-$(16, 16, 16)$ compression of the tensor when increasing the number of processors from 8 to 512. The QRTP slice of each bar is the sum of the 3 execution times of the QRTP algorithm (one for each unfolding). The Matrix Product slice of each bar is the cumulated time of the matrix products used to compute the factor matrix (i.e. to compute $\mathbf{R}_{11}$ and $\mathbf{R}_{12}$ at line 11 of Algorithm 15). The Other operations slice corresponds mainly to the SVD of $\mathbf{R}_{11}$ and $\mathbf{R}_{12}$ and the projection of the tensor to compute the core tensor.

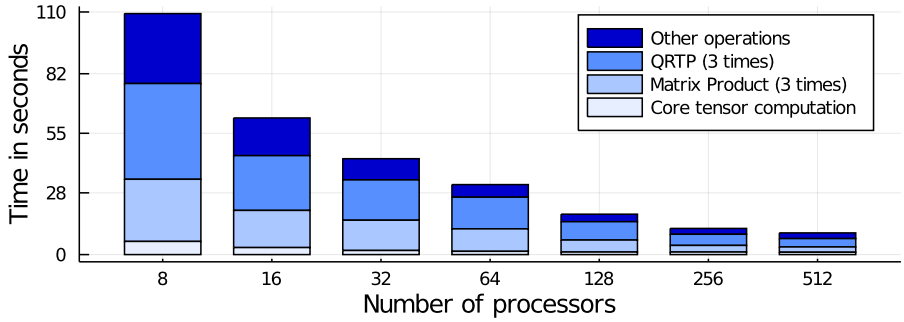


Figure 4.5: Strong scaling performance and speedup of HOQRTP applied to a logarithm tensor of size 1024×1024×1024 on varying numbers of processors.

Figure 4.6 displays the weak scaling experiment. The size of the input tensor increases with the number of processors such that on each processor the local subtensor has a fixed size of $256 \times 256 \times 256$. The approximation ranks are fixed to $(16, 16, 16)$. The different runtime slices have the same meaning as in the previous figure.

These figures show very similar scaling as in previous study of QRTP, in Section 3.6. Even though the weak scaling is not a straight line, it tends towards it as the number of processors increases. The QRTP operations do not seem to be the main bottleneck, as they scale similarly to other parts involving ScaLAPACK routines such as `pdgeqpf`, `pdormqr` and `pdgemm`.

4.4 Sequentially Truncated HOQRTP

In this section we consider the sequentially truncated HOSVD as introduced in Vannieuwenhoven et al. [114], in which an approximation is computed for each

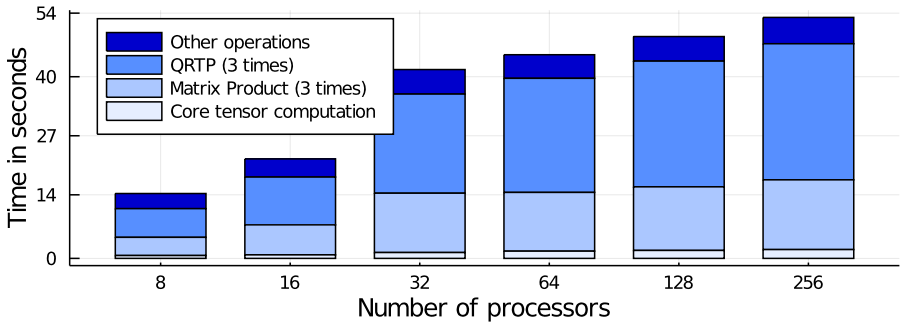


Figure 4.6: Weak scaling performance and speedup of HOQRTP applied to logarithm tensors of varying sizes on a varying numbers of processors, where the subtensor on each processor has a fixed size 256×256×256.

mode iteratively. Thus, the size of the tensor is progressively reduced and hence a lower computational cost is obtained with respect to HOSVD. Algorithm 16 is a sequentially truncated version of HOSVD in which the compression of each unfolding matrix is obtained through QRTP. We refer to this algorithm as sequentially truncated HOQRTP. The main difference with Algorithm 15 is step 15, which appears inside the for loop at line 14. In this manner the tensor is projected onto a subspace corresponding to a mode at each iteration.

Updating the tensor (line 14) is done with limited communication. The tensor $\mathcal{A}$ is distributed amongst processors, while the matrix $\mathbf{U}_i$ is broadcast to all processors. Let (I, J) be the coordinate of a processor in the i-unfolding processor tensor, $\mathcal{A}^{(I,J)}$ be the local tensor unfolding, and $\mathbf{U}_i^{(I)}$ be the columns of $\mathbf{U}_i$ corresponding to the rows of $\mathcal{A}_i^{(I,J)}$ in $\mathcal{A}_i$. Then the projection is done in two steps:

1. Locally on processor (I, J): Compute $\mathcal{A}_i^{(I,J)} \leftarrow \left(\mathbf{U}_i^{(I)}\right)^\intercal \mathcal{A}_i^{(I,J)}$.

2. Sum-reduce to the first processor of the column:

$$\mathcal{A}_i(1, J) = \sum_{I=1}^{d_i} A_i(I, J).$$

The approximated tensor is then stored on a subset of processors (the first row of the i-unfolded processor tensor). The algorithm continues on these processors, thus avoiding communication for redistributing data and taking into account that the amount of computation decreases for the subsequent modes. The local tensor dimensions, denoted by $b_1 \times \ldots \times b_d$, would increase if $k_i > b_i$ for some i, leading to memory problems. Thus, the experiments are limited to the case $k_i \leq b_i$. Another option would be to redistribute the data amongst all the processors after each iteration, but this would lead to extra communication. We do not explore this option in this work.

Algorithm 16 ST-HOQRTP: Low-rank approximation of a tensor

Require: A tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \ldots \times n_d}$ and a set of ranks $\{k_i\}_{1 \leq i \leq d}$

Distribute the sub-tensors of $\mathcal{A}$ amongst the processors
for i in $1 : d$ **do**
  Extract i-mode unfolding of sub-tensor on each processor
  if unfolding matrix $\mathcal{A}_i$ is tall **then**
    Unfold processors according to the i-mode unfolding so that $\mathcal{A}_{i\backslash\backslash}$ is logically distributed on the processor grid
    Run QRTP (for rank k_i) on $\mathcal{A}_{i\backslash\backslash}$ to get $\mathcal{A}_{i\backslash\backslash}\mathbf{\Pi} = \mathbf{QR}$
    Truncate $\mathbf{Q}$ to get the factor matrix $\mathbf{U}_i = \mathbf{Q}[:, 1 : k_i]$
  else if /* $\mathcal{A}_i$ is wide */ **then**
    Unfold processors according to the i-mode unfolding so that $\mathcal{A}^\intercal_{i\backslash\backslash}$ is logically distributed on the processor grid
    Run QRTP (for rank k_i) on $\mathcal{A}^\intercal_{i\backslash\backslash}$ to get $\mathcal{A}^\intercal_{i\backslash\backslash}\mathbf{\Pi} = \mathbf{QR}$
    Run SVD on $\mathbf{\Pi}_1\mathbf{R}_{11}^\intercal + \mathbf{\Pi}_2\mathbf{R}_{12}^\intercal$ to get $\mathbf{\Pi}_1\mathbf{R}_{11}^\intercal + \mathbf{\Pi}_2\mathbf{R}_{12}^\intercal = \mathbf{U\Sigma V}^\intercal$
    Let $\mathbf{U}_i = \mathbf{U}[:, 1 : k_i]$ be the factor matrix associated with the i-mode unfolding
  end if
  Update the tensor $\mathcal{A} \leftarrow \mathcal{A} \times_i \mathbf{U}_i^\intercal$
end for
$\mathcal{C}, \mathbf{U}_1, \mathbf{U}_2, \ldots, \mathbf{U}_d$

Ensure: A Tucker decomposition (core tensor $\mathcal{C}$ and factor matrices $\{\mathbf{U}_i\}_{1 \leq i \leq d}$ such that $\tilde{\mathcal{A}} = \mathcal{C} \times_1 \mathbf{U}_1\mathbf{U}_1^\intercal \times_2 \cdots \times_d \mathbf{U}_d^\intercal$)

4.4.1 Error bound

In this section we show that ST-HOQRTP has the same error bound as HO-QRTP.

