Parallel Tensor Train Rounding using Gram SVD

Hussam Al Daas
Computational Mathematics Group
Rutherford Appleton Laboratory
Didcot, Oxfordshire, UK
hussam.al-daas@stfc.ac.uk

Grey Ballard
Department of Computer Science
Wake Forest University
Winston-Salem, NC, USA
ballard@wfu.edu

Lawton Manning
Department of Computer Science
Wake Forest University
Winston-Salem, NC, USA
mannlg15@wfu.edu

***Abstract*—Tensor Train (TT) is a low-rank tensor representation consisting of a series of three-way cores whose dimensions specify the TT ranks. Formal tensor train arithmetic often causes an artificial increase in the TT ranks. Thus, a key operation for applications that use the TT format is rounding, which truncates the TT ranks subject to an approximation error guarantee. Truncation is performed via SVD of a highly structured matrix, and current rounding methods require careful orthogonalization to compute an accurate SVD. We propose a new algorithm for TT-Rounding based on the Gram SVD algorithm that avoids the expensive orthogonalization phase. Our algorithm performs less computation and can be parallelized more easily than existing approaches, at the expense of a slight loss of accuracy. We demonstrate that our implementation of the rounding algorithm is efficient, scales well, and consistently outperforms the existing state-of-the-art parallel implementation in our experiments.**

I. Introduction

Low-rank representations of tensors help to make algorithms addressing large-scale multidimensional problems computationally feasible. While the size of explicit representations of these tensors grows very quickly (an instance of the "curse of dimensionality"), low-rank representations can often approximate explicit forms to sufficient accuracy while requiring orders of magnitude less space and computational time. For example, suppose a parametrized PDE depends on 10 parameters, where each parameter has 10 possible values. Computing the solution for each of the 10^{10} configurations becomes infeasible even for modest discretizations of the state space, but if the solution depends smoothly on the parameters, then the qualitative behavior of the solution over the entire configuration space can be captured using far fewer than 10^{10} parameters [1], [2], [3].

As we describe in detail in §II, the Tensor Train (TT) format [4] is a low-rank representation with a number of parameters that is linear in the sum of the tensor dimensions, as compared to an explicit representation whose size is the product of the tensor dimensions. The TT format consists of a series of 3-way tensors, or TT cores, with one dimension corresponding to an original tensor dimension and two dimensions corresponding to typically much smaller TT ranks. TT approximations can be computed from explicit tensors as a means of compression for scientific computing and machine learning applications [4], [5], [6], [7], but they are also often used to represent tensors that cannot be formed explicitly at all. In the context of parametrized PDEs, the TT format has been used to represent both the discretized operators as well as the solution, residual, and other related vectors [8], [9], [10], [11]. In this case, TT tensors are manipulated using operations such as additions, dot products, and elementwise multiplications, which causes the TT ranks to grow in size. The key operation that prevents uncontrolled growth in TT ranks is known as TT-Rounding, in which a TT tensor is approximated by another TT tensor with minimal ranks subject to a specified approximation error. This operation requires a sequence of highly structured matrix singular value decomposition (SVD) problems and is typically a computational bottleneck.

There exists a wide array of high-performance, parallel implementations of tensor computations for computing decompositions such as CP and Tucker of dense and sparse tensors [12], [13], [14], [15], [16], [17], as well as for performing contractions of dense, sparse, and structured tensors [18], [19], [20]. However, the available software for computing, manipulating, and rounding TT tensors is largely limited to productivity languages such as MATLAB and Python [21], [22], and there are far fewer parallelizations of TT computations [23], [24], [25]. One of the aims of this paper is to raise the bar for parallel performance for TT-Rounding and demonstrate that TT-based approaches can scale to scientific problems with more and higher dimensions using efficient parallelization. The focus in this paper is on applications arising from parameter dependent PDEs and uncertainty quantification [26], [27], [28], [29], where one or more modes are large, ranging from thousands to millions, and the TT ranks are small, ranging from tens to hundreds.

The TT-Rounding algorithm utilizes multiple truncated SVDs. The central contribution of this paper is the development of a parallel algorithm that performs these truncated SVDs more efficiently than the existing approach, by reducing both computational and communication costs. The basic tool of the algorithm is the Gram SVD algorithm, which exploits the connection between the SVD of a matrix $\mathbf{A}$ and the eigenvalue decomposition of its Gram matrix $\mathbf{A}^\top\mathbf{A}$. The truncated SVD must be performed on a highly structured matrix which is analogous to a matrix represented as $\mathbf{X} = \mathbf{A}\mathbf{B}^\top$, where $\mathbf{A}$ and $\mathbf{B}$ are tall-skinny matrices, as we describe in §III. We present our TT-Rounding algorithm in §IV, showing how the ideas from the matrix case can be applied within the context of TT. The key to efficiency of our TT-Rounding is the computation of Gram matrices of matrices with overlapping TT structure.

