Higher-order QR with tournament pivoting for tensor compression

David Frenkiel, Laura Grigori, Matthias Beaupère

▶ **To cite this version:**

David Frenkiel, Laura Grigori, Matthias Beaupère. Higher-order QR with tournament pivoting for tensor compression. 2020. hal-03079236v2

HAL Id: hal-03079236

https://inria.hal.science/hal-03079236v2

Preprint submitted on 17 Dec 2020 (v2), last revised 2 Feb 2022 (v5)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Higher-Order QR with Tournament Pivoting for Tensor Compression

David Frenkiel*, Laura Grigori†, and Matthias Beaupère‡

2020/12/17

Abstract

We present in this paper a parallel algorithm that generates a low-rank approximation of a distributed tensor using QR decomposition with tournament pivoting (QRTP). The algorithm generates factor matrices for a Tucker decomposition by applying QRTP to the unfolding matrices of a tensor distributed block-wise (by sub-tensor) on a set of processors. For each unfolding mode the algorithm logically reorganizes (unfolds) the processors so that the associated unfolding matrix has a suitable logical distribution. We also establish error bounds between a tensor and the compressed version of the tensor generated by the algorithm.

1 Introduction

In the fields of scientific computing and big data one often encounters multi-dimensional arrays called *tensors*. In physics, for example, tensors appear in the representation of the states of a quantum mechanical system, while in data analytics tensors are used to represent multi-dimensional data sets such as crime statistics or customer product reviews. For problems of real interest these tensors are generally extremely large. A quantum mechanical system consisting of just 100 electron spins, for example, is described by a tensor containing a rather astronomical 2^{100} elements. Given the size of such tensors, their storage and manipulation necessitates some form of data compression.

In the particular case of a matrix (i.e., a 2-dimensional tensor) compression is generally realized using a technique called *low-rank approximation*. A common way to generate a low-rank approximation of a matrix is by performing a truncated *singular value decomposition (SVD)*. However, while SVD provides the best low-rank approximation of a matrix (see, e.g., Eckart and Young [8]), it is also computationally expensive and difficult to parallelize. Beyond the computational complexity of any specific algorithm, high-performance computing faces the increasing challenge posed by communication overhead. On massively parallel machines the communication between processors required by parallel algorithms is steadily replacing floating-point computation as the primary performance bottleneck in scientific computation (see, e.g., Grigori [9]).

An alternative way to generate a low-rank approximation of a matrix is provided by a type of matrix decomposition called *rank-revealing QR (RRQR)*. To address communication avoidance in the parallel execution of RRQR, Demmel, Grigori, Gu and Xiang [7] propose an algorithm called *communication-avoiding rank-revealing QR (CARRQR)*. CARRQR distributes the columns of a matrix across a set of processors and utilizes tournament

*ALPINES, Inria Paris, France (david.frenkiel@inria.fr)
†ALPINES, Inria Paris, France (laura.grigori@inria.fr)
‡ALPINES, Inria Paris, France (matthias.beaupere@inria.fr)

pivoting to identify a set of *spectrum-revealing* columns while minimizing inter-process communication, where the spectrum-revealing columns are defined as the columns associated with the largest singular values of the matrix. The algorithm presented in this paper specifically uses a variant of CARRQR called *QR with tournament pivoting (QRTP)*. As described in Beaupère and Grigori [4], instead of a column of processors, QRTP exploits a 2D grid of processors to identify a set of spectrum-revealing columns of a matrix,

For a general tensor (i.e., of dimension greater than 2), there exists a well-known low-rank approximation method called *higher-order singular value decomposition (HOSVD)*. HOSVD, as its name suggests, utilizes truncated SVD decompositions to extract a low-rank approximation from a tensor and, consequently, suffers from the same performance limitations. We thus present in this paper a parallel algorithm that, while structurally similar to HOSVD, utilizes QRTP (instead of SVD) to extract a low-rank approximation of a tensor. The algorithm is called *Higher-Order QR with Tournament Pivoting (HO-QRTP)*. As truncated SVD provides the best low-rank approximation of a matrix, one can expect that HOSVD provides better compression results for a tensor than does HO-QRTP. However, as described in [4], QRTP is computationally less expensive than SVD and requires less communication between processors. HO-QRTP is thus a low-cost alternative to HOSVD that is useful for directly computing low-rank approximations of large tensors, or as part of a more general iterative low-rank approximation algorithm. It should be noted however that, while HO-QRTP is technically applicable to a tensor of any order, there are preferable options for tensors of very high order, such as the tensor-train decomposition.

HO-QRTP differentiates itself from HOSVD in two ways. Firstly, as noted above, HO-QRTP utilizes QRTP instead of SVD to efficiently compute low-rank approximations with minimal communication. Secondly, HO-QRTP employs a technique called *partitioned unfolding* to distribute tensor data across a set of processors. Partitioned unfolding facilitates the application of QRTP to the full set of unfolding matrices without introducing additional communication overhead. Austin, Ballard and Kolda [2] propose a parallel HOSVD implementation that employs a data distribution strategy they call *block distribution*. Partitioned unfolding expands upon block distribution by defining how tensor blocks are logically rearranged and unfolded to form the full set of distributed unfolding matrices. In another related work, Shah, Wieser and Bry [16] present an HOSVD algorithm that leverages a parallel SVD implementation called *Hestenes' SVD*. While this approach is more direct than ours, it does not explicitly address communication avoidance. The use of QRTP in HO-QRTP delivers a lower communication cost in addition to a reduced computational cost. A notable alternative parallelization strategy to that employed by HO-QRTP is described in Zhou, Cichocki and Xie [17], where they use randomized methods to reduce the complexity of their parallel HOSVD variant.

For a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, HO-QRTP yields a compressed tensor $\widetilde{\mathcal{A}}$ that satisfies the following error bound in the Frobenius norm:

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

where

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

and where $f > 1$, k_i is the truncation rank along the i^{th} mode (dimension), P is the total number of processors, and $P_r^{(i)}$ is the height of the processor grid related to the i^{th} unfolding mode. See Theorem 2.1 for the definition of $\mathcal{A}_{\text{best}}$, and Section 4.1 for further details

concerning the error bound. We also note that our implementation of HO-QRTP achieves a strong-scaling performance speedup of 11 on 512 processors (relative to 8 processors). This is further discussed in Section 5.2

The rest of this paper is organized as follows. In Section 2 we provide some background on low-rank matrix approximations, QRTP, Tucker decompositions, and HOSVD. In Section 3 we explain how partitioned unfolding is used to distribute tensor data across a set of processors, and we prove that partitioned unfolding does not perturb compression results. In Section 4 we present a full description of the HO-QRTP algorithm and provide a proof of the error bound mentioned above in (1). We also compute the computational and communication cost of HO-QRTP. Lastly, in Section 5 we present numerical experiments that depict to what degree HO-QRTP preserves singular values, and how our HO-QRTP implementation scales on varying numbers of processors.

2 Background

2.1 Low-Rank Matrix Approximation

QR factorization with column pivoting can be used to compute a low-rank approximation. Consider the decomposition of a matrix $A \in \mathbb{R}^{m\times n}$ as

$$A\Pi = QR = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}, \tag{2}$$

where $\Pi \in \mathbb{R}^{n\times n}$ is a permutation matrix, $Q \in \mathbb{R}^{m\times m}$ is orthogonal, $Q_1 \in \mathbb{R}^{m\times k}$, $R \in \mathbb{R}^{m\times n}$, and $R_{11} \in \mathbb{R}^{k\times k}$ is upper triangular. A low-rank approximation $\widetilde{A}_k$ is obtained by projecting A onto the vector space spanned by the k leading columns of ΠA. We thus have the approximation

$$\widetilde{A}_k = Q_1 Q_1^\intercal A = Q_1 \begin{bmatrix} R_{11} & R_{12} \end{bmatrix}. \tag{3}$$

In the original algorithm introduced by Businger and Golub [5], the permutation matrix Π is determined by selecting the column with the largest norm at each step of the QR factorization. We refer to this algorithm as QRCP.

As shown in Gu and Eisenstat [10], given $f > 1$, a matrix $A \in \mathbb{R}^{m\times n}$ where $m \geq n$, and a rank k, there exists a permutation Π such that

$$1 \leq \frac{\sigma_i(A)}{\sigma_i(R_{11})} \leq \sqrt{1 + kf^2(n-k)}, \quad 1 \leq \frac{\sigma_j(R_{22})}{\sigma_{k+j}(A)} \leq \sqrt{1 + kf^2(n-k)}, \tag{4}$$

for any $j \in \{1, \ldots, n-k\}$ and $i \in \{1, \ldots, k\}$. Such a factorization is referred to as strong rank-revealing QR (RRQR). Using the terminology in [6], this factorization is both spectrum-preserving and a kernel approximation of A, where the spectrum-preserving property is satisfied for the singular values of R_{11} instead of $\widetilde{A}_k$ as in the original definition. We also note that the approximation $\widetilde{A}_k$ satisfies

$$\left\|A - \widetilde{A}_k\right\|_F^2 = \left\|(I - Q_1 Q_1^\intercal)A\right\|_F^2 = \left\|R_{22}\right\|_F^2 = \sum_j^r \sigma_j^2(R_{22}),$$

where r is the rank of the matrix R_{22}.