Theorem 13. *Given a set of ranks $\{k_i\}_{1 \leq i \leq d}$ and $f > 1$, the QRTP factorizations of the unfolding matrices of the tensor $\mathcal{A}$ are such that the low-multilinear rank tensor approximation $\widetilde{\mathcal{A}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ generated by Algorithm 16 satisfies the error bound,*

$$\|\mathcal{A} - \tilde{\mathcal{A}}\|_F^2 \leq \sum_{i=1}^{d} \sum_{j=k_i+1}^{r_i} \sigma_j^2(\mathcal{A}_i)[1 + \tilde{f}_i^2(n_i - k_i)],$$

where

$$\tilde{f}_i = \sqrt{P^{(i)}} P_r^{(i)} k_i^{\log_2(P^{(i)})} f^{\log_2(P^{(i)})+1},$$

and r_i and $\sigma_j(\mathcal{A}_i)$ are respectively the rank and singular values of the i-mode unfolding $\mathcal{A}_i$, and where $P_r^{(i)}$ is the height of the processor grid related to the i^{th} unfolding and $P^{(i)} = \prod_{j=i}^{d} P_r^{(j)}$ is the total number of processors still involved at step i. Then at step i the unfolding lies on the processor grid $P_r^{(i)} \times P_r^{(i+1)} P_r^{(i+2)} \ldots P_r^{(d)}$.

Proof. As in the proof of Theorem 12 we first consider the case of $\mathcal{A}_i$ being tall. Using the Theorem 5.1 from Vannieuwenhoven et al. [114] with the QRTP

factor matrices as the orthogonal projection matrices we can link the error and the residual of each QRTP operation as follows.

$$\begin{aligned}\|\mathcal{A}-\tilde{\mathcal{A}}\|_F^2 &= \sum_{i=1}^{d} \left\|\mathcal{A}\times_1 \bar{\mathbf{Q}}_1^\intercal\bar{\mathbf{Q}}_1 \ldots \times_{i-1} \bar{\mathbf{Q}}_{i-1}^\intercal\bar{\mathbf{Q}}_{i-1} \times_i (\mathbf{I}-\bar{\mathbf{Q}}_i^\intercal\bar{\mathbf{Q}}_i)\right\|_F^2 \\ &= \sum_{i=1}^{d} \left\|(\mathbf{I}-\bar{\mathbf{Q}}_i\bar{\mathbf{Q}}_i^\intercal)(\mathcal{A}\times_1 \bar{\mathbf{Q}}_1^\intercal\bar{\mathbf{Q}}_1 \ldots \times_{i-1} \bar{\mathbf{Q}}_{i-1}^\intercal\bar{\mathbf{Q}}_{i-1})_{i\backslash\backslash}\right\|_F^2 \quad \text{(Th. 11)} \\ &= \sum_{i=1}^{d} \left\|\mathbf{R}_i^{(22)}\right\|_F^2, \end{aligned} \tag{4.6}$$

where $R_i^{(22)}$ is the residual matrix obtained after the QRTP compression i. Note that unlike in the proof of Theorem 12 we obtain a strict equality. In the case of a wide unfolding, starting again from expression Eq. (4.6) and substituting Eq. (4.4), we obtain

$$\begin{aligned}\|\mathcal{A}-\tilde{\mathcal{A}}\|_F^2 &= \sum_{i=1}^{d} \left\|(\mathbf{I}-\bar{\mathbf{Q}}_i\bar{\mathbf{Q}}_i^\intercal)(\mathcal{A}\times_1 \bar{\mathbf{Q}}_1^\intercal\bar{\mathbf{Q}}_1 \ldots \times_{i-1} \bar{\mathbf{Q}}_{i-1}^\intercal\bar{\mathbf{Q}}_{i-1})_{i\backslash\backslash}\right\|_F^2 \\ &= \sum_{i=1}^{d} \left\|(\mathbf{I}-\bar{\mathbf{Q}}_i\bar{\mathbf{Q}}_i^\intercal)\mathbf{\Pi}\left(\mathbf{R}_i^{(22)\intercal}\right)\mathbf{Q}^\intercal\right\|_F^2 \\ &= \sum_{i=1}^{d} \left\|\mathbf{R}_i^{(22)}\right\|_F^2. \end{aligned}$$

By using the bounds of QRTP given in Eq. (3.11) we obtain,

$$\begin{aligned}\sum_{i=1}^{d} \left\|\mathbf{R}_i^{(22)}\right\|_F^2 &= \sum_{i=1}^{d}\sum_{j=1}^{r_i-k_i} \sigma_j^2(\mathbf{R}_i^{(22)}) \\ &\leq \sum_{i=1}^{d}\sum_{j=k_i+1}^{r_i} \sigma_j^2(\tilde{\mathcal{A}}_i)[1+\tilde{f}_i^2(n_i-k_i)],\end{aligned}$$

where $\tilde{\mathcal{A}}_i = (\mathcal{A}\times_1 \mathbf{U}_1 \ldots \times_{i-1} \mathbf{U}_{i-1})_i$ is the unfolding of the projected tensor at step i, $\tilde{f}_i = \sqrt{P^{(i)}}P_r^{(i)}k_i^{\log_2(P^{(i)})}f^{\log_2(P^{(i)})+1}$, since QRTP is applied on $\tilde{\mathcal{A}}_i$ and executed on a subset of processors $P^{(i)} = P_r^{(i)}P_r^{(i+1)}\ldots P_r^{(d)}$, as explained previously in this section.

We now prove that for any $k_i+1 \leq j \leq r_i$, $\sigma_j(\tilde{\mathcal{A}}_i) \leq \sigma_j(\mathcal{A}_i)$. Fix such a j and an i such that $1 \leq i \leq d$. As mentioned in Bader and Kolda [10, Sec. 4.2], $\tilde{\mathcal{A}}_i$ can be expressed as:

$$\tilde{\mathcal{A}}_i = \mathbf{U}_i\mathcal{A}_i(\mathbf{U}_1^\intercal \otimes \ldots \otimes \mathbf{U}_{i-1}^\intercal \otimes \mathbf{I}_{i+1} \otimes \ldots \otimes \mathbf{I}_d). \tag{4.7}$$

We use the singular value inequality from Theorem 2. Given two matrices $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{Q} \in \mathbb{R}^{p\times n}$, if $\mathbf{Q}$ has orthonormal columns, we obtain

$$\sigma_i(\mathbf{A}\mathbf{Q}^\intercal) \leq \sigma_i(\mathbf{A}). \tag{4.8}$$

The rightmost term $\mathbf{U}_1^\intercal \otimes \ldots \otimes \mathbf{U}_{i-1}^\intercal \otimes \mathbf{I}_{i+1} \otimes \ldots \otimes \mathbf{I}_d$ of Eq. (4.7) has orthonormal columns because each term in the Kronecker product has orthonormal columns (see e.g. Tucker [111, Prop. (f)]). By substituting it to $\mathbf{Q}$ and $\mathbf{U}_i\mathcal{A}_i$ to $\mathbf{A}$ in Eq. (4.8) and then using its transposed version we obtain that $\sigma_j(\tilde{\mathcal{A}}_i) \leq \sigma_j(\mathbf{U}_i\mathcal{A}_i) \leq \sigma_j(\mathcal{A}_i)$. □

We note that the bound of ST-HOQRTP is slightly smaller than the bound of HOQRTP. Indeed, $\bar{f}_i$ in the bound of ST-HOQRTP depends on the number of processors used for each i-mode unfolding, which is decreasing as the algorithm proceeds. However, Corollary 2 still holds for ST-HOQRTP by replacing $\tilde{f}_i$ with $\bar{f}_i$ from Theorem 13.