1530-2075/22/$31.00 ©2022 IEEE
DOI 10.1109/IPDPS53621.2022.00095

We present performance results in §V, demonstrating the efficiency of our algorithm compared to the existing state of the art. In a MATLAB-based experiment, we show that improvement of a TT-Rounding implementation leads to overall performance improvement for a TT-based linear solver. Then we demonstrate that our parallel implementation written in C/MPI is both weakly and strongly scalable on TT tensors with representative dimensions and ranks. In particular, we achieve up to $39\times$ parallel speedup when scaling from 1 node to 64 nodes of a distributed-memory platform for rounding a 16-way tensor with dimensions of size $100M \times 50K \times \cdots \times 50K \times 10M$ and TT ranks all of size 20. On that tensor, we achieve a $6\times$ speedup over a state-of-the-art implementation of the standard TT-Rounding approach using 64 nodes. We also observe a $28\times$ speedup over the same implementation on a smaller tensor with memory footprint less than 1MB using a single node (32 cores). Our results demonstrate that TT-Rounding is highly efficient and scalable using our algorithm, and we target parallelization of TT-based solvers based on our approach as future work.

II. Preliminaries

A. Tensor Train Notation

An N-mode or order-N low rank tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is in the Tensor Train (TT) format if there exist strictly positive integers $R_0, \ldots, R_N$ with $R_0 = R_N = 1$ and N order-3 tensors $\mathcal{T}_{\mathcal{X},1}, \ldots, \mathcal{T}_{\mathcal{X},N}$, called TT cores, with $\mathcal{T}_{\mathcal{X},n} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$, such that:

$$\mathcal{X}(i_1, \ldots, i_N) = \mathcal{T}_{\mathcal{X},1}(i_1, :) \cdots \mathcal{T}_{\mathcal{X},n}(:, i_n, :) \cdots \mathcal{T}_{\mathcal{X},N}(:, i_N).$$

Since $R_0 = R_N = 1$, the first and last TT cores are (order-2) matrices so $\mathcal{T}_{\mathcal{X},1}(i_1, :) \in \mathbb{R}^{R_1}$ and $\mathcal{T}_{\mathcal{X},N}(:, i_N) \in \mathbb{R}^{R_{N-1}}$ and hence $\mathcal{T}_{\mathcal{X},1}(i_1, :) \cdots \mathcal{T}_{\mathcal{X},n}(:, i_n, :) \cdots \mathcal{T}_{\mathcal{X},N}(:, i_N) \in \mathbb{R}$. We refer to the $R_{n-1} \times R_n$ matrix $\mathcal{T}_{\mathcal{X},n}(:, i_n, :)$ as the i_nth slice of the nth TT core of $\mathcal{X}$, where $1 \leq i_n \leq I_n$.

Different types of matricization (also known as unfolding) of a tensor are used to express linear algebra operations on tensors. The mode-n unfolding maps mode-n fibers (vectors) to the columns of a matrix, denoted by $\mathbf{X}_{(n)}$ for a tensor $\mathcal{X}$. The tensor-times-matrix product is specified for a particular mode of the tensor and is defined so that $\mathcal{Y} = \mathcal{X} \times_n \mathbf{M}$ is equivalent to $\mathbf{Y}_{(n)} = \mathbf{M}\mathbf{X}_{(n)}$.

In this work, we will often use two particular matricizations of 3D tensors. The *horizontal unfolding* of TT core $\mathcal{T}_{\mathcal{X},n}$ corresponds to stacking the slices $\mathcal{T}_{\mathcal{X},n}(:, i_n, :)$ for $i_n = 1, \ldots, I_n$ horizontally and is equivalent to the standard mode-1 unfolding. The horizontal unfolding operator is denoted by $\mathcal{H}$, therefore, $\mathcal{H}(\mathcal{T}_{\mathcal{X},n}) \in \mathbb{R}^{R_{n-1} \times R_n I_n}$. The *vertical unfolding* corresponds to stacking the slices $\mathcal{T}_{\mathcal{X},n}(:, i_n, :)$ for $i_n = 1, \ldots, I_n$ vertically and is the transpose of the standard mode-3 unfolding. The vertical unfolding operator is denoted by $\mathcal{V}$, therefore, $\mathcal{V}(\mathcal{T}_{\mathcal{X},n}) \in \mathbb{R}^{R_{n-1} I_n \times R_n}$. These two unfoldings are important for the linearization of tensor entries in memory as they enable performing matrix operations on the TT core without shuffling or permuting data.