QRTP is based on tournament pivoting [4], and provides a method to select k spectrum-revealing columns from a matrix $A \in \mathbb{R}^{m\times n}$ that is distributed by block both horizontally

and vertically on a $P_r \times P_c$ processor grid, where $P = P_r P_c$ is the total number of processors. The term *spectrum-revealing* indicates that the selected columns are those associated with the k largest singular values of the matrix. QRTP is an extension of CARRQR, which is an algorithm that selects k columns from a matrix partitioned horizontally into blocks of columns. Tournament pivoting, as used by QRTP and CARRQR, is the key mechanism behind the extraction of rank-revealing columns. The tournament pivoting algorithm first selects k columns from each block of A by using strong RRQR. These columns are then combined and the selection process continues via a reduction operation during which k columns are chosen via strong RRQR from the $2k$ columns extracted during the previous reduction step. This binary reduction process continues until k columns are finally selected on the root processor. We focus here on the specific case where tournament pivoting first considers each block of columns of A and selects k columns from the k sub-columns selected in each block, and then selects the k final columns from these P_c sets of k columns. The QRTP decomposition as given in (2) satisfies the following error bounds:

$$1 \leq \frac{\sigma_i(A)}{\sigma_i(R_{11})} \leq \sqrt{1 + k\widetilde{f}^2(n-k)}, \quad 1 \leq \frac{\sigma_j(R_{22})}{\sigma_{j+k}(A)} \leq \sqrt{1 + k\widetilde{f}^2(n-k)}, \tag{5}$$

where $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, n-k\}$, and

$$\widetilde{f} = \sqrt{P} P_r k^{\log_2(P)} f^{\log_2(P)+1}.$$

QRTP thus generates a kernel approximation of A that is also spectrum preserving. As indicated in [4], QRTP requires $(\log_2 P_c + \log_2 P_r)(1 + \log_2 P_r)$ messages. Another feature of QRTP is that it allows one to modify the reduction tree used during the tournament pivoting operation. As we discuss later in the paper, this facet of QRTP allows our algorithm (HO-QRTP) to efficiently extract a low-rank approximation from a tensor that is distributed by sub-tensors on a set of processors.

2.2 Tucker Decomposition of a Tensor

The algorithm presented in this paper extracts a low-rank approximation of a tensor by generating its *Tucker decomposition*. A Tucker decomposition is essentially the generalization of a low-rank matrix approximation to a tensor. Specifically, given a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and a set of ranks $\{k_i\}_{1 \leq i \leq d}$, a Tucker decomposition of $\mathcal{A}$ consists of a *core tensor* $\mathcal{C} \in \mathbb{R}^{k_1 \times \cdots \times k_d}$ and a set of *factor matrices* $\{U_i\}_{1 \leq i \leq d}$, where

$$\mathcal{C} = \mathcal{A} \times_1 U_1^\intercal \times_2 \cdots \times_d U_d^\intercal, \tag{6}$$

and where each $U_i \in \mathbb{R}^{n_i \times k_i}$ contains k_i orthonormal columns. Figure 1 depicts a 4×4×4 tensor and its rank 2×2×2 Tucker decomposition where the 3 factor matrices are computed using QRCP from the Julia programming language [13].

The operation $\mathcal{A} \times_j B$ is the *n-mode product*, where $1 \leq j \leq d$. Given a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and a matrix $B \in \mathbb{R}^{m \times n_i}$, the n-mode product is a tensor $\mathcal{C} \in \mathbb{R}^{n_1 \times \cdots \times n_{j-1} \times m \times n_{j+1} \cdots \times n_d}$ where the elements of $\mathcal{C}$ are given by

$$\mathcal{C}_{s_1 \cdots s_d} = (\mathcal{A} \times_j B)_{s_1 \cdots s_d} = \sum_{t=1}^{n_j} \mathcal{A}_{s_1 \cdots s_{j-1} t s_{j-1} \cdots s_d} B_{s_j t}.$$

The total size of the Tucker decomposition is $\prod_{i=1}^d k_i + \sum_{i=1}^d n_i k_i$. Assuming the ranks are sufficiently small relative to the dimensions of the tensor, we see clearly that the Tucker

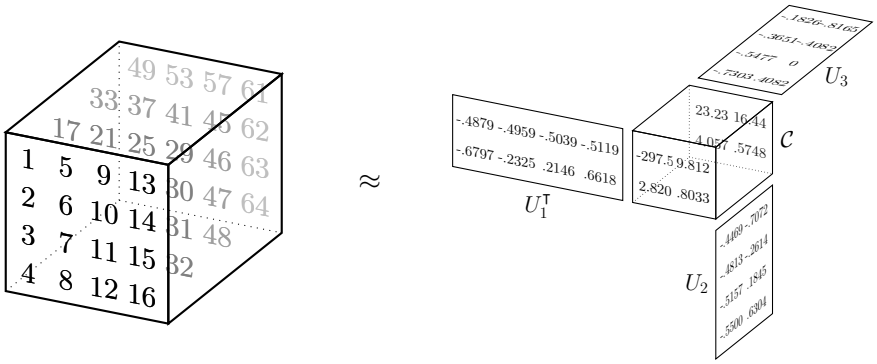


Figure 1: A 4×4×4 tensor and its rank 2×2×2 Tucker decomposition where the 3 factor matrices are computed using QRCP.

decomposition is in fact a compressed version of the tensor. In Section 4.1 we compute error bounds between a tensor $\mathcal{A}$ and its compressed version $\widetilde{\mathcal{A}}$, where the compressed version of the tensor is recovered from the Tucker decomposition by

$$\begin{aligned}\widetilde{\mathcal{A}} &= \mathcal{C} \times_1 U_1 \times_2 \cdots \times_d U_d = (\mathcal{A} \times_1 U_1^\intercal \times_2 \cdots \times_d U_d^\intercal) \times_1 U_1 \times_2 \cdots \times_d U_d \\ &= \mathcal{A} \times_1 U_1 U_1^\intercal \times_2 \cdots \times_d U_d U_d^\intercal.\end{aligned}$$

From a high-level perspective, algorithms for computing Tucker decompositions – such as HOSVD and the HO-QRTP algorithm described in this paper – perform two principal tasks. These algorithms first extract a set of factor matrices, and then they employ an n-mode tensor-matrix product between the factor matrices and the original tensor to generate the core tensor. In order to extract factor matrices from a tensor we must first convert the tensor itself into a set of matrices. Then, from each of these matrices, we extract a factor matrix. This matrix representation of a tensor, or *matricization*, is called an *unfolding*.

More precisely, an unfolding of a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ is a set of matrices $\{A_i\}_{1 \le i \le d}$, where $A_i \in \mathbb{R}^{n_i \times N_i}$ and $N_i = n_1 \cdots n_{i-1} n_{i+1} \cdots n_d$. Each matrix of the unfolding contains all the elements of the tensor and can be viewed as the flattening of the tensor to 2 dimensions along a specific mode. As described in [14], the columns of the n-mode unfolding matrix are the mode-n fibers extracted from the tensor $\mathcal{A}$. An element of the i-mode unfolding matrix A_i is thus mapped to an element of the tensor $\mathcal{A}$ by the relation

$$A_i(k_i, c) = \mathcal{A}(k_1, \cdots, k_{i-1}, k_i, k_{i+1}, \cdots k_d), \tag{7}$$

where the column index c is given by

$$c = 1 + \sum_{\substack{s=1 \\ s \neq i}}^{d} \Big[(k_s - 1) \prod_{\substack{t=1 \\ t \neq i}}^{s-1} n_t \Big].$$

We also have the following relation between the n-mode (tensor-matrix) product and the matrix-matrix product of an unfolding matrix with an arbitrary matrix. Let $\mathcal{C} = \mathcal{A} \times_j B$. Then $C_i = B A_i$, where A_i and C_i are, respectively, the i-mode unfolding matrices of the tensors $\mathcal{A}$ and $\mathcal{C}$. For further details about tensor properties and operations see for example [14].

The general structure of an algorithm to extract a Tucker decomposition (e.g. HOSVD) is presented in Algorithm 1.