4.4.2 Cost of ST-HOQRTP

Let $\mathcal{A} \in \mathbb{R}^{n\times\cdots\times n}$ and equidimensional tensor. Consider P processors forming an equidimensional processor tensor, i.e. there exists l such that $P = l^d$. Using the cost of QRTP given in Section 3.5, the costs of the sequentially truncated version of the algorithm is:

$$\#\text{flops} = \sum_{i=1}^{d} \left[4k^i \left(\frac{n}{\sqrt[d]{P}} \right)^{d-i+1} + K_1^{(i)} k^3 + K_2 k^{i+1} \left(\frac{n}{\sqrt[d]{P}} \right)^{d-i} \right].$$

where

$$K_1^{(i)} = 16 \log_2 P^{\frac{d-i+1}{d}} + 8 \log_2 P_r \left[1 + \frac{5 \log_2 P_r}{3} \right] + \frac{16}{3} \left[5 \log_2 P_r - 1 \right],$$
$$K_2 = 8(\log_2 P_r + 1).$$

and

$$\begin{aligned} \#\text{messages} &= \sum_{i=1}^{d} \log_2 P^{\frac{d-i+1}{d}} (1 + \log_2 P_r) \\ &= \left(d - \frac{d-1}{2} \right) \log_2 P (1 + \log_2 P_r). \end{aligned}$$

In terms of computation cost, the dimension of the problem decreases at each iteration of the algorithm, and the cost of the first iteration dominates the overall cost. In terms of number of messages, the dominant factor is decreased by almost a factor of 2 with respect to HOQRTP.

4.4.3 Numerical experiments

In this section we present results obtained with the sequentially truncated version of the algorithm and compare them with HOQRTP and ST-HOSVD. For ST-HOSVD we use TuckerMPI[1] presented in Austin et al. [8]. The input tensor is the same tensor as in Figure 4.5.

Figure 4.7 presents weak scaling results, for which the subtensor on each core has constant size 256×256×256, for a number of cores increasing from 8 to 512. The multilinear rank is 128 in this experiment. It can be seen that

[1] Commit number `2b3c356` from https://gitlab.com/tensors/TuckerMPI

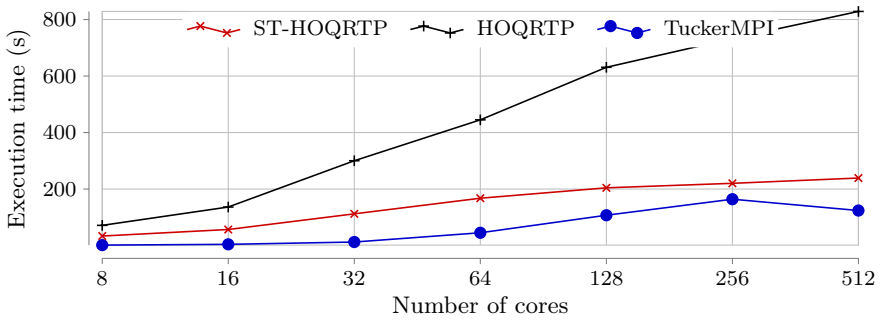


Figure 4.7: Execution time of HOQRTP and ST-HOQRTP having as input a distributed logarithm tensor where each processor owns a subtensor of size 256×256×256. The final core tensor is of size 128×128×128.

ST-HOQRTP is faster by a factor of almost 4 on 512 cores with respect to HOQRTP, and also scales better than HOQRTP for 128 cores and more. On the other hand, TuckerMPI is faster than HOQRTP and ST-HOQRTP for this example. Even if QRTP is less expensive than SVD, TuckerMPI uses a Gram matrix, which leads to computing the SVD of a smaller matrix in this case. As an example, for an equidimensional d-order tensor of size $n \times \ldots \times n$ the first SVD of TuckerMPI will be computed on a matrix of size $n \times n$, whereas in ST-HOQRTP the first QRTP will be computed on a $n^d \times n$ matrix. Figure 4.8 gives the corresponding Frobenius error $\frac{\|\mathcal{A}-\mathcal{A}_k\|_F}{\|\mathcal{A}\|_F}$, where $\mathcal{A}$ is the input tensor and $\mathcal{A}_k$ its compressed version. We see that ST-HOQRTP and HOQRTP have very similar accuracy. The accuracy for TuckerMPI is not represented here because for this tensor and a compression rank of 128 the error is exactly zero.

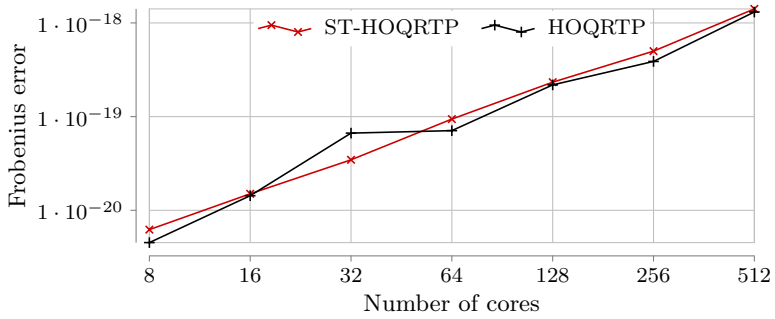


Figure 4.8: Frobenius relative error of HOQRTP and ST-HOQRTP having as input a distributed logarithm tensor where each processor owns a subtensor of size 256×256×256. The final core tensor is of size 128×128×128.

We consider the experiment illustrated in Figure 4.8. For this experiment, the value of the bound $\sqrt{1+\max_i(f_i^2(n_i-k_i))}$ given in Corollary 2, which holds for HOQRTP, goes from 6.75×10^5 for 8 cores to 4.09×10^{12} for 512 cores. Therefore, this bound is pessimistic.

We now study the case of tensors having unbalanced dimensions. In Figure 4.9 we consider a 4-order tensor having one large mode varying from 256 to 16384 and three small modes of constant size 32. While increasing the number

of processors from 8 to 32 to investigate the scaling of the approximation time, we also increase the size of the large mode. The data is distributed such that each local subtensor has size $512 \times 32 \times 32 \times 32$.

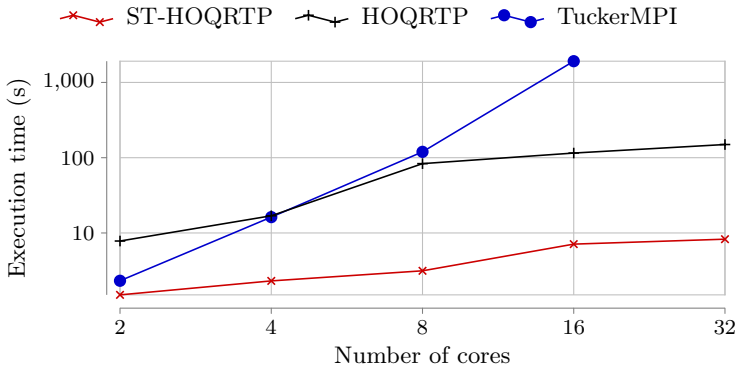


Figure 4.9: Compressing a logarithm tensor of size $2^a \times 32 \times 32 \times 32$ on 2^p cores where $1 \leq p \leq 5$ and $a = p + 9$ to multilinear rank 16

In this experiment ST-HOQRTP times stay under 10 seconds while HOQRTP times increase above 100 seconds and TuckerMPI times increase above 1500 seconds. When running on 32 cores TuckerMPI lasted more than 3 hours. This shows that for particular cases ST-HOQRTP computes a Tucker decomposition in a smaller amount of time compared to other algorithms. It should be noted that in this case, TuckerMPI is parameterized with a fixed rank such that it can be compared with ST-HOQRTP, meaning that it may be better tuned for this specific problem.