Another type of unfolding which we will use to express mathematical relationships among TT cores maps the first n modes to rows and the rest to columns [30]. We use the notation $\mathbf{X}_{(1:n)}$ to represent this unfolding, so that $\mathbf{X}_{(1:n)} \in \mathbb{R}^{I_1 \cdots I_n \times I_{n+1} \cdots I_N}$. The nth TT rank of $\mathcal{X}$ is the rank of $\mathbf{X}_{(1:n)}$.

B. Gram SVD

Given a tall and skinny matrix $\mathbf{A}$, recall that the corresponding Gram matrices are $\mathbf{A}\mathbf{A}^\top$ and $\mathbf{A}^\top\mathbf{A}$. We are typically interested in $\mathbf{G}_A = \mathbf{A}^\top\mathbf{A}$ for efficient algorithms because it is a smaller matrix.

Gram SVD is an algorithm that exploits the connection between the SVD of a matrix and the eigenvalue decompositions of its Gram matrices. For $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, we have $\mathbf{G}_A = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top$. We see that the eigenvalues of $\mathbf{G}_A$ are the squares of the singular values of $\mathbf{A}$ and the eigenvectors of $\mathbf{G}_A$ are the right singular vectors of $\mathbf{A}$. We can recover the left singular vectors via $\mathbf{U} = \mathbf{A}\mathbf{V}\mathbf{\Sigma}^{-1}$ (assuming full rank). Gram SVD computes an accurate decomposition but suffers from higher orthogonality error of $\mathbf{U}$ as well as reduced accuracy of the singular values. SVD algorithms using orthogonal transformations compute singular values with error proportional to $\|\mathbf{A}\| \cdot \varepsilon$, where ε is the working precision, while the error for Gram SVD can be larger by a factor as large as the condition number of $\mathbf{A}$ [31]. This implies that backwards stable SVD algorithms can compute singular values in a range of $1/\varepsilon$, while Gram SVD is limited to computing singular values in a range of $1/\sqrt{\varepsilon}$.

C. Cookies Problem and TT-GMRES

As a concrete example of a parametrized PDE for which TT methods work well, we consider the two-dimensional *cookies problem* [26], [32] described as follows:

$$\begin{aligned} -\text{div}(\sigma(x, y; \boldsymbol{\rho})\nabla(u(x, y; \boldsymbol{\rho}))) &= f(x, y) && \text{in } \Omega, \\ u(x, y; \boldsymbol{\rho}) &= 0 && \text{on } \partial\Omega, \end{aligned}$$

where Ω is $(-1, 1) \times (-1, 1)$, $\partial\Omega$ is the boundary of Ω and σ is defined as:

$$\sigma(x, y; \boldsymbol{\rho}) = \begin{cases} 1 + \rho_i & \text{if } (x, y) \in D_i \\ 1 & \text{elsewhere} \end{cases}$$

where D_i for $i = 1, \ldots, p$ are disjoint disks distributed in Ω such that their centers are equidistant and ρ_i is selected from a set of samples $J_i \subset \mathbb{R}$ for $i = 1, \ldots, p$. One way to solve this problem is, for each combination of values $(\rho_1, \ldots, \rho_p)$, to solve the linear system $(\mathbf{G}_{1,1} + \sum_{i=1}^{p} \rho_i \mathbf{G}_{i+1,1})\,\mathbf{u} = \mathbf{f}$, where $\mathbf{G}_{1,1} \in \mathbb{R}^{I_1 \times I_1}$ is the discretization of the operator $-\text{div}(\nabla(\cdot))$ in Ω, $\mathbf{G}_{i+1,1}$ is the discretization of $-\text{div}(\chi_{D_i}\nabla(\cdot))$ in Ω where χ_S is the indicator function of the set S, and $\mathbf{f}$ is the discretization of the function f. The number of linear systems to solve in that case is the product of the cardinalities of the sets $(J_i)_{1 \leq i \leq p}$. Knowing that the set of solutions can be well approximated by a low-rank tensor [1], [3], another approach to solve the problem is to use an iterative method that exploits the low-rank structure and solves one large system including