Algorithm 1 Compress a tensor (generate a Tucker decomposition)
Input: A tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and a set of ranks $\{k_i\}_{1 \le i \le d}$
Output: A Tucker decomposition (a core tensor $\mathcal{C}$ and factor matrices $\{U_i\}_{1 \le i \le d}$ such that $\widetilde{\mathcal{A}} = \mathcal{C} \times_1 U_1 U_1^\intercal \times_2 \cdots \times_d U_d U_d^\intercal$)

1: Extract the unfolding matrices $\{A_i\}_{1 \le i \le d}$ from the tensor $\mathcal{A}$
2: Extract a factor matrix $U_i \in \mathbb{R}^{n_i \times k_i}$ from each unfolding matrix A_i. In the case of HOSVD, for example, U_i contains the first k_i left singular vectors of the SVD decomposition of the unfolding matrix A_i.
3: Generate the core tensor $(\mathcal{C} = \mathcal{A} \times_1 U_1^\intercal \times_2 \cdots \times_d U_d^\intercal)$

In the specific case of HOSVD, we have the following result (Theorem 10.3 of [11]):

Theorem 2.1. *(HOSVD quasi-optimal) Let $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ be a tensor, $\widetilde{\mathcal{A}}$ be its rank-$(k_1, \ldots, k_d)$ HOSVD decomposition, and $\mathcal{A}_{best}$ be the best rank-$(k_1, \ldots, k_d)$ approximation of $\mathcal{A}$. Then*

$$\|\mathcal{A} - \widetilde{\mathcal{A}}\|_F \le \sqrt{d}\, \|\mathcal{A} - \mathcal{A}_{best}\|_F ,$$

where, more precisely,

$$\mathcal{A}_{best} = \operatorname*{argmin}_{\mathcal{B} \in \mathcal{T}_k} \|\mathcal{A} - \mathcal{B}\|_F,$$

and where $\mathcal{T}_k \subset \mathbb{R}^{n_1 \times \cdots \times n_d}$ is the space of all tensors that possess a $(k_1, \ldots, k_d)$-tensor subspace representation (as described in Definition 8.1 of [11]).

3 Partitioned Unfolding

In order to efficiently exploit QRTP, our algorithm utilizes a technique called *partitioned unfolding* to distribute tensor data across a set of processors. Given a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ we first organize the set of processors into a tensor with the same number of dimensions as $\mathcal{A}$. The tensor $\mathcal{A}$ is then accordingly split into blocks and distributed amongst the processors. For example, for a 3-dimensional tensor and a set of 16 processors, we could use a processor tensor $\mathcal{P} \in \mathbb{N}^{2 \times 2 \times 4}$, where each entry of $\mathcal{P}$ contains the ID of a processor. In this case the tensor $\mathcal{A}$ is divided into blocks of size $\frac{n_1}{2} \times \frac{n_2}{2} \times \frac{n_3}{4}$.

The organization of the processor tensor is essentially arbitrary. The only requirement is that it has the correct number of dimensions (and processors). For example, we could also use $\mathcal{P} \in \mathbb{N}^{1 \times 4 \times 4}$ or $\mathcal{P} \in \mathbb{N}^{16 \times 1 \times 1}$. However, in order to optimally distribute data across the processors, it is desirable to define a $\mathcal{P}$ that splits $\mathcal{A}$ along its largest dimensions. Conceptually, $\mathcal{P}$ should have the same shape as $\mathcal{A}$. For example, a "long" tensor $\mathcal{A} \in \mathbb{N}^{64 \times 4 \times 4}$ is best distributed with $\mathcal{P} \in \mathbb{N}^{16 \times 1 \times 1}$.

In what follows we suppose for simplicity that the number of processors is such that we can always create an equidimensional processor tensor. For example, for 8 processors we choose to use a processor tensor of size $2 \times 2 \times 2$. Thus, given P processors, we suppose that we can organize the set of processors into the d-dimensional processor tensor $\mathcal{P} \in \mathbb{R}^{\sqrt[d]{P} \times \cdots \times \sqrt[d]{P}}$. In other words, we assume that there exists an integer l such that $l^d = P$.

Once the partitioning has been established and the tensor blocks have been distributed amongst the processors, we extract each (distributed) unfolding matrix from the tensor by

performing a 2-stage unfolding. Given an unfolding mode $i \in \{1, \ldots, d\}$, we extract the i-mode unfolding of the processor (stage 1), and each processor extracts the i-mode unfolding of its tensor block (stage 2). The resulting processor grid and distributed matrix together comprise the *partitioned i-mode unfolding* of the tensor $\mathcal{A}$. We denote the partitioned i-mode unfolding as A_{i^2}.

Partitioned unfolding plays an important role in our algorithm as it allows us to avoid communication. The sub-tensors are distributed once amongst the processors at the beginning of the algorithm, and then each factor matrix is generated in parallel using the same data distribution.

3.1 Partitioned Unfolding Example

To better describe the structure of a partitioned unfolding we present here an example of the distribution of 3-dimensional tensor $\mathcal{A} \in \mathbb{R}^{4\times4\times4}$ on a set of 8 processors followed by a 1^2-mode partitioned unfolding of $\mathcal{A}$. The example tensor $\mathcal{A}$ contains the integers from 1 to 64 and is depicted in Figure 2.

$\mathcal{A} =$

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

17	21	25	29
18	22	26	30
19	23	27	31
20	24	28	32

33	37	41	45
34	38	42	46
35	39	43	47
36	40	44	48

49	53	57	61
50	54	58	62
51	55	59	63
52	56	60	64

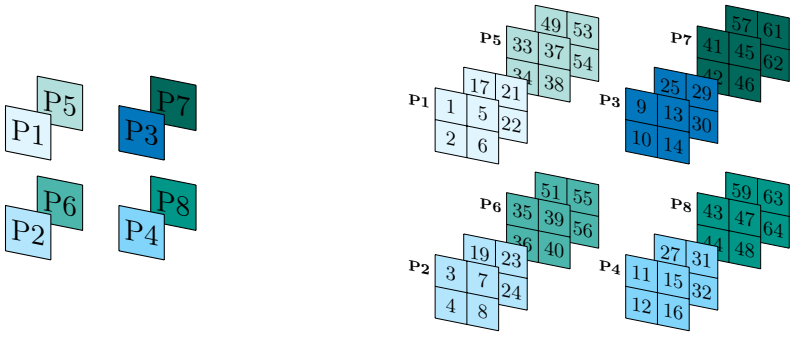
Figure 2: 4×4×4 example tensor.

As depicted in Figure 3, partitioned unfolding of the tensor $\mathcal{A}$ consists of the following 3 steps:

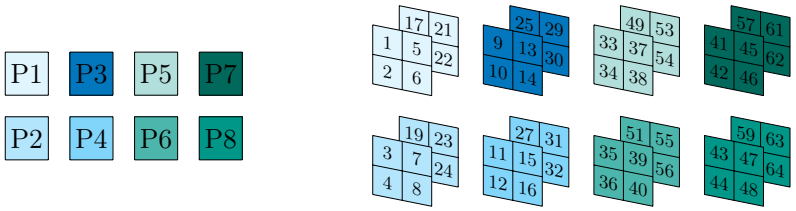
Step 1 (distribution of $\mathcal{A}$) We first organize the 8 processors into a 2×2×2 tensor $\mathcal{P}$, and then we cut $\mathcal{A}$ into blocks (sub-tensors) and distribute the blocks amongst the processors.

Step 2 (processor tensor unfolding) We *unfold* the 8 processors. That is, we perform a 1-mode unfolding of the 2×2×2 tensor formed by the 8 processors.

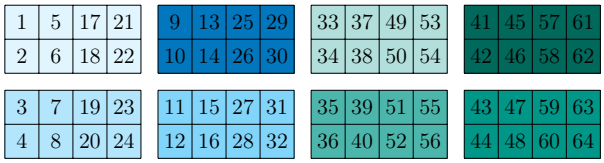
Step 3 (sub-tensor unfolding) Finally, each processor extracts the 1-mode unfolding from its sub-tensor. This results in the set of unfolding sub-matrices distributed amongst the processors.



Step 1: Organize processors into 2×2×2 tensor and distribute sub-tensors.



Step 2: Unfold processor tensor.



Step 3: Unfold sub-tensor on each processor.

Figure 3: Partitioned unfolding applied to a 4×4×4 tensor.