To study the difference between ST-HOQRTP and TuckerMPI on a less extreme experiment, we compare the two algorithms on a logarithm tensor having two large modes varying from 128 to 2048 and two small modes of fixed size 32. Figure 4.10 gives the execution time of ST-HOQRTP and TuckerMPI for these cases, increasing the number of processors along with the size of the large modes to keep the local tensors size constant (when we double the number of cores, we first double the first mode, then the second mode and so on). The data is distributed such that each local subtensor has size $128 \times 128 \times 32 \times 32$. In these figures we can see that ST-HOQRTP is 3 times faster for compressing the tensor on 256 cores and 10 times faster on 512 cores.

To conclude, ST-HOQRTP is faster by large factors with respect to HOQRTP, and is faster than TuckerMPI in the case of tensors having unbalanced dimensions. In the case of an equidimensional tensor TuckerMPI is faster, mainly due to the use of a Gram matrix for computing the SVD of unfolding matrices.

4.5 Conclusion

In this chapter we introduced Higher-Order QR with Tournament Pivoting (HOQRTP). HOQRTP is a parallel variant of HOSVD that uses QRTP (instead of SVD) and partitioned unfolding to extract factor matrices from the unfolding matrices of a dense distributed tensor. We have shown that HOQRTP provides good error bounds for tensor approximations, and exhibits good strong

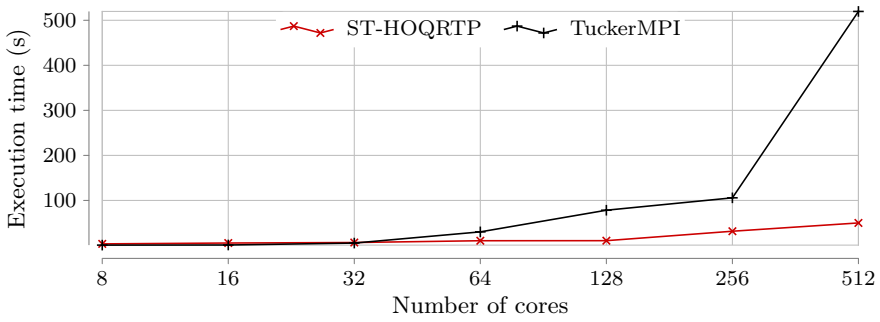


Figure 4.10: Compressing a logarithm tensor of size $2^a \times 2^b \times 32 \times 32$ on 2^p cores where $3 \leq p \leq 9$, $a = \lceil \frac{p}{2} \rceil + 7$ and $b = \lfloor \frac{p}{2} \rfloor + 7$ to rank 16.

and weak scaling performance. We derived this algorithm into ST-HOQRTP, an algorithm that exploits the projections to provide lower execution costs while retaining a similar error bound. From the numerical experiments it was shown that TuckerMPI, the reference software for our analysis, gives better performance than ST-HOQRTP for equidimensional tensors. In cases of unbalanced mode sizes, however, TuckerMPI performs worse. ST-HOQRTP is thus useful to compute the Tucker decomposition of tensors with unbalanced mode sizes. We have also shown that partitioned unfolding provides a method to distribute tensor data amongst a set of processors such that each factor matrix can be computed without additional communication. Although the SVD decomposition provides the best low-rank approximation, the scalability of QRTP makes it a good candidate for the compression of a large tensor.

Conclusion and perspectives

This thesis introduced several scalable distributed algorithms to perform compression through low rank approximation of dense arbitrary matrices. The first part presented a block version of SRHT, thereby demonstrating sampling of a large matrix in a distributed manner. Block SRHT exhibits similar accuracy with Gaussian sampling, while better speedup characteristics on large scale matrices. It is further adapted to compute the Nyström approximation, a matrix low rank approximation. By illustrating that SRHT can be efficiently parallelized, this new algorithm stands out as a new fast parallel method to perform randomization on a large dense matrix. Afterwards was presented QRTP, a parallel method to compute the truncated QRCP decomposition of a large distributed matrix. We proved that the previous error bound used in the literature still applies to this new method, with a minor modification. Through various highly parallel steps, the complete implementation provides an efficient method to select pivot columns from a very large matrix with a low amount of communications. To select a small number of columns, this new algorithm scales on a large number of nodes more efficiently than the state-of-the-art counterpart. The pivot selection being a major bottleneck for various linear algebra routines, this new algorithm is expected to speedup their parallel execution on a large number of nodes, for a large input matrix. At the end, the QRTP method is adapted to be applied to tensors and compute the Tucker decomposition, in a way that the current state-of-the-art only operates with the truncated SVD. Using the truncated SVD is indeed very efficient on tensors with balanced dimensions, but on particular cases where the dimensions are very different, QRTP appears to be a reasonable alternative. The sequentially truncated version applies well to our algorithm, giving similar bounds and increasing further the speedup of our method. This algorithm enables to compute in parallel the Tucker decomposition of a large tensor without the use of the SVD or Gram matrices, hence making it efficient at large scale.

We now discuss the perspectives of this work. First are presented the direct possible improvements of our work, then we discuss supplementary topics of interest. Concerning the tournament pivoting, the next step is to couple QRTP with the idea inside randomized QRCP. It would cumulate advantages from both methods. Indeed, we can substitute QRTP to `pdgeqpf` in the RQRCP algorithm (see Algorithm 9), which was identified as the longest step in our experiments. The QRTP is then executed on the sketch of the input matrix, having less rows hence requiring less resources. Concerning the computation of the Tucker decomposition, randomization is also an interesting approach as suggested in the recent contributions on this topic [91, 90, 28]. These contributions propose to substitute randomized SVD (as given in Halko et al. [60]) to the truncated

SVD in HOSVD and ST-HOSVD algorithms. We suggest to go further and use a randomized QRTP method to compute each factor matrix. Additionally, applying our methods to the tensor train is promising as well, especially speeding up tensor train manipulations such as normalization, compression, addition, and contraction. Existing work introduces randomization for the tensor train compression [27, 3]. As an example, applying QRTP on different matricizations of a tensor train would produce useful projection matrices to decrease the bond dimensions connecting two tensors. By preventing these bond dimensions from increasing to large values, especially in the middle of the train, its manipulation would remain efficient. As another example, when computing the inner product of two or three tensor trains, for example to perform the alternate least squares scheme (ALS), we could use a randomized inner product to speed up each ALS iteration. This is already addressed for the ALS in Tucker format [84, 83]. Closely related, Oseledets [92] uses randomization to check the error of the tensor train compression. Molecular simulations presented in the first chapter expose several research problems, such as computing the tensor train representation of the Hamiltonian or computing the inner product of tensors in the alternate least squares scheme, where these optimized operations are key to improve the efficiency of the simulation.

More generally, this study raises several topics of interest that are part of our scientific domain. On the topic of strong rank revealing QR, it would be interesting to find a tight error bound, perhaps it should depend on the properties of the input matrix. This would enable to compute the low rank approximation through another approach, by minimizing the approximation rank and fixing the error, which is already possible with the truncated SVD. Another point is the opportunity of software development behind these methods. There is probably a way to expose these algorithms in a common scientific library, compressing a matrix or a tensor in various ways and manipulating it. Furthermore, a benchmark to evaluate and compare practically the numerical performance of a low rank approximation would certainly improve the scientific debate. The algorithms presented in this work offer scalable methods to compute the low rank approximation of any matrix. Nowadays, numerous domains manipulate very large matrices and tensors, such as climatology, medical imagery or molecular simulations—involved in drug design, new material design. By representing the data in a reduced format, and distributing it on a computer cluster, our novel methods are deemed to greatly increase the computational capabilities of the current numerical methods, hence extending the outreach in these domains.