all combinations of parameters. That is, we solve a $(p+1)$-order problem of the form $\mathcal{G}\mathcal{U} = \mathcal{F}$. The operator $\mathcal{G}$ is given as $\mathcal{G} = \sum_{i=1}^{p+1} \mathbf{G}_{i,1} \otimes \cdots \otimes \mathbf{G}_{i,p+1}$, where $\mathbf{G}_{i,i} \in \mathbb{R}^{I_i \times I_i}$ for $i = 2, \ldots, p+1$ is a diagonal matrix containing the samples of ρ_i, and the remaining matrices $\mathbf{G}_{i,j}$ for $i = 1, \ldots, p+1$, $j = 2, \ldots, p+1$ and $j \neq i$ are the identity matrices of suitable size. The right-hand side $\mathcal{F} = \mathbf{f} \otimes \mathbf{1}_{I_2} \otimes \cdots \otimes \mathbf{1}_{I_{p+1}}$, where $\mathbf{1}_{I_i}$ is the vector of ones of size I_i.

In this application and many others, the operator $\mathcal{G}$ has a low operator rank and the right-hand side $\mathcal{F}$ is given in a low-rank form [2], [33], [9], [10], [34]. One way to approximate the solution by a low-rank tensor is to apply a Krylov method adapted to low rank tensors such as TT-GMRES [8]. In each iteration, the operator $\mathcal{G}$ is applied to a low rank tensor leading to a formal expansion of the ranks. Furthermore, one needs to orthogonalize the new basis tensor against previous ones by using a Gram–Schmidt procedure, see Alg. 1. Again, the ranks will increase formally. In order to keep memory and computations tractable, one has to round the resulting tensors after performing these two steps. Most of the time, a small reduction in the final relative residual norm is sufficient, which allows performing aggressive TT-Rounding with loose tolerances. Note that a structured preconditioner is usually combined with TT-GMRES to accelerate the convergence and potentially decrease the TT ranks of the Krylov basis tensors [8]. A simple choice is the mean preconditioner, which is of operator rank one [26].

Algorithm 1 TT-GMRES [8]

function $\mathcal{U} = $ TT-GMRES$(\mathcal{G}, \mathcal{F}, m, \varepsilon)$
  Set $\beta = \|\mathcal{F}\|_F$, $\mathcal{V}_1 = \mathcal{F}/\beta$, $r = \beta$
  for $j = 1 : m$ **do**
    Set $\delta = \frac{\varepsilon\beta}{r}$
    $\mathcal{W} = $ TT-ROUND$(\mathcal{G}\mathcal{V}_j, \delta)$
    for $i = 1 : j$ **do**
      $\mathbf{H}(i,j) = $ INNERPROD$(\mathcal{W}, \mathcal{V}_i)$
    end for
    $\mathcal{W} = $ TT-ROUND$(\mathcal{W} - \sum_{i=1}^{j} \mathbf{H}(i,j)\mathcal{V}_i, \delta)$
    $\mathbf{H}(j+1,j) = \|\mathcal{W}\|_F$
    $r = \min \|\mathbf{H}(1:j+1, 1:j)\mathbf{y} - \beta\mathbf{e}_1\|_2$
    $\mathcal{V}_{j+1} = \mathcal{W}/\mathbf{H}(j+1,j)$
  end for
  $\mathbf{y}_m = \arg\min_{\mathbf{y}} \|\mathbf{H}\mathbf{y} - \beta\mathbf{e}_1\|_2$
  $\mathcal{U} = \sum_{j=1}^{m} \mathbf{y}_m(j)\mathcal{V}_j$
end function

D. TT-Rounding via Orthogonalization

The standard algorithm for TT-Rounding [4] is given in Alg. 2. This procedure is composed of two phases, an orthogonalization phase and a truncation phase. The orthogonalization phase consists of a sequence of QR decompositions of the vertical unfolding of each core starting from the leftmost to orthogonalize its columns and then a multiplication of the triangular factor by the following core. The truncation phase consists of a sequence of truncated SVDs of the horizontal unfolding of each core starting from the rightmost, leaving its rows orthonormal (set as the leading right singular vectors), and multiplying the preceding core by the singular values and the leading left singular vectors. The direction of these two phases can be reversed. Given a required accuracy, the TT-Rounding procedure provides a quasi-optimal approximation with given TT ranks [4].