After executing these 3 steps we arrive at the following partitioned unfolding matrix A_{1^2} distributed by block on the grid of 8 processors:

$$A_{1^2} = \begin{bmatrix} 1 & 5 & 17 & 21 & 9 & 13 & 25 & 29 & 33 & 37 & 49 & 53 & 41 & 45 & 57 & 61 \\ 2 & 6 & 18 & 22 & 10 & 14 & 26 & 30 & 34 & 38 & 50 & 54 & 42 & 46 & 58 & 62 \\ 3 & 7 & 19 & 23 & 11 & 15 & 27 & 31 & 35 & 39 & 51 & 55 & 43 & 47 & 59 & 63 \\ 4 & 8 & 20 & 24 & 12 & 16 & 28 & 32 & 36 & 40 & 52 & 56 & 44 & 48 & 60 & 64 \end{bmatrix} \tag{8}$$

3.2 Partitioned Unfolding vs Direct Unfolding

Let's now compare the matrices generated by a partitioned unfolding and a direct unfolding of the tensor $\mathcal{A}$. We already computed the 1^2-mode partitioned unfolding A_{1^2} (8), and for the direct 1-mode unfolding A_1 of $\mathcal{A}$ we have the following matrix:

$$A_1 = \begin{bmatrix} 1 & 5 & 9 & 13 & 17 & 21 & 25 & 29 & 33 & 37 & 41 & 45 & 49 & 53 & 57 & 61 \\ 2 & 6 & 10 & 14 & 18 & 22 & 26 & 30 & 34 & 38 & 42 & 46 & 50 & 54 & 58 & 62 \\ 3 & 7 & 11 & 15 & 19 & 23 & 27 & 31 & 35 & 39 & 43 & 47 & 51 & 55 & 59 & 63 \\ 4 & 8 & 12 & 16 & 20 & 24 & 28 & 32 & 36 & 40 & 44 & 48 & 52 & 56 & 60 & 64 \end{bmatrix}$$

Comparing these two matrices we immediately note that the matrix A_{1^2} is just a column permutation of the matrix A_1:

$$A_{1^2} = A_1\Pi_1,$$

where Π_1 is a permutation matrix. In the following theorem we prove this relation between the i-mode unfolding and the i^2 mode partitioned unfolding.

Theorem 3.1. *(Partitioned unfolding) For each $i \in \{1, \ldots, d\}$, the i^2-mode partitioned unfolding A_{i^2} 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, $A_{i^2} = A_i\Pi_i \ \forall i \in \{1, \ldots, d\}$, where each Π_i is a permutation matrix, A_{i^2} is the i^2-mode partitioned unfolding, and A_i is the ordinary i-mode unfolding of $\mathcal{A}$.*

Proof. We'd like to show that for any $i \in \{1, \ldots, d\}$ there exists a permutation matrix Π_i such that $A_{i^2} = A_i\Pi_i$. In order to prove the result we only need to show that the 2 matrices (A_{i^2} and A_i) contain the same set of columns. The precise order of the columns (and thus Π_i) aren't explicitly required.

We need to consider the unfoldings of 3 different d-dimensional tensors: 1) the unfolding of the $n_1 \times \cdots \times n_d$ tensor $\mathcal{A}$; 2) the unfolding of the $p_1 \times \cdots \times p_d$ processor tensor $\mathcal{P}$; and 3) the unfolding of the $b_1 \times \cdots \times b_d$ sub-tensor $\mathcal{S}$. We also suppose that the dimensions divide equally. In other words, $n_i = p_i b_i$ for all $i \in \{1, \ldots, d\}$.

For the i-mode unfolding A_i of the tensor $\mathcal{A}$, each element of $\mathcal{A}$ is mapped to an element of A_i by

$$\mathcal{A}(k_1^{\mathcal{A}}, \cdots, k_{i-1}^{\mathcal{A}}, k_i^{\mathcal{A}}, k_{i+1}^{\mathcal{A}}, \cdots k_d^{\mathcal{A}}) = A_i(k_i^{\mathcal{A}}, c^{\mathcal{A}}),$$

where the column index $c^{\mathcal{A}}$ of the element is given by

$$c^{\mathcal{A}} = 1 + \sum_{\substack{s=1 \\ s \neq i}}^{d} \left[(k_s^{\mathcal{A}} - 1) \prod_{\substack{t=1 \\ t \neq i}}^{s-1} n_t \right]. \tag{9}$$

We thus have similar expressions for the elements of the unfolding matrices of the tensors $\mathcal{P}$ and $\mathcal{S}$, where $c^{\mathcal{P}}$ is a function of $k_s^{\mathcal{P}}$ and $c^{\mathcal{S}}$ is a function of $k_s^{\mathcal{S}}$.

We can map the coordinates of an element of $\mathcal{A}$ to a processor in $\mathcal{P}$ by

$$k_j^{\mathcal{P}} = \left\lceil \frac{k_j^{\mathcal{A}}}{b_j} \right\rceil.$$

And similarly, the (relative) coordinates of an element of $\mathcal{A}$ in a sub-tensor $\mathcal{S}$ are given by

$$k_j^{\mathcal{S}} = k_j^{\mathcal{A}} - \left\lfloor \frac{k_j^{\mathcal{A}} - 1}{b_j} \right\rfloor b_j.$$

Now consider the partitioned unfolding A_{i^2} of $\mathcal{A}$. We can relate the column index c^{pu} of an element in A_{i^2} to the column indices of the processor tensor and a sub-tensor by

$$c^{pu} = (c^{\mathcal{P}} - 1) \prod_{\substack{t=1 \\ t \neq i}}^{d} b_t + c^{\mathcal{S}}.$$

Using several of the expressions above we thus have

$$c^{pu} = 1 + \prod_{\substack{t=1 \\ t\neq i}}^{d} b_t \sum_{\substack{s=1 \\ s\neq i}}^{d} \left\lfloor \frac{k_s^{\mathcal{A}} - 1}{b_s} \right\rfloor \prod_{\substack{t=1 \\ t\neq i}}^{s-1} p_t + \sum_{\substack{s=1 \\ s\neq i}}^{d} \left([k_s^{\mathcal{A}} - 1] \bmod b_s \right) \prod_{\substack{t=1 \\ t\neq i}}^{s-1} b_t.$$

Finally, for the row r^{pu} of an element in the i^2-mode partitioned unfolding, we have the (much simpler) result:

$$r^{pu} = k_i^{\mathcal{A}}.$$

This shows that the row index of an element in the ordinary unfolding is identical to the row index in the partitioned unfolding. We thus have a mapping between the elements of the tensor $\mathcal{A}$ and the two unfoldings:

$$\mathcal{A}(k_1^{\mathcal{A}}, \cdots, k_{i-1}^{\mathcal{A}}, k_i^{\mathcal{A}}, k_{i+1}^{\mathcal{A}}, \cdots i_d) = A_i(k_i^{\mathcal{A}}, c^{\mathcal{A}}) = A_{i^2}(k_i^{\mathcal{A}}, c^{pu}). \tag{10}$$

We've also established that neither $c^{\mathcal{A}}$ nor c^{pu} depends on $k_i^{\mathcal{A}}$. As $k_i^{\mathcal{A}}$ goes from 1 to n_i in (10), both $c^{\mathcal{A}}$ and c^{pu} remain fixed. This means that each column of A_i is also contained in A_{i^2}, and thus that A_i and A_{i^2} are comprised of the same set of columns. We've thus proved that there exists a permutation matrix Π_i such that $A_{i^2} = A_i\Pi_i$. □

Since a partitioned unfolding matrix is just a permutation by column of a normal unfolding matrix ($A_{i^2} = A_i\Pi_i$), we have the following QR decomposition,

$$A_{i^2} = A_i\Pi_i = QR\Pi^\intercal\Pi_i.$$

With $\widetilde{\Pi} = \Pi_i^\intercal\Pi$ we have $A_{i^2}\widetilde{\Pi} = QR$, and thus $A_{i^2}\widetilde{\Pi}$ possesses the same QR decomposition as $A_i\Pi_i$. This means that, when the same strategy is used for column selection, the Tucker decomposition generated using the QR factorizations with column pivoting of the partitioned unfolding matrices A_{i^2} is identical to the standard Tucker decomposition generated using the QR factorizations with column pivoting of the normal unfolding matrices A_i. Similarly, for the SVD decomposition of a partitioned unfolding we have

$$A_{i^2} = A_i\Pi_i = U\Sigma V^\intercal\Pi_i = U\Sigma\widetilde{V}^\intercal,$$

where $\widetilde{V}^\intercal = V^\intercal\Pi_i$. The partitioned unfolding A_{i^2} thus possesses exactly the same singular values as the normal unfolding A_i.

4 Algorithm HO-QRTP: Low-Rank Approximation of a Tensor with Partitioned Unfolding and QRTP

To compute a low-rank approximation of a tensor, we propose a parallel algorithm modeled after HOSVD that utilizes partitioned unfolding and QR with tournament pivoting (QRTP).