Appendix A

Source code

A.1 Subsampled Randomized Hadamard Transform

```
using Hadamard
using Random
using TimerOutputs
using LinearAlgebra
using Dates

#=
Compute the Subsampled Randomized Hadamard Transform of A with a
    sampling size of l.
Return B = Omega x A where A is mxn, Omega is lxm and B is lxn
Omega = sqrt(m/l) Dr R H Dl
* Dl and Dr are diagonal matrices of random signs
* H is the Hadamard matrix
* R is a sampling matrix, selecting l rows out of m

A should be stored in column major format.

In the context of a block SRHT: (Omega_1 ... Omega_p) (A_1 ... A_p)
    ^\tr, then the
random number generator seeds should be passed to all srht() calls.
param global_seed: same for all block
param block_seed: different for each block
=#
function srht(A::Array{Float64,2}, l::Int; to::TimerOutput=
    TimerOutput(), global_seed::UInt=rand(UInt), block_seed::UInt=
    rand(UInt))::Array{Float64,2}
  rng_global = MersenneTwister(global_seed)
  rng_block = MersenneTwister(block_seed)
  @assert l <= size(A)[1]
  @timeit to "init rand" begin
    # Rademacher vector
    Dr::Array{Int} = rand(rng_block, (-1,1), size(A)[1])
    Dl::Array{Int} = rand(rng_block, (-1,1), l)

    # Sampling without replacement gives better results in practice
        for this usecase

    # Random sampling permutation without replacement
    #P::Array{Int} = randperm(rng_global, size(A)[1])[1:l]
```

```
    # Random sampling permutation with replacement
    P::Array{Int} = rand(rng_global, 1:size(A)[1], l)

    # Rescaling
    scale::Float64 = size(A)[1]/sqrt(l)
  end

  # X = Dr A
  @timeit to "Dr" lmul!(Diagonal(Dr), A)

  # X1 = H X
  @timeit to "H" fwht_natural!(A, 1)

  # X2 = R X1
  @timeit to "R" B::Array{Float64,2} = scale .* A[P,:]

  # Compute B = R Dl H Dr A
  @timeit to "Dl" lmul!(Diagonal(Dl), B)

  return B
end
```

A.2 Gaussian sampling

```
using Random
using Distributions

function sketch_right_gaussian(A::Array{Float64,2},l::Int; seed::
    UInt64=rand(UInt64))::Array{Float64,2}
  m,n = size(A)
  d = Normal(0, 1 / sqrt(l))
  Ω' = rand(MersenneTwister(seed), d, (n, l))
  return A * Ω
end

function sketch_left_gaussian(A::Array{Float64,2},l::Int; seed::
    UInt64=rand(UInt64))::Array{Float64,2}
  m,n = size(A)
  d = Normal(0, 1 / sqrt(l))
  Ω = rand(MersenneTwister(seed), d, (m, l))
  return Ω' * A
end
```

Bibliography

[1] D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66 (4):671–687, June 2003. doi: 10.1016/S0022-0000(03)00025-4.

[2] N. Ailon and B. Chazelle. The Fast Johnson–Lindenstrauss Transform and Approximate Nearest Neighbors. *SIAM Journal on Computing*, 39 (1):302–322, Jan. 2009. doi: 10.1137/060673096.

[3] H. Al Daas, G. Ballard, P. Cazeaux, E. Hallman, A. Międlar, M. Pasha, T. W. Reid, and A. K. Saibaba. Randomized algorithms for rounding in the tensor-train format. *SIAM Journal on Scientific Computing*, 45(1): A74–A95, 2023. doi: 10.1137/21M1451191.

[4] M. Ali and A. Nouy. Approximation with Tensor Networks. Part I: Approximation Spaces. Feb. 2021, arXiv: 2007.00118.

[5] P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, and C. Weisbecker. Improving multifrontal methods by means of block low-rank representations. *SIAM Journal on Scientific Computing*, 37(3): A1451–A1474, 2015. doi: 10.1137/120903476.

[6] P. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary. On the complexity of the block low-rank multifrontal factorization. *SIAM Journal on Scientific Computing*, 39(4):A1710–A1740, 2017. doi: 10.1137/16M1077192.

[7] E. Anderson, editor. *LAPACK users' guide*. Software, environments, tools. Society for Industrial and Applied Mathematics, Philadelphia, 3rd ed edition, 1999.

[8] W. Austin, G. Ballard, and T. G. Kolda. Parallel Tensor Compression for Large-Scale Scientific Data. *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016. doi: 10.1109/ipdps.2016.67.

[9] Y. Azar, A. Fiat, A. Karlin, F. McSherry, and J. Saia. Spectral analysis of data. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing - STOC '01*, pages 619–626, Hersonissos, Greece, 2001. ACM Press. doi: 10.1145/380752.380859.

[10] B. W. Bader and T. G. Kolda. Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. *ACM Transactions on Mathematical Software*, 32(4):635–653, Dec. 2006. doi: 10.1145/1186785.1186794.

[11] A. Baiardi and M. Reiher. The density matrix renormalization group in chemistry and molecular physics: Recent developments and new challenges. *The Journal of Chemical Physics*, 152(4):040903, Jan. 2020. doi: 10.1063/1.5129672.

[12] O. Balabanov, M. Beaupère, L. Grigori, and V. Lederer. Block subsampled randomized Hadamard transform for low-rank approximation on distributed architectures. 2022, arXiv: 2210.11295.

[13] G. Ballard, A. Klinvex, and T. G. Kolda. TuckerMPI: A Parallel C++/MPI Software Package for Large-scale Data Compression via the Tucker Tensor Decomposition. *ACM Transactions on Mathematical Software*, 46(2):1–31, June 2020. doi: 10.1145/3378445.

[14] O. Beaumont, L. Eyraud-Dubois, and T. Lambert. A New Approximation Algorithm for Matrix Partitioning in Presence of Strongly Heterogeneous Processors. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 474–483, Chicago, IL, USA, May 2016. IEEE. doi: 10.1109/IPDPS.2016.32.

[15] O. Beaumont, L. Eyraud-Dubois, and M. Verite. 2D Static Resource Allocation for Compressed Linear Algebra and Communication Constraints. In *2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pages 181–191, Pune, India, Dec. 2020. IEEE. doi: 10.1109/HiPC50609.2020.00032.

[16] M. Beaupère, D. Frenkiel, and L. Grigori. Higher-order qr with tournament pivoting for tensor compression. *SIAM Journal on Matrix Analysis and Applications*, 44(1):106–127, 2023. doi: 10.1137/20M1387663.

[17] M. Bebendorf. Approximation of boundary element matrices:. *Numerische Mathematik*, 86(4):565–589, Oct. 2000. doi: 10.1007/PL00005410.

[18] L. S. Blackford and S. for Industrial and Applied Mathematics, editors. *ScaLAPACK user's guide*. Software, environments, tools. SIAM, Philadelphia, 1997. doi: 10.1137/1.9780898719642.

[19] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, 3(null):993–1022, Mar. 2003. Publisher: JMLR.org.

[20] C. Boutsidis and A. Gittens. Improved Matrix Algorithms via the Subsampled Randomized Hadamard Transform. *SIAM Journal on Matrix Analysis and Applications*, 34(3):1301–1340, Jan. 2013. doi: 10.1137/120874540.

[21] P. Businger and G. H. Golub. Linear least squares solutions by householder transformations. *Numerische Mathematik*, 7(3):269–276, June 1965. doi: 10.1007/BF01436084.