Algorithm 2 TT-Rounding via Orthogonalization [4], [25]

function $\mathcal{Y} = $ TT-ROUND-QR$(\mathcal{X}, \varepsilon)$
  Set $\mathcal{T}_{\mathcal{Y},1} = \mathcal{T}_{\mathcal{X},1}$
  for $n = 1$ to $N-1$ **do**
    $[\mathcal{V}(\mathcal{T}_{\mathcal{Y},n}), \mathbf{R}] = \mathrm{QR}(\mathcal{V}(\mathcal{T}_{\mathcal{Y},n}))$
    $\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1}) = \mathbf{R}\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})$
  end for
  Compute $\|\mathcal{X}\| = \|\mathcal{T}_{\mathcal{Y},N}\|_F$ and $\varepsilon_0 = \frac{\|\mathcal{X}\|_F}{\sqrt{N-1}}\varepsilon$
  for $n = N$ down to 2 **do**
    $[\mathbf{Q}, \mathbf{R}] = \mathrm{QR}(\mathcal{H}(\mathcal{T}_{\mathcal{Y},n})^\top)$
    $[\hat{\mathbf{U}}, \hat{\mathbf{\Sigma}}, \hat{\mathbf{V}}] = $ TSVD$(\mathbf{R}, \varepsilon_0)$
    $\mathcal{H}(\mathcal{T}_{\mathcal{Y},n})^\top = \mathbf{Q}\hat{\mathbf{U}}$
    $\mathcal{V}(\mathcal{T}_{\mathcal{Y},n-1}) = \mathcal{V}(\mathcal{T}_{\mathcal{Y},n-1})\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}$
  end for
end function

Algorithm 2 has been parallelized by Al Daas et al. [25], who use a 1-D distribution of TT cores to partition a TT tensor across processors. Each core is distributed over all processors along the physical mode such that each processor owns I_k/P slices of the kth core. This distribution guarantees a load balancing and allows to perform TT arithmetic efficiently. In particular, the QR decompositions are performed via the Tall-Skinny QR algorithm [35], and multiplications involving TT cores are parallelized following the 1D distributions. We improve upon this prior work by using an alternate TT-Rounding approach that avoids QR decompositions, reducing arithmetic by a constant factor and also reducing communication.

III. TRUNCATION OF MATRIX PRODUCT

To gain intuition for the use of Gram SVD within TT-Rounding, we focus in this section on the (degenerate) case of TT with 2 modes, with dimensions $I \times J$. In this case, the tensor is a matrix represented by a low-rank product of matrices:

$$\mathbf{X} = \mathbf{A}\mathbf{B}^\top, \tag{1}$$

where $\mathbf{A}$ and $\mathbf{B}$ are tall and skinny matrices with R columns. The goal is to approximate $\mathbf{X}$ with a lower rank representation

$$\mathbf{X} \approx \hat{\mathbf{A}}\hat{\mathbf{B}}^\top, \tag{2}$$

where $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ have $L < R$ columns.

A. Truncation via Orthogonalization

A numerically accurate and reasonably efficient approach to truncate the representation of $\mathbf{X}$ is via orthogonalization. By computing (compact) QR decompositions $\mathbf{A} = \mathbf{Q}_A\mathbf{R}_A$ and $\mathbf{B} = \mathbf{Q}_B\mathbf{R}_B$, we have

$$\mathbf{X} = \mathbf{Q}_A\mathbf{R}_A\mathbf{R}_B^\top\mathbf{Q}_B^\top \tag{3}$$

and the SVD of $\mathbf{R}_A\mathbf{R}_B^\top$ yields the (compact) SVD of $\mathbf{X}$ because $\mathbf{Q}_A$ and $\mathbf{Q}_B$ have orthonormal columns. Note that $\mathbf{R}_A\mathbf{R}_B^\top$ is $R \times R$, so its SVD is much cheaper to compute.

We formalize this approach in Alg. 3. In order to truncate the rank of $\mathbf{X}$, we can truncate the SVD of $\mathbf{R}_A\mathbf{R}_B^\top$. To obtain factors $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$, we apply $\mathbf{Q}_A$ and $\mathbf{Q}_B$ to the left and right singular vectors, respectively. The singular values can be distributed arbitrarily, we choose to distribute them evenly to left and right factors.