In order to obtain reasonable error bounds (Section 4.1) we selectively use the transpose of the unfolding matrices. This guarantees that the QRTP operation is always performed on a tall matrix. In other words, if the width of an unfolding matrix is greater than its height – as is often the case with unfoldings – we run QRTP on the transpose of the unfolding matrix. This is preferable because, as presented in Beaupère and Grigori [4] (and Gu

and Eisenstat [10]), error bounds for QRTP (Equation (5)) vary directly with the matrix width. For example, the width of an unfolding matrix of an equidimensional tensor is n^{d-1}, which becomes astronomical for a high-order tensor.

Let $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ be a tensor and $A \in \mathbb{R}^{n_i \times N_i}$ be its i-mode partitioned unfolding ($A \equiv A_{i^2}$), where $N_i = n_1 \cdots n_{i-1} n_{i+1} \cdots n_d$. As noted in the previous paragraph HO-QRTP extracts each factor matrix from $\mathcal{A}$ according to the size of the associated unfolding matrix (tall or wide):

Case 1 : Tall unfolding matrix If the unfolding matrix A is tall ($n_i \geqslant N_i$) we simply run QRTP directly on the unfolding:

$$A\Pi = QR. \tag{11}$$

In this case we use $U_i = Q[:, 1{:}k_i]$ for the factor matrix. The factor matrix $U_i \in \mathbb{R}^{n_i \times k_i}$ thus contains the first k_i (orthonormal) columns of Q. $U_i U_i^\intercal$ is thus an orthogonal projection onto the k_i spectrum-revealing columns of the unfolding matrix A.

Case 2 : Wide unfolding matrix If A is wide ($n_i < N_i$) we run QRTP on the transpose of A, and then use an SVD decomposition to extract a factor matrix. From QRTP we get the decomposition of the transpose of A:

$$A^\intercal \Pi = QR = Q \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}. \tag{12}$$

We can thus represent A as

$$\begin{aligned} A &= \Pi \begin{bmatrix} R_{11}^\intercal & 0 \\ R_{12}^\intercal & R_{22}^\intercal \end{bmatrix} Q^\intercal = [\Pi_1 \;\; \Pi_2] \begin{bmatrix} R_{11}^\intercal & 0 \\ R_{12}^\intercal & R_{22}^\intercal \end{bmatrix} Q^\intercal \\ &= [(\Pi_1 R_{11}^\intercal + \Pi_2 R_{12}^\intercal) \;\; \Pi_2 R_{22}^\intercal] \, Q^\intercal. \end{aligned}$$

Then, applying an SVD decomposition to $\Pi_1 R_{11}^\intercal + \Pi_2 R_{12}^\intercal$, we get

$$\Pi_1 R_{11}^\intercal + \Pi_2 R_{12}^\intercal = U \Sigma V^\intercal. \tag{13}$$

We thus have the full decomposition (QRTP + SVD) of the (wide) unfolding matrix A:

$$A = [U\Sigma V^\intercal \;\; \Pi_2 R_{22}^\intercal] \, Q^\intercal. \tag{14}$$

Finally, we use $U_i = U \in \mathbb{R}^{m \times k}$ as the factor matrix.

Putting everything together we thus arrive at Algorithm 2 for the computation of a Tucker decomposition of a tensor using QRTP:

Algorithm 2 HO-QRTP: Low-rank approximation of a tensor
Input: A tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and a set of ranks $\{k_i\}_{1 \le i \le d}$
Output: A Tucker decomposition (a core tensor $\mathcal{C}$ and factor matrices $\{U_i\}_{1 \le i \le d}$ such that $\widetilde{\mathcal{A}} = \mathcal{C} \times_1 U_1 U_1^\intercal \times_2 \cdots \times_d U_d^\intercal$)

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 A_i is tall **then**
5: Unfold processors according to the i-mode unfolding so that A_{i^2} is logically distributed on the processor grid
6: Run QRTP (for rank k_i) on A_{i^2} to get $A_{i^2}\Pi = QR$
7: Truncate Q to get the factor matrix $U_i = Q[:, 1{:}k_i]$
8: **else** /* A_i is wide */
9: Unfold processors according to the i-mode unfolding so that $A_{i^2}^\intercal$ is logically distributed on the processor grid
10: Run QRTP (for rank k_i) on $A_{i^2}^\intercal$ to get $A_{i^2}^\intercal \Pi = QR$
11: Run SVD on $\Pi_1 R_{11}^\intercal + \Pi_2 R_{12}^\intercal$ to get $\Pi_1 R_{11}^\intercal + \Pi_2 R_{12}^\intercal = U \Sigma V^\intercal$
12: Let $U_i = U[:, 1{:}k_i]$ be the factor matrix associated with the i-mode unfolding
13: **end if**
14: **end for**
15: Compute the core tensor: $\mathcal{C} = \mathcal{A} \times_1 U_1^\intercal \times_2 U_2^\intercal \cdots \times_d U_d^\intercal$
16: Return $\mathcal{C}, U_1, U_2, \ldots, U_d$

In the algorithm, steps 5 and 6 (and 9 and 10) are presented as distinct operations. In practice, however, the unfolding of the processors is implicitly performed during the QR decomposition by dynamically updating the reduction tree (MPI communicator) used by the parallel QRTP algorithm. For further information regarding the parallel aspects of the QRTP algorithm see [4].

4.1 Error Bounds

In Algorithm 2 we use a combination of QRTP and SVD decompositions to generate a set of factor matrices $\{U_i\}_{1 \le i \le d}$ for a d-dimensional tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ according to a set of ranks $\{k_i\}_{1 \le i \le d}$. With these factor matrices we then compute a core tensor $\mathcal{C} \in \mathbb{R}^{k_1 \times \cdots \times k_d}$, where

$$\mathcal{C} = \mathcal{A} \times_1 U_1^\intercal \times_2 \cdots \times_d U_d^\intercal,$$

and where $U_i \in \mathbb{R}^{n_i \times k_i}$. With this Tucker decomposition ($\mathcal{C}$ and $\{U_i\}_{1 \le i \le d}$) we can construct a compressed version (low-rank approximation) of $\widetilde{\mathcal{A}}$:

$$\widetilde{\mathcal{A}} = \mathcal{C} \times_1 U_1 \times_2 \cdots \times_d U_d.$$

In order to prove the error bound between $\mathcal{A}$ and $\widetilde{\mathcal{A}}$ in terms of the Frobenius norm ($\|\cdot\|_F$) in Theorem 4.2, we first establish the following lemma:

Lemma 4.1. *Let the matrix $A \in \mathbb{R}^{m \times n}$ be defined by $A = \begin{bmatrix} UB & C \end{bmatrix}$, where $U \in \mathbb{R}^{m \times k}$ contains orthonormal columns, $B \in \mathbb{R}^{k \times k}$, $C \in \mathbb{R}^{m \times n-k}$, and $k \in \{1, \ldots, \min(m, n)\}$.*

Then

$$\|(I - UU^\intercal)A\|_F \leq \|C\|_F.$$

Proof.

$$\begin{aligned}
\|(I - UU^\intercal)A\|_F &= \left\|(I - UU^\intercal)\begin{bmatrix} UB & C \end{bmatrix}\right\|_F \\
&= \left\|\begin{bmatrix}(I - UU^\intercal)UB & (I - UU^\intercal)C\end{bmatrix}\right\|_F \\
&= \left\|\begin{bmatrix}(U - UU^\intercal U)B & (I - UU^\intercal)C\end{bmatrix}\right\|_F \\
&= \left\|\begin{bmatrix}(U - U)B & (I - UU^\intercal)C\end{bmatrix}\right\|_F \\
&= \left\|\begin{bmatrix}0 & (I - UU^\intercal)C\end{bmatrix}\right\|_F \\
&= \|(I - UU^\intercal)C\|_F \\
&\leq \|C\|_F .
\end{aligned}$$

□

We can now prove the error bound for $\|\mathcal{A} - \widetilde{\mathcal{A}}\|_F$.

Theorem 4.2. *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 $\mathcal{A}$ are such that the low-rank tensor approximation $\widetilde{\mathcal{A}} \in \mathbb{R}^{n_1\times\cdots\times n_d}$ generated by Algorithm 2 satisfies the error bound,*

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

where

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

and r_i and $\sigma_j(A_i)$ are respectively the rank and singular values of the i-mode unfolding 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.