[22] A. Carasso. Determining Surface Temperatures from Interior Observations. *SIAM Journal on Applied Mathematics*, 42(3):558–574, June 1982. doi: 10.1137/0142040.

[23] J. D. Carroll and J.-J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika*, 35(3):283–319, Sept. 1970. doi: 10.1007/BF02310791.

[24] G. K.-L. Chan, A. Keselman, N. Nakatani, Z. Li, and S. R. White. Matrix product operators, matrix product states, and ab initio density matrix renormalization group algorithms. *The Journal of Chemical Physics*, 145 (1):014102, July 2016. doi: 10.1063/1.4955108.

[25] T. F. Chan. Rank revealing QR factorizations. *Linear Algebra and its Applications*, 88-89:67–82, Apr. 1987. doi: 10.1016/0024-3795(87)90103-0.

[26] T. F. Chan and P. C. Hansen. Computing truncated singular value decomposition least squares solutions by rank revealing QR-factorizations. *Society for Industrial and Applied Mathematics. Journal on Scientific and Statistical Computing*, 11(3):519–530, 1990. doi: 10.1137/0911029.

[27] M. Che and Y. Wei. Randomized algorithms for the approximations of Tucker and the tensor train decompositions. *Advances in Computational Mathematics*, 45(1):395–428, Feb. 2019. doi: 10.1007/s10444-018-9622-8.

[28] M. Che, Y. Wei, and H. Yan. Randomized algorithms for the low multilinear rank approximations of tensors. *Journal of Computational and Applied Mathematics*, 390:113380, July 2021. doi: 10.1016/j.cam.2020.113380.

[29] H. Chen, J. Zhao, Q. Luo, and Y. Hou. Distributed randomized singular value decomposition using count sketch. In *2017 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, pages 187–191, Shenzhen, Dec. 2017. IEEE. doi: 10.1109/SPAC.2017.8304273.

[30] J. Choi, X. Liu, and V. Chakaravarthy. High-Performance Dense Tucker Decomposition on GPU Clusters. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 543–553, Dallas, TX, USA, Nov. 2018. IEEE. doi: 10.1109/SC.2018.00045.

[31] B. Cobb, H. Kolla, E. Phipps, and U. V. Çatalyürek. FIST-HOSVD: fused in-place sequentially truncated higher order singular value decomposition. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, pages 1–11, Basel Switzerland, June 2022. ACM. doi: 10.1145/3539781.3539798.

[32] S. Dasgupta and A. Gupta. An Elementary Proof of a Theorem of Johnson and Lindenstrauss. *Random Struct. Algorithms*, 22(1):60–65, Jan. 2003. doi: 10.1002/rsa.10073. Place: USA Publisher: John Wiley & Sons, Inc.

[33] L. De Lathauwer, B. De Moor, and J. Vandewalle. On the Best Rank-1 and Rank-(R_1 , R_2 ,. . ., R_N) Approximation of Higher-Order Tensors. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1324–1342, Jan. 2000. doi: 10.1137/S0895479898346995.

[34] L. De Lathauwer, B. De Moor, and J. Vandewalle. A Multilinear Singular Value Decomposition. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1253–1278, Jan. 2000. doi: 10.1137/S0895479896305696.

[35] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal Parallel and Sequential QR and LU Factorizations. *SIAM Journal on Scientific Computing*, 34(1):A206–A239, Apr. 2008. doi: 10.1137/080731992.

[36] J. W. Demmel, L. Grigori, M. Gu, and H. Xiang. Communication avoiding rank revealing QR factorization with column pivoting. *SIAM Journal on Matrix Analysis and Applications*, 36(1):55–89, 2015. doi: 10.1137/13092157X.

[37] M. Dolfi, B. Bauer, S. Keller, A. Kosenkov, T. Ewart, A. Kantian, T. Giamarchi, and M. Troyer. Matrix product state applications for the ALPS project. *Computer Physics Communications*, 185(12):3430–3440, Dec. 2014. doi: 10.1016/j.cpc.2014.08.019.

[38] S. Dolgov and M. Stoll. Low-Rank Solution to an Optimization Problem Constrained by the Navier–Stokes Equations. *SIAM Journal on Scientific Computing*, 39(1):A255–A280, Jan. 2017. doi: 10.1137/15M1040414.

[39] P. Drineas, I. Kerenidis, and P. Raghavan. Competitive recommendation systems. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing - STOC '02*, page 82, Montreal, Quebec, Canada, 2002. ACM Press. doi: 10.1145/509907.509922.

[40] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering Large Graphs via the Singular Value Decomposition. *Machine Learning*, 56(1-3):9–33, July 2004. doi: 10.1023/B:MACH.0000033113.59016.96.

[41] Z. Drmač and Z. Bujanović. On the Failure of Rank-Revealing QR Factorization Software – A Case Study. *ACM Transactions on Mathematical Software*, 35(2):1–28, July 2008. doi: 10.1145/1377612.1377616.

[42] J. A. Duersch and M. Gu. Randomized QR with Column Pivoting. *SIAM Journal on Scientific Computing*, 39(4):C263–C291, Jan. 2017. doi: 10.1137/15M1044680.

[43] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, Sept. 1936. doi: 10.1007/BF02288367.

[44] H. El Bouhargani, A. Jamal, D. Beck, J. Errard, L. Grigori, and R. Stompor. MAPPRAISER: A massively parallel map-making framework for multi-kilo pixel CMB experiments. *Astronomy and Computing*, 39:100576, Apr. 2022. doi: 10.1016/j.ascom.2022.100576.

[45] D. Feldman, M. Schmidt, and C. Sohler. Turning Big Data into Tiny Data: Constant-Size Coresets for k-Means, PCA and Projective Clustering. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 1434–1453, USA, 2013. Society for Industrial and Applied Mathematics. event-place: New Orleans, Louisiana.

[46] S. Fischman, J. Pérez-Anker, L. Tognetti, A. Di Naro, M. Suppa, E. Cinotti, T. Viel, J. Monnier, P. Rubegni, V. del Marmol, J. Malvehy, S. Puig, A. Dubois, and J.-L. Perrot. Non-invasive scoring of cellular atypia in keratinocyte cancers in 3D LC-OCT images using Deep Learning. *Scientific Reports*, 12(1):481, Dec. 2022. doi: 10.1038/s41598-021-04395-1.

[47] S. Friedland, V. Mehrmann, A. Miedlar, and M. Nkengla. Fast low rank approximations of matrices and tensors. *The Electronic Journal of Linear Algebra*, 22, Jan. 2011. doi: 10.13001/1081-3810.1489.

[48] A. Frieze and R. Kannan. The regularity lemma and approximation schemes for dense problems. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 12–20, Burlington, VT, USA, 1996. IEEE Comput. Soc. Press. doi: 10.1109/SFCS.1996.548459.

[49] A. Frieze and R. Kannan. Quick Approximation to Matrices and Applications. *Combinatorica*, 19(2):175–220, Feb. 1999. doi: 10.1007/s004930050052.

[50] A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)*, pages 370–378, Palo Alto, CA, USA, 1998. IEEE Comput. Soc. doi: 10.1109/SFCS.1998.743487.

[51] F. Fröwis, V. Nebendahl, and W. Dür. Tensor operators: Constructions and applications for long-range interaction systems. *Physical Review A*, 81(6):062337, June 2010. doi: 10.1103/PhysRevA.81.062337.

[52] Y. Garniron, T. Applencourt, K. Gasperich, A. Benali, A. Ferté, J. Paquier, B. Pradines, R. Assaraf, P. Reinhardt, J. Toulouse, P. Barbaresco, N. Renon, G. David, J.-P. Malrieu, M. Véril, M. Caffarel, P.-F. Loos, E. Giner, and A. Scemama. Quantum Package 2.0: An Open-Source Determinant-Driven Suite of Programs. *Journal of Chemical Theory and Computation*, 15(6):3591–3609, June 2019. doi: 10.1021/acs.jctc.9b00176.