Algorithm 3 Rounding Matrix Product $\mathbf{AB}^\top$ using QR

function $[\hat{\mathbf{A}}, \hat{\mathbf{B}}] = \text{MAT-ROUNDING-QR}(\mathbf{A}, \mathbf{B}, \varepsilon)$
  $[\mathbf{Q}_A, \mathbf{R}_A] = \text{QR}(\mathbf{A})$
  $[\mathbf{Q}_B, \mathbf{R}_B] = \text{QR}(\mathbf{B})$
  $[\hat{\mathbf{U}}, \hat{\mathbf{\Sigma}}, \hat{\mathbf{V}}] = \text{TSVD}(\mathbf{R}_A\mathbf{R}_B^\top, \varepsilon)$
  $\hat{\mathbf{A}} = \mathbf{Q}_A\left(\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}^{1/2}\right)$
  $\hat{\mathbf{B}} = \mathbf{Q}_B\left(\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{1/2}\right)$
end function

B. Truncation via Gram SVD

We now show our proposed method for a faster but potentially less accurate rounding algorithm for the matrix product. Our method is based on the Gram SVD algorithm, but we note it is not a straightforward application. For example, we can represent $\mathbf{X}\mathbf{X}^T$ as $\mathbf{A}\mathbf{B}^\top\mathbf{B}\mathbf{A}^\top$, and while $\mathbf{B}^\top\mathbf{B}$ is $R \times R$, we cannot obtain the eigenvalue decomposition easily without orthogonalizing $\mathbf{A}$. Instead, we consider the Gram matrices of $\mathbf{A}$ and $\mathbf{B}$ separately, letting $\mathbf{G}_A = \mathbf{A}^\top\mathbf{A}$ and $\mathbf{G}_B = \mathbf{B}^\top\mathbf{B}$. For clarity, we first describe the method using Cholesky QR, then discuss pivoting within Cholesky, and finally explain the use of Gram SVD.

1) Cholesky QR: Let us first assume $\mathbf{A}$ and $\mathbf{B}$ are full rank, and use Cholesky QR to orthonormalize the columns of $\mathbf{A}$ and $\mathbf{B}$. Computing Cholesky decompositions, we have $\mathbf{R}_A^\top\mathbf{R}_A = \mathbf{G}_A$ and $\mathbf{R}_B^\top\mathbf{R}_B = \mathbf{G}_B$. Then eq. (3) becomes

$$\mathbf{X} = (\mathbf{A}\mathbf{R}_A^{-1})\mathbf{R}_A\mathbf{R}_B^\top(\mathbf{B}\mathbf{R}_B^{-1})^\top.$$

Given the truncated SVD $\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^\top = \mathbf{R}_A\mathbf{R}_B^\top$, we can compute $\hat{\mathbf{A}} = \mathbf{A}\left(\mathbf{R}_A^{-1}\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}^{1/2}\right)$ and $\hat{\mathbf{B}} = \mathbf{B}\left(\mathbf{R}_B^{-1}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{1/2}\right)$ to obtain eq. (2).

In the case that $\mathbf{A}$ or $\mathbf{B}$ is low rank, the standard Cholesky algorithm will fail, and we must employ a pivoted Cholesky algorithm. Pivoted Cholesky QR works well for the low rank case in exact arithmetic, but in the case of numerically low rank matrices, it provides a sharp truncation for each of $\mathbf{A}$ and $\mathbf{B}$ individually. That is, as soon as a nonpositive diagonal entry is encountered, which corresponds to a singular value of $\mathbf{A}$ or $\mathbf{B}$ on the order of $\sqrt{\varepsilon}$ relative to the largest singular value, the algorithm terminates and approximates all remaining singular values as zeros.

2) Gram SVD: We now consider using the Gram SVD approach, which is more robust than pivoted Cholesky QR. Here, we consider $\mathbf{A}$ and $\mathbf{B}$ to be possibly low rank. Given the SVDs $\mathbf{A} = \mathbf{U}_A\mathbf{\Sigma}_A\mathbf{V}_A^\top$ and $\mathbf{B} = \mathbf{U}_B\mathbf{\Sigma}_B\mathbf{V}_B^\top$, we have eigenvalue decompositions $\mathbf{G}_A = \mathbf{V}_A\mathbf{\Sigma}_A^2\mathbf{V}_A^\top = \hat{\mathbf{V}}_A\hat{\mathbf{\Sigma}}_A^2\hat{\mathbf{V}}_A^\top$ and $\mathbf{G}_B = \mathbf{V}_B\mathbf{\Sigma}_B^2\mathbf{V}_B^\top = \hat{\mathbf{V}}_B\hat{\mathbf{\Sigma}}_B^2\hat{\mathbf{V}}_B^\top$, where $\hat{\mathbf{\Sigma}}_A$ and $\hat{\mathbf{\Sigma}}_B$ represent the nonzero singular values and $\hat{\mathbf{V}}_A$ and $\hat{\mathbf{V}}_B$ are the corresponding vectors. We can then write the corresponding left singular vectors via $\hat{\mathbf{U}}_A = \mathbf{A}\hat{\mathbf{V}}_A\hat{\mathbf{\Sigma}}_A^{-1}$ and $\hat{\mathbf{U}}_B = \mathbf{A}\hat{\mathbf{V}}_B\hat{\mathbf{\Sigma}}_B^{-1}$. With these quantities, eq. (1) becomes