Proof. In Algorithm 2 we compute each factor matrix differently depending upon whether the associated unfolding matrix is tall ($n_i \geq N_i$) or wide ($n_i < N_i$). We thus have 2 different cases that we need to consider:

Case 1 (tall unfolding) Recall that $A_{i^2}P_i = Q_iR_i$ and that the i^{th} factor matrix contains the leftmost k_i columns of the matrix Q_i. Let $\overline{Q}_i = Q_i[:, 1{:}k_i]$ be the i^{th} factor

matrix. We thus have

$$\begin{aligned}
\|\mathcal{A}-\widetilde{\mathcal{A}}\|_F^2 &\leq \sum_{i=1}^{d} \left\|(I-\overline{Q}_i\overline{Q}_i^\intercal)A_i\right\|_F^2 \qquad \text{(by Lemma 2.1 of [15])} \\
&= \sum_{i=1}^{d} \left\|(I-\overline{Q}_i\overline{Q}_i^\intercal)A_{i^2}\Pi_i^\intercal\right\|_F^2 \qquad \text{(by Theorem 3.1)} \\
&= \sum_{i=1}^{d} \left\|(I-\overline{Q}_i\overline{Q}_i^\intercal)A_{i^2}\right\|_F^2 \qquad \text{(Frobenius norm)} \\
&= \sum_{i=1}^{d} \left\|R_i^{(22)}\right\|_F^2 \qquad \text{(by (19) of [3]).}
\end{aligned} \tag{15}$$

Case 2 (wide unfolding) Recall that U_i is the i^{th} factor matrix, where $i \in \{1,\ldots,d\}$. Using (15) from case 1 and (14) we have

$$\begin{aligned}
\|\mathcal{A}-\widetilde{\mathcal{A}}\|_F^2 &= \sum_{i=1}^{d} \|(I-U_iU_i^\intercal)A_{i^2}\|_F^2 \\
&= \sum_{i=1}^{d} \left\|(I-U_iU_i^\intercal)\begin{bmatrix} U_i\Sigma_iV_i^\intercal & \Pi_2^{(i)}R_i^{(22)\intercal}\end{bmatrix} Q_i^\intercal\right\|_F^2 \\
&= \sum_{i=1}^{d} \left\|(I-U_iU_i^\intercal)\begin{bmatrix} U_i\Sigma_iV_i^\intercal & \Pi_2^{(i)}R_i^{(22)\intercal}\end{bmatrix}\right\|_F^2 \\
&\leq \sum_{i=1}^{d} \left\|R_i^{(22)\intercal}\right\|_F^2 \qquad \text{(by Lemma 4.1)} \\
&= \sum_{i=1}^{d} \left\|R_i^{(22)}\right\|_F^2.
\end{aligned}$$

We thus have the same error bound in both cases. The error is bounded by the sum of the Frobenius norms of the matrices $R_i^{(22)}$.

Now, for any matrix B, the Frobenius norm satisfies $\|B\|_F^2 = \sum_j \sigma_j^2(B)$. Thus

$$\sum_{i=1}^{d} \left\|R_i^{(22)}\right\|_F^2 = \sum_{i=1}^{d}\sum_{j=1}^{r_i-k_i} \sigma_j^2(R_i^{(22)}),$$

where r_i is the rank of the matrix $R_i^{(22)}$. Then, using an error bound from [4], we have

$$\sum_{i=1}^{d}\sum_{j=1}^{r_i-k_i} \sigma_j^2(R_i^{(22)}) \leq \sum_{i=1}^{d}\sum_{j=k_i+1}^{r_i} \sigma_j^2(A_i)[1+\widetilde{f}_i^2(n_i-k_i)].$$

We thus arrive at the desired result:

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

□

Corollary 4.3. *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-rank tensor approximation $\widetilde{\mathcal{A}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ generated by Algorithm 2 satisfies the error bound,*

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

where

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

Proof. By Theorem 4.2 and Theorem 10.3 of [11] we have

$$\begin{aligned}
\|\mathcal{A} - \widetilde{\mathcal{A}}\|_F^2 &\leq \sum_{i=1}^{d} \sum_{j=k_i+1}^{r_i} \sigma_j^2(A_i)[1 + \widetilde{f}_i^2(n_i - k_i)] \\
&\leq (1 + \max_i[\widetilde{f}_i^2 k_i(n_i - k_i)]) \sum_{i=1}^{d} \sum_{j=k_i+1}^{r_i} \sigma_j^2(A_i) \\
&\leq d(1 + \max_i[\widetilde{f}_i^2 k_i(n_i - k_i)]) \, \|\mathcal{A} - \mathcal{A}_{\text{best}}\|_F^2 \,.
\end{aligned}$$

□

4.2 Cost of QRTP for equidimensional processor tensor

As described in Section 3, HO-QRTP is applicable to arbitrary tensors and can be executed on any number of processors. In order to simplify the calculations, we consider here the cost of HO-QRTP applied to an equidimensional tensor where the number of processors is such that we can conveniently organize the processors into an equidimensional processor tensor. In other words, for the total number of processors P, we have $P = l^d$, where l is an integer.

For an equidimensional tensor $\mathcal{A} \in \mathbb{R}^{n \times \cdots \times n}$ each unfolding matrix has the dimensions $n \times n^{d-1}$. Thus, since the unfolding matrices are all wide, Algorithm 2 executes QRTP on the transpose of each of the d unfolding matrices. This also implies that the unfoldings of the processor tensor results in a processor grid for which the number of rows and columns are given by

$$P_r = (\sqrt[d]{P})^{d-1} = P^{\frac{d-1}{d}}, \quad P_c = \sqrt[d]{P} = P^{\frac{1}{d}},$$

where P is the total number of processors. Similarly, for the block height and width of the unfolding matrices we have

$$b_r = \left(\frac{n}{\sqrt[d]{P}}\right)^{d-1} = \frac{n^{d-1}}{P^{\frac{d-1}{d}}}, \quad b_c = \frac{n}{\sqrt[d]{P}} = nP^{-\frac{1}{d}}.$$

Then, using the cost of QRTP from [4], the maximum number of flops per processor for QRTP (for d unfolding matrices) is

$$\#\text{flops} = d\left[4\frac{n^d}{P}k + K_1 k^3 + K_2 \frac{n^{d-1}}{P^{\frac{d-1}{d}}} k^2\right].$$

where

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

The maximum number of messages (as defined in [4]) required for the QRTP operations is

$$\#\text{messages} = d(\log_2 P_c + \log_2 P_r)(1+\log_2 P_r) = d\log_2 P(1+\log_2 P_r).$$

4.3 Cost of QRTP for general processor tensor

Here we consider the cost of HO-QRTP applied to an equidimensional tensor for a general processor tensor $\mathcal{P}$ that has dimensions $p_1 \times \cdots \times p_d$ where the total number of processors is given by $P = \prod_{i=1}^{d} p_i$, and where $p_i \geq 1$ for all $i \in \{1,\ldots,d\}$. For example, for a 3-dimensional tensor and 16 processors we could use the processor tensor $\mathcal{P} \in \mathbb{N}^{4\times 1\times 4}$. In this case $p_1 = p_3 = 4$ and $p_2 = 1$.

As in the previous section, given that the tensor $\mathcal{A} \in \mathbb{R}^{n\times\cdots\times n}$ is equidimensional, Algorithm 2 executes QRTP on the transpose of each of the d unfolding matrices. The number of rows of the processor grid and the block height associated with the i-mode unfolding are thus given by

$$P_r^{(i)} = \prod_{\substack{j=1\\ j\neq i}}^{d} p_j = \frac{P}{p_i}, \quad P_c^{(i)} = p_i, \quad b_r^{(i)} = \frac{n^{d-1}p_i}{P}.$$

Then, using the cost of QRTP from [4], the maximum number of flops per processor for QRTP (for d unfolding matrices) is

$$\#\text{flops} = \sum_{i=1}^{d}\left[4\frac{n^d}{P}k + K_1^{(i)}k^3 + K_2^{(i)}\frac{n^{d-1}p_i}{P}k^2\right].$$

where

$$\begin{aligned}
K_1^{(i)} &= 16\log_2 P + 8\log_2 P_r^{(i)}\left[1+\frac{5}{3}\log_2 P_r^{(i)}\right] + \frac{16}{3}\left[5\log_2 P_r^{(i)} - 1\right]\\
&= 24\log_2 P - 8\log_2 p_i + \frac{40}{3}\log_2 P_r^{(i)}\log_2 P_r^{(i)} + \frac{16}{3}\left[5\log_2 P_r^{(i)} - 1\right]\\
&= \frac{152}{3}\log_2 P - \frac{104}{3}\log_2 p_i + \frac{40}{3}(\log_2^2 P - 2\log_2 P\log_2 p_i + \log_2^2 p_i) - \frac{16}{3}\\
&\leq 78\log_2^2 P,\\
K_2^{(i)} &= 8(\log_2 P_r^{(i)} + 1) = 8(\log_2 P - \log_2 p_i + 1) \leq 8(\log_2 P + 1).
\end{aligned}$$