[53] W. Givens. Computation of plain unitary rotations transforming a general matrix to triangular form. *Journal of the Society for Industrial and Applied Mathematics*, 6(1):26–50, 1958. doi: 10.1137/0106004.

[54] G. H. Golub and C. F. Van Loan. *Matrix computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, fourth edition, 2013.

[55] N. Golubev, M. S. Zhdanov, and B. Chernobay. Three-dimensional inversion of array magnetotelluric data based on quasi-analytical approximation. In *SEG Technical Program Expanded Abstracts 2002*, pages 637–640. Society of Exploration Geophysicists, Jan. 2002. doi: 10.1190/1.1817333.

[56] S. Goreinov and E. Tyrtyshnikov. The maximal-volume concept in approximation by low-rank matrices. *Contemporary Mathematics*, 208, 01 2001. doi: 10.1090/conm/280/4620.

[57] L. Grasedyck, D. Kressner, and C. Tobler. A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen*, 36(1):53–78, Aug. 2013. doi: 10.1002/gamm.201310004.

[58] D. Gross and V. Nesme. Note on sampling without replacing from a finite collection of matrices. May 2010, arXiv: 1001.2738.

[59] M. Gu and S. C. Eisenstat. Efficient Algorithms for Computing a Strong Rank-Revealing QR Factorization. *SIAM Journal on Scientific Computing*, 17(4):848–869, July 1996. doi: 10.1137/0917055.

[60] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Review*, 53(2):217–288, Jan. 2011. doi: 10.1137/090771806. arXiv: 0909.4061.

[61] R. A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. Technical Report 16, UCLA, Dec. 1970.

[62] W. J. Hehre, R. F. Stewart, and J. A. Pople. Self-Consistent Molecular-Orbital Methods. I. Use of Gaussian Expansions of Slater-Type Atomic Orbitals. *The Journal of Chemical Physics*, 51(6):2657–2664, Sept. 1969. doi: 10.1063/1.1672392.

[63] F. L. Hitchcock. The Expression of a Tensor or a Polyadic as a Sum of Products. *Journal of Mathematics and Physics*, 6(1-4):164–189, Apr. 1927. doi: 10.1002/sapm192761164.

[64] S. Holtz, T. Rohwedder, and R. Schneider. The Alternating Linear Scheme for Tensor Optimization in the Tensor Train Format. *SIAM Journal on Scientific Computing*, 34(2):A683–A713, Jan. 2012. doi: 10.1137/100818893.

[65] S. Holtz, T. Rohwedder, and R. Schneider. On manifolds of tensors of fixed TT-rank. *Numerische Mathematik*, 120(4):701–731, Apr. 2012. doi: 10.1007/s00211-011-0419-7.

[66] Y. P. Hong and C.-T. Pan. Rank-revealing QR factorizations and the singular value decomposition. *Mathematics of Computation*, 58(197):213–213, Jan. 1992. doi: 10.1090/S0025-5718-1992-1106970-4.

[67] R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, 1 edition, Apr. 1991. doi: 10.1017/CBO9780511840371.

[68] A. S. Householder. Unitary triangularization of a nonsymmetric matrix. *J. ACM*, 5(4):339–342, oct 1958. doi: 10.1145/320941.320947.

[69] H. Huang, C. D. Sherrill, and E. Chow. Techniques for high-performance construction of Fock matrices. *The Journal of Chemical Physics*, 152(2): 024122, Jan. 2020. doi: 10.1063/1.5129452.

[70] J. Håstad. Tensor rank is NP-complete. *Journal of Algorithms*, 11(4): 644–654, Dec. 1990. doi: 10.1016/0196-6774(90)90014-6.

[71] W. B. Johnson, J. Lindenstrauss, and G. Schechtman. Extensions of lipschitz maps into Banach spaces. *Israel Journal of Mathematics*, 54(2): 129–138, June 1986. doi: 10.1007/BF02764938.

[72] E. Karahan, P. A. Rojas-Lopez, M. L. Bringas-Vega, P. A. Valdes-Hernandez, and P. A. Valdes-Sosa. Tensor Analysis and Fusion of Multimodal Brain Images. *Proceedings of the IEEE*, 103(9):1531–1559, Sept. 2015. doi: 10.1109/JPROC.2015.2455028.

[73] A. R. Karlin. Web Search via Hub Synthesis. In G. Goos, J. Hartmanis, J. van Leeuwen, M. Goemans, K. Jansen, J. D. P. Rolim, and L. Trevisan, editors, *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*, volume 2129, pages 6–6. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. doi: 10.1007/3-540-44666-4_3. Series Title: Lecture Notes in Computer Science.

[74] O. Kaya and B. Uçar. High Performance Parallel Algorithms for the Tucker Decomposition of Sparse Tensors. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 103–112, Philadelphia, PA, Aug. 2016. IEEE. doi: 10.1109/ICPP.2016.19.

[75] O. Kaya and B. Uçar. Parallel Candecomp/Parafac Decomposition of Sparse Tensors Using Dimension Trees. *SIAM Journal on Scientific Computing*, 40(1):C99–C130, Jan. 2018. doi: 10.1137/16M1102744.

[76] S. Keller, M. Dolfi, M. Troyer, and M. Reiher. An Efficient Matrix Product Operator Representation of the Quantum-Chemical Hamiltonian. *The Journal of Chemical Physics*, 143(24):244118, Dec. 2015. doi: 10.1063/1.4939000. arXiv: 1510.02026.

[77] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, Sept. 1999. doi: 10.1145/324133.324140.

[78] T. G. Kolda and B. W. Bader. Tensor Decompositions and Applications. *SIAM Review*, 51(3):455–500, 2009. doi: 10.1137/07070111X.

[79] E. Korkmaz. *Improving the memory and time overhead of low-rank parallel linear sparse direct solvers*. PhD thesis, ED39 Mathématiques et Informatique, Université de Bordeaux, 2022. Supervised by Ramet, Pierre and Faverge, Mathieu.

[80] K. Kormann. Low-rank tensor discretization for high-dimensional problems. Technical report, Max-Planck-Institut für Plasmaphysik, Aug. 2017.

[81] P. M. Kroonenberg and J. de Leeuw. Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika*, 45(1):69–97, Mar. 1980. doi: 10.1007/BF02293599.

[82] N. Lee and A. Cichocki. Fundamental tensor operations for large-scale data analysis using tensor network formats. *Multidimensional Systems and Signal Processing*, 29(3):921–960, July 2018. doi: 10.1007/s11045-017-0481-0.

[83] L. Ma and E. Solomonik. Fast and accurate randomized algorithms for low-rank tensor decompositions. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 24299–24312. Curran Associates, Inc., 2021.

[84] L. Ma and E. Solomonik. Accelerating alternating least squares for tensor decomposition by pairwise perturbation. *Numerical Linear Algebra with Applications*, 29(4), Aug. 2022. doi: 10.1002/nla.2431.

[85] P.-G. Martinsson and J. A. Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 29:403–572, May 2020. doi: 10.1017/S0962492920000021.

[86] P.-G. Martinsson, G. Quintana Ortì, N. Heavner, and R. van de Geijn. Householder QR Factorization With Randomization for Column Pivoting (HQRRP). *SIAM Journal on Scientific Computing*, 39(2):C96–C115, Jan. 2017. doi: 10.1137/16M1081270.

[87] D. A. Matthews. High-Performance Tensor Contraction without Transposition. July 2017, arXiv: 1607.00291.

[88] L. E. McMurchie and E. R. Davidson. One- and two-electron integrals over cartesian gaussian functions. *Journal of Computational Physics*, 26 (2):218–231, Feb. 1978. doi: 10.1016/0021-9991(78)90092-X.