$$\mathbf{X} = \underbrace{(\mathbf{A}\hat{\mathbf{V}}_A\hat{\mathbf{\Sigma}}_A^{-1})}_{\hat{\mathbf{U}}_A}\underbrace{\hat{\mathbf{\Sigma}}_A\hat{\mathbf{V}}_A^\top\hat{\mathbf{V}}_B\hat{\mathbf{\Sigma}}_B}_{\mathbf{M}}\underbrace{(\mathbf{B}\hat{\mathbf{V}}_B\hat{\mathbf{\Sigma}}_B^{-1})}_{\hat{\mathbf{U}}_B}{}^\top = \hat{\mathbf{U}}_A\mathbf{M}\hat{\mathbf{U}}_B^\top.$$

Given the truncated SVD $\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^\top = \mathbf{M}$, we compute

$$\hat{\mathbf{A}} = \mathbf{A}\left(\hat{\mathbf{V}}_A\hat{\mathbf{\Sigma}}_A^{-1}\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}^{1/2}\right) \quad \text{and} \quad \hat{\mathbf{B}} = \mathbf{B}\left(\hat{\mathbf{V}}_B\hat{\mathbf{\Sigma}}_B^{-1}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{1/2}\right)$$

to obtain eq. (2). In the case of numerically low rank matrices $\mathbf{A}$ or $\mathbf{B}$, we note that the Gram SVD will not compute singular values on the order of $\sqrt{\varepsilon}$ (relative to the largest singular value) accurately, but it will approximate them with a small, nonzero quantity, unlike the Pivoted Cholesky QR approach. This leads to more robustness because the corresponding singular vector directions can be amplified in the multiplication by the other matrix to capture better approximation of the product $\mathbf{AB}^\top$.

The algorithm for the Gram SVD approach is given as Alg. 4, which can be adapted to Pivoted Cholesky QR following a modification to the algebra of §III-B1. Note that the distribution of the singular values to the two factor matrices $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ is arbitrary. Because the Gram SVD approach is the most robust Gram-based method and the extra cost is insignificant when R is small, we use Gram SVD in the context of TT-Rounding as described in §IV.

Algorithm 4 Truncated SVD of $\mathbf{AB}^\top$ using Gram SVDs

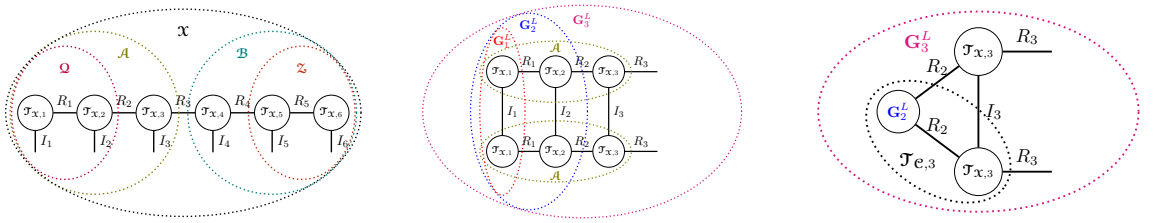
function $[\hat{\mathbf{A}}, \hat{\mathbf{B}}] = \text{TSVD-ABT-GRAM}(\mathbf{A}, \mathbf{B}, \varepsilon)$
  $\mathbf{G}_A = \mathbf{A}^\top\mathbf{A}$
  $\mathbf{G}_B = \mathbf{B}^\top\mathbf{B}$
  $[\mathbf{V}_A, \mathbf{\Lambda}_A] = \text{EIG}(\mathbf{G}_A)$
  $[\mathbf{V}_B, \mathbf{\Lambda}_B] = \text{EIG}(\mathbf{G}_B)$
  $[\hat{\mathbf{U}}, \hat{\mathbf{\Sigma}}, \hat{\mathbf{V}}] = \text{TSVD}(\mathbf{\Lambda}_A^{1/2}\mathbf{V}_A^\top\mathbf{V}_B\mathbf{\Lambda}_B^{1/2}, \varepsilon)$
  $\hat{\mathbf{A}} = \mathbf{A}\left(\mathbf{V}_A\mathbf{\Lambda}_A^{-1/2}\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}^{1/2}\right)$
  $\hat{\mathbf{B}} = \mathbf{B}\left(\mathbf{V}_B\mathbf{\Lambda}_B^{-1/2}\hat{\mathbf{V}}\hat{\mathbf{\Sigma}}^{1/2}\right)$
end function