And thus

$$\#\text{flops} \leq d\left[4k\frac{n^d}{P} + 78k^3\log_2^2 P + 8k^2\frac{n^{d-1}p_i}{P}(\log_2 P + 1)\right].$$

The maximum number of messages (as defined in [4]) required for the QRTP operations is

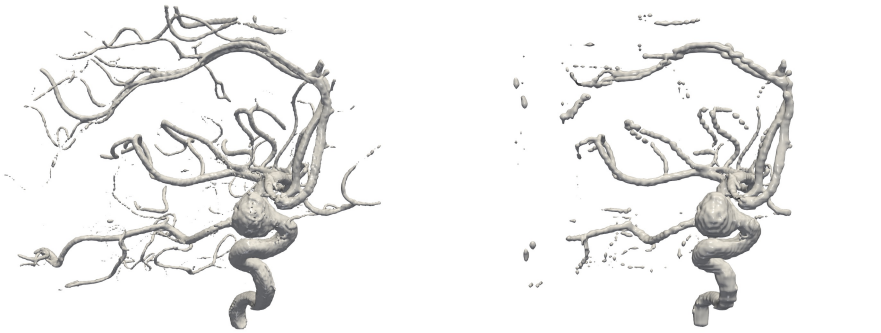
$$\begin{aligned}
\#\text{messages} &= \sum_{i=1}^{d}(\log_2 P_c^{(i)} + \log_2 P_r^{(i)})(1+\log_2 P_r^{(i)}) = \sum_{i=1}^{d}\log_2 P(1+\log_2 P_r^{(i)})\\
&\leq d\log_2 P(1+\log_2 P).
\end{aligned}$$

5 Numerical Experiments

The numerical experiments described below were coded in C++ using GCC 8.3, Open MPI 3.1 and OpenBLAS 0.3. The code was executed on a network of 1280 logical cores consisting of 20 nodes, each equipped with 2 Cascade Lake Intel Xeon 5218 processors.

5.1 3D Image Compression

The following figures demonstrate the compression of a 256×256×256 3D image of an aneurysm from TC18 [12] that was compressed using HO-QRTP (Algorithm 2). Each image is an isosurface view of the compressed or uncompressed 3D image data. The compressed version (low-rank approximation) of the image tensor has rank 64×64×64. Given that each of the unfolding matrices is wide (dimensions 256×65536), the factor matrices are all extracted following case 2 of Algorithm 2. In other words, each factor matrix is extracted according to (14).



(a) Uncompressed.

(b) Rank 64×64×64 approximation on 64 processors.

Figure 4: Isosurface views of 256×256×256 aneurysm 3D image from https://tc18.org/3D_images.html.

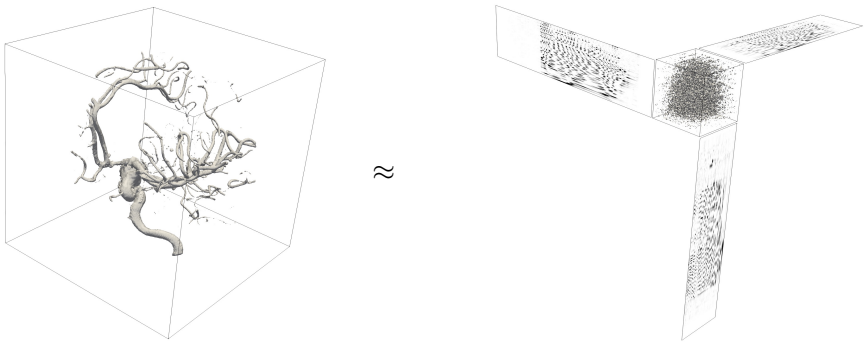


Figure 5: 256×256×256 aneurysm tensor and its rank 64×64×64 Tucker decomposition computed with HO-QRTP.

Figure 6 shows the singular values of the compressed and uncompressed aneurysm 3D image for the 1-mode unfolding matrix. The singular values are similarly distributed for the other two unfolding matrices. We note that the largest singular values for the compressed tensor remain very close to those of the uncompressed tensor. This indicates that, for tensors with rapidly decaying singular values, HO-QRTP exhibits very good compression properties.

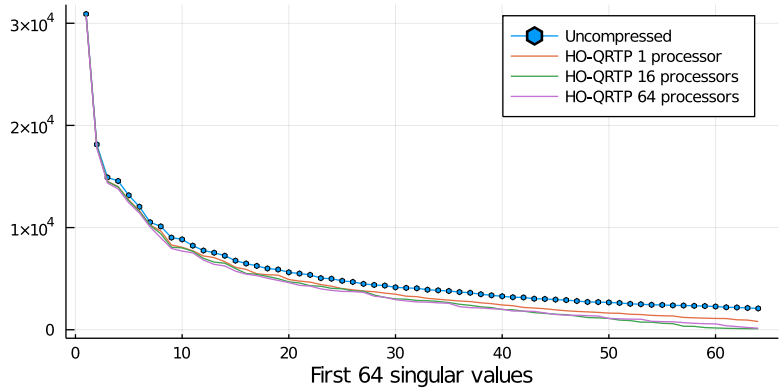


Figure 6: Singular values of the 1-mode unfolding matrix of the uncompressed and compressed aneurysm 3D image.

5.2 Strong-Scaling Performance

Figure 7 and Figure 8 show the strong-scaling performance of HO-QRTP (Algorithm 2) applied to a 1024×1024×1024 tensor whose elements are computed using the logarithm function[1]:

$$\mathcal{A}_{ijk} = \log(i + 2j + 3k). \tag{16}$$

Given that each of the unfolding matrices is wide (dimensions 1024×1024^2), the factor matrices are extracted by applying QRTP to the transposes of the unfolding matrices (case 2 of Algorithm 2).

Figure 7 shows the total execution time and speedup (relative to 8 processors) for the generation of the rank 16×16×16 Tucker decomposition of the logarithm tensor performed on varying numbers of processors. The QRTP slice of each bar is the sum of the 3 execution times of the QRTP algorithm (1 for each unfolding). The Matrix Product slice of each bar is the sum of the 3 matrix products performed during the extraction of the 3 factor matrices. The current implementation of the algorithm requires this step in order to generate the first k rows (here $k = 16$) of the matrix R from (11).

The speedup represented in the bottom graph of Figure 7 is defined as t_8/t_n, where n is the number of processors. We note in particular that the speedup for 512 processors relative to 8 processors is approximately 11.

Figure 8 focuses specifically on the strong-scaling performance of just the QRTP operations performed during HO-QRTP. The orange line represents the total time in seconds for the 3 QRTP operations, and the black line represents the percentage of the total time required for the 3 QRTP operations relative to the HO-QRTP total time.

These figures indicate that HO-QRTP scales consistently as we increase the number of processors. In particular, the QRTP operations scale similarly to the highly optimized ScaLAPACK operations utilized by HO-QRTP. The ScaLAPACK operations include `pdgeqpf` (for QR decompositions), `pdormqr` (for matrix products with an orthogonal matrix), and `pdgemm` (for ordinary matrix products).

[1] See, e.g., [1]

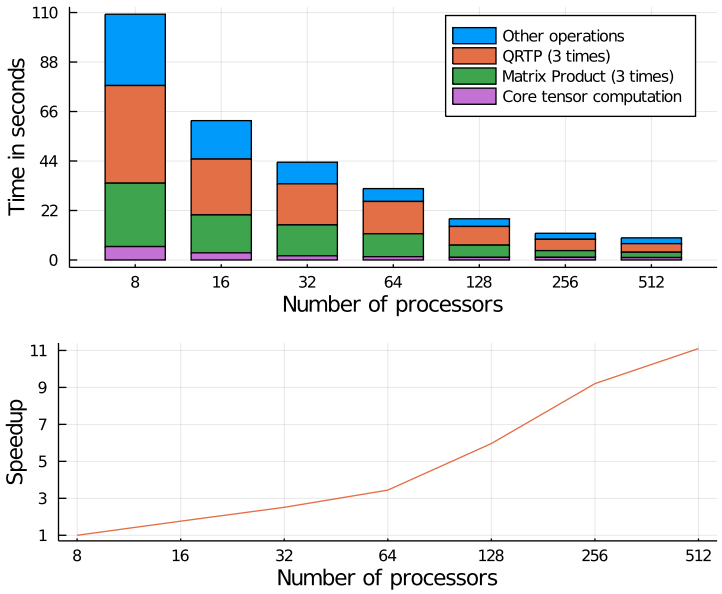


Figure 7: Strong-scaling performance and speedup of HO-QRTP applied to logarithm tensor of size 1024×1024×1024 on varying numbers of processors.