[89] T. Minka. A Comparison of Numerical Optimizers for Logistic Regression. Technical report, Microsoft, Mar. 2003.

[90] R. Minster, A. K. Saibaba, and M. E. Kilmer. Randomized Algorithms for Low-Rank Tensor Decompositions in the Tucker Format. *SIAM Journal on Mathematics of Data Science*, 2(1):189–215, Jan. 2020. doi: 10.1137/19M1261043.

[91] R. L. Minster. *Randomized Algorithms for Tensors and Matrices with Applications*. PhD thesis, North Carolina State University, 2021.

[92] I. Oseledets. DMRG Approach to Fast Linear Algebra in the TT-Format. *Computational Methods in Applied Mathematics*, 11(3):382–393, 2011. doi: 10.2478/cmam-2011-0021.

[93] I. Oseledets and E. Tyrtyshnikov. TT-cross approximation for multidimensional arrays. *Linear Algebra and its Applications*, 432(1):70–88, Jan. 2010. doi: 10.1016/j.laa.2009.07.024.

[94] I. V. Oseledets. Tensor-Train Decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, Jan. 2011. doi: 10.1137/090752286.

[95] I. V. Oseledets, D. V. Savostianov, and E. E. Tyrtyshnikov. Tucker Dimensionality Reduction of Three-Dimensional Arrays in Linear Time. *SIAM Journal on Matrix Analysis and Applications*, 30(3):939–956, Jan. 2008. doi: 10.1137/060655894.

[96] C.-T. Pan and P. T. P. Tang. Bounds on singular values revealed by QR factorizations. *Bit Numerical Mathematics*, 39(4):740–756, 1999. doi: 10.1023/A:1022395308695.

[97] V. Y. Pan, G. Qian, and A.-L. Zheng. Randomized Matrix Computations. Oct. 2012, arXiv: 1210.7476.

[98] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: a probabilistic analysis. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems - PODS '98*, pages 159–168, Seattle, Washington, United States, 1998. ACM Press. doi: 10.1145/275487.275505.

[99] G. Quintana Ortì, X. Sun, and C. H. Bischof. A BLAS-3 Version of the QR Factorization with Column Pivoting. *SIAM Journal on Scientific Computing*, 19(5):1486–1494, Sept. 1998. doi: 10.1137/S1064827595296732.

[100] R. Sands and F. W. Young. Component models for three-way data: An alternating least squares algorithm with optimal scaling features. *Psychometrika*, 45(1):39–67, Mar. 1980. doi: 10.1007/BF02293598.

[101] D. V. Savostyanov and E. E. Tyrtyshnikov. Approximate multiplication of tensor matrices based on the individual filtering of factors. *Computational Mathematics and Mathematical Physics*, 49(10):1662–1677, Oct. 2009. doi: 10.1134/S0965542509100029.

[102] D. V. Savostyanov, E. E. Tyrtyshnikov, and N. L. Zamarashkin. Fast truncation of mode ranks for bilinear tensor operations. *Numerical Linear Algebra with Applications*, 19(1):103–111, Jan. 2012. doi: 10.1002/nla.765.

[103] U. Schollwoeck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1):96–192, Jan. 2011. doi: 10.1016/j.aop.2010.09.012. arXiv: 1008.3477.

[104] M. Sharma and S. Kumar. A Flexible Lossy Depth Video Coding Scheme Based on Low-rank Tensor Modelling and HEVC Intra Prediction for Free Viewpoint Video. Apr. 2021, arXiv: 2104.04678.

[105] P. B. Snajberk. *Direct density matrix renormalization group approaches for strong correlation effects in quantum chemistry.* PhD thesis, Ludwig Maximilians Universität München, 2017.

[106] G. W. Stewart. Rank Degeneracy. *SIAM Journal on Scientific and Statistical Computing*, 5(2):403–413, June 1984. doi: 10.1137/0905030.

[107] S. Szalay, M. Pfeffer, V. Murg, G. Barcza, F. Verstraete, R. Schneider, and O. Legeza. Tensor product methods and entanglement optimization for *ab initio* quantum chemistry. *International Journal of Quantum Chemistry*, 115(19):1342–1391, Oct. 2015. doi: 10.1002/qua.24898.

[108] P. Tarits, S. Hautot, J. D'Eu, P. Wawrzyniak, F. Bretaudeau, S. Védrine, M. Darnet, and A. Stopin. On the Importance of Nearshore Extension of MT Arrays for Geothermal Exploration of Coastal Geothermal Systems. In *EAGE GET 2022*, pages 1–5, The Hague, Netherlands,, 2022. European Association of Geoscientists & Engineers. doi: 10.3997/2214-4609.202221077.

[109] R. C. Thompson. Principal submatrices IX: Interlacing inequalities for singular values of submatrices. *Linear Algebra and its Applications*, 5(1): 1–12, 1972.

[110] J. A. Tropp. Improved analysis of the subsampled randomized Hadamard transform. *Advances in Adaptive Data Analysis*, 03(01n02):115–126, Apr. 2011. doi: 10.1142/S1793536911000787.

[111] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, Sept. 1966. doi: 10.1007/BF02289464.

[112] E. Tyrtyshnikov. Incomplete Cross Approximation in the Mosaic-Skeleton Method. *Computing*, 64(4):367–380, June 2000. doi: 10.1007/s006070070031.

[113] J. Vandewalle and B. De Moor. On the Use of the Singular Value Decomposition in Identification and Signal Processing. In G. H. Golub and P. Van Dooren, editors, *Numerical Linear Algebra, Digital Signal Processing and Parallel Algorithms*, pages 321–360. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991. doi: 10.1007/978-3-642-75536-1_16.

[114] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen. A New Truncation Strategy for the Higher-Order Singular Value Decomposition. *SIAM Journal on Scientific Computing*, 34(2):A1027–A1052, Jan. 2012. doi: 10.1137/110836067.

[115] S. R. White. Density matrix formulation for quantum renormalization groups. *Phys. Rev. Lett.*, 69:2863–2866, Nov 1992. doi: 10.1103/PhysRevLett.69.2863.

[116] G. M. Wing and J. D. Zahrt. *A primer on integral equations of the first kind: the problem of deconvolution and unfolding.* Society for Industrial and Applied Mathematics, Philadelphia, 1991.

[117] D. P. Woodruff. Sketching as a Tool for Numerical Linear Algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1-2):1–157, 2014. doi: 10.1561/0400000060. arXiv: 1411.4357.

[118] J. Xiao, M. Gu, and J. Langou. Fast Parallel Randomized QR with Column Pivoting Algorithms for Reliable Low-rank Matrix Approximations. *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*, pages 233–242, Dec. 2017. doi: 10.1109/HiPC.2017.00035. arXiv: 1804.05138.

[119] X. Xing, H. Huang, and E. Chow. A linear scaling hierarchical block low-rank representation of the electron repulsion integral tensor. *The Journal of Chemical Physics*, 153(8):084119, Aug. 2020. doi: 10.1063/5.0010732.

[120] Y. Yang, M. Pilanci, and M. J. Wainwright. Randomized sketches for kernels: Fast and optimal non-parametric regression. Jan. 2015, arXiv: 1501.06195.

[121] H.-F. Yu, C.-J. Hsieh, S. Si, and I. S. Dhillon. Parallel matrix factorization for recommender systems. *Knowledge and Information Systems*, 41(3): 793–819, Dec. 2014. doi: 10.1007/s10115-013-0682-2.

[122] R. Yu, D. Cheng, and Y. Liu. Accelerated Online Low Rank Tensor Learning for Multivariate Spatiotemporal Streams. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 238–247, Lille, France, July 2015. PMLR.

[123] G. Zhou, A. Cichocki, and S. Xie. Decomposition of Big Tensors With Low Multilinear Rank. 2014, arXiv: 1412.1885.