IV. TT-ROUNDING VIA GRAM SVD

In this section we present the Gram SVD TT-Rounding. In §IV-A, we explain the analogues of matrices $\mathbf{A}$ and $\mathbf{B}$ within the TT-Rounding algorithm, and in §IV-B we show how to compute the Gram matrices for the associated structured matrices. We then present two algorithmic variants of TT-Rounding based on the approach in §IV-C, explain their parallelization in §IV-D, and provide complexity analysis in §IV-E with comparison against the standard TT-Rounding via orthogonalization.

A. TT-Rounding Structure

The nth TT rank of a tensor $\mathcal{X}$ is the rank of the unfolding $\mathbf{X}_{(1:n)}$, which is an $I_1 \cdots I_n \times I_{n+1} \cdots I_N$ matrix where each column is a vectorization of an n-mode subtensor. If $\mathcal{X}$ is



(a) Equation (4) for $N = 6$, $n = 3$ (b) $\mathbf{A}_{(1:n)}{}^\top \mathbf{A}_{(1:n)}$ for $n = 3$ (c) Computing $\mathbf{G}_n^L$ from $\mathbf{G}_{n-1}^L$ for $n = 3$

Fig. 1: Tensor network diagrams

already in TT format, then $\mathbf{X}_{(1:n)}$ has the following structure [25, Eq. (2.3)]:

$$\mathbf{X}_{(1:n)} = (\mathbf{I}_{I_n} \otimes \mathbf{Q}_{(1:n-1)})\mathcal{V}(\mathcal{T}_{\mathcal{X},n})\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})(\mathbf{I}_{I_{n+1}} \otimes \mathbf{Z}_{(1)}), \tag{4}$$

where $\mathcal{Q}$ is $I_1 \times \cdots \times I_{n-1} \times R_{n-1}$ with

$$\mathcal{Q}(i_1, \ldots, i_{n-1}, r_{n-1}) = \mathcal{T}_{\mathcal{X},1}(i_1, :) \cdot \mathcal{T}_{\mathcal{X},2}(:, i_2, :) \cdots \mathcal{T}_{\mathcal{X},n-1}(:, i_{n-1}, r_{n-1}),$$

and $\mathcal{Z}$ is $R_{n+1} \times I_{n+2} \times \cdots \times I_N$ with

$$\mathcal{Z}(r_{n+1}, i_{n+2}, \ldots, i_N) = \mathcal{T}_{\mathcal{X},n+2}(r_{n+1}, i_{n+2}, :) \cdot \mathcal{T}_{\mathcal{X},n+3}(:, i_{n+3}, :) \cdots \mathcal{T}_{\mathcal{X},N}(:, i_N).$$

Truncating or rounding the TT rank of $\mathcal{X}$ in this case corresponds to performing a truncated SVD of $\mathbf{X}_{(1:n)}$. The correctness of Alg. 2 stems from the fact that at the nth step of the truncation loop, the matrix $\mathbf{I}_{I_n} \otimes \mathbf{Q}_{(1:n-1)}$ has orthonormal columns and the matrix $\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})(\mathbf{I}_{I_{n+1}} \otimes \mathbf{Z}_{(1)})$ has orthonormal rows, and therefore the truncated SVD of $\mathcal{V}(\mathcal{T}_{\mathcal{X},n})$ yields the truncated SVD of $\mathbf{X}_{(1:n)}$.

In our proposed approach, we do not impose orthonormality on the exterior matrices and instead use a Gram SVD based approach. To follow the analogy of the matrix case from eq. (1), we consider $\mathbf{A} = \mathbf{A}_{(1:n)} = (\mathbf{I}_{I_n} \otimes \mathbf{Q}_{(1:n-1)})\mathcal{V}(\mathcal{T}_{\mathcal{X},n})$ and $\mathbf{B}^\top = \mathbf{B}_{(1)} = \mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})(\mathbf{I}_{I_{n+1}} \otimes \mathbf{Z}_{(1)})$, where $\mathcal