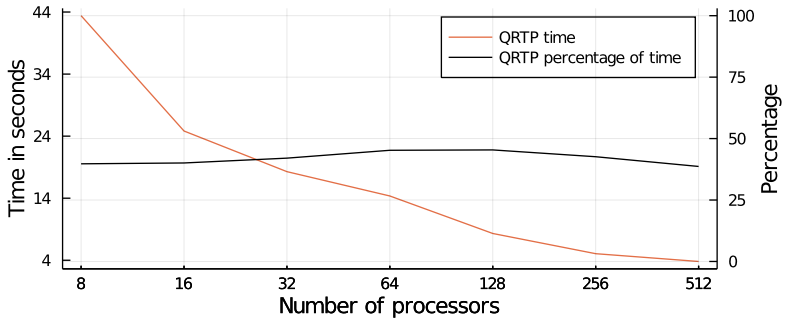


Figure 8: Strong-scaling performance of QRTP operations performed during HO-QRTP applied to logarithm tensor of size 1024×1024×1024 on varying numbers of processors.

As we might expect, doubling the number of processors does not double the speed of the algorithm. This is certainly due to the increased communication overhead.

5.3 Weak-Scaling Performance

Figure 9 and Figure 10 show the weak-scaling performance of HO-QRTP applied to logarithm tensors (16) of varying sizes on a varying number of processors where the block size is fixed at 256×256×256. Given that each of the unfolding matrices is wide (dimensions 1024×1024^2), the factor matrices are extracted by applying QRTP to the transposes of the unfolding matrices (case 2 of Algorithm 2).

Figure 9 shows the total execution time and weak-scaling efficiency (relative to 8 processors) for the generation of the rank 16×16×16 Tucker decomposition of the logarithm tensor performed on varying numbers of processors. The QRTP slice of each bar is the sum of the 3 execution times of the QRTP algorithm (1 for each unfolding). The Matrix Product slice of each bar is the sum of the 3 matrix products performed during the extraction of the 3 factor matrices. The current implementation of the algorithm requires this step in order to generate the first k rows (here $k = 16$) of the matrix R from (11).

The efficiency represented in the bottom graph of Figure 9 is defined as t_8/t_n, where n is the number of processors. For the first 3 configurations (8, 16 and 32 processors) a single node was used. For the remaining configurations we doubled the number of nodes, each containing 32 processors. We note in particular that, starting from 32 processors, the efficiency remains almost constant. This indicates that we can handle larger problems by adding additional nodes (each with 32 cores) without sacrificing efficiency.

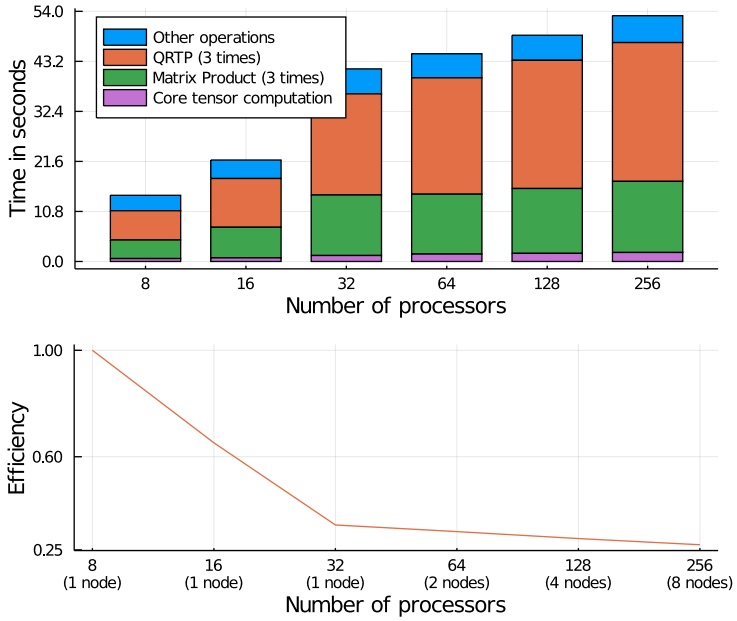


Figure 9: Weak-scaling performance and efficiency of HO-QRTP applied to logarithm tensor of varying sizes on a varying numbers of processors where block size is fixed at 256×256×256.

Figure 10 shows the weak-scaling performance of just the QRTP operations performed during HO-QRTP. The orange line represents the total time in seconds for the 3 QRTP

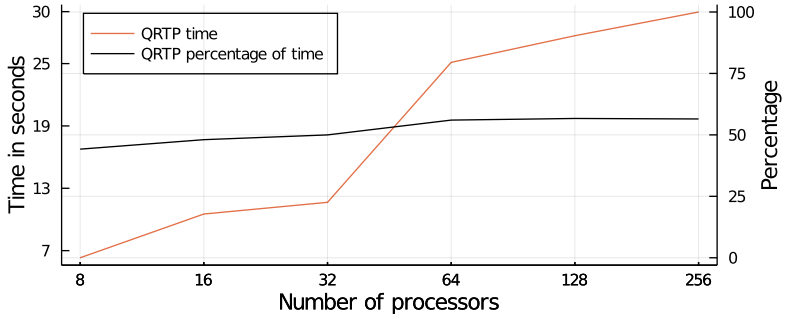


Figure 10: Weak-scaling performance of QRTP operations performed during HO-QRTP applied to logarithm tensor of size 1024×1024×1024 on varying numbers of processors.

operations, and the black line represents the percentage of the total time required for the 3 QRTP operations relative to the HO-QRTP total time. As with strong-scaling performance, these results indicate that the QRTP operations scale similarly to ScaLAPACK operations.

6 Conclusion

In this work we introduced the algorithm, Higher-Order QR with Tournament Pivoting for Tensor Compression (HO-QRTP). HO-QRTP is a parallel variant of HOSVD that utilizes QRTP (instead of SVD) and partitioned unfolding to extract factor matrices from the unfolding matrices of an arbitrary tensor. We have shown that HO-QRTP provides good error bounds for tensor approximations, and exhibits good strong and weak-scaling performance. 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 extracted without additional data retrieval. Although the SVD decomposition provides the best low-rank approximation, the relative rapidity and scalability of QRTP makes it more amenable to the compression of large-scale tensors.

7 Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 810367).

References

[1] S. Ahmadi-Asl, A. Cichocki, A. H. Phan, I. Oseledets, S. Abukhovich, and T. Tanaka, *Randomized Algorithms for Computation of Tucker decomposition and Higher Order SVD (HOSVD)*, 2020.

[2] 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), (2016).

[3] A. Ayala, X. Claeys, and L. Grigori, *ALORA: Affine Low-Rank Approximations*, Journal of Scientific Computing, 79 (2019), pp. 1135–1160.

[4] M. Beaupère and L. Grigori, *Communication avoiding low rank approximation based on QR with tournament pivoting.* working paper or preprint, Sept. 2020.

[5] P. Businger and G. H. Golub, *Linear Least Squares Solutions by Housholder Transformations*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1971, pp. 111–118.

[6] J. Demmel, L. Grigori, and A. Rusciano, *An improved analysis and unified perspective on deterministic and randomized low rank matrix approximations*, ArXiv, abs/1910.00223 (2019).

[7] 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 (2015), pp. 55–89.

[8] C. Eckart and G. Young, *The approximation of one matrix by another of lower rank*, Psychometrika, 1 (1936), pp. 211–218.

[9] L. Grigori, *Introduction to Communication Avoiding Algorithms for Direct Methods of Factorization in Linear Algebra*, Springer International Publishing, Cham, 2017, pp. 153–185.

[10] M. Gu and S. C. Eisenstat, *Efficient Algorithms for Computing a Strong Rank-Revealing QR Factorization*, SIAM Journal on Scientific Computing, 17 (1996), pp. 848–869.

[11] W. Hackbusch, *Tensor Spaces and Numerical Tensor Calculus*, Springer-Verlag Berlin Heidelberg, 2012.

[12] IAPR, *TC18 Discrete Geometry and Mathematical Morphology.* Available online.

[13] JuliaLang.org, *Julia QR Function*, 2020.

[14] T. G. Kolda and B. W. Bader, *Tensor Decompositions and Applications*, SIAM Review, 51 (2009), pp. 455–500.

[15] A. K. Saibaba, *HOID: Higher Order Interpolatory Decomposition for tensors based on Tucker representation*, 2015.

[16] P. Shah, C. Wieser, and F. Bry, *Parallel higher-order SVD for tag-recommendations*, 2012.

[17] G. Zhou, A. Cichocki, and S. Xie, *Decomposition of Big Tensors With Low Multilinear Rank*, 2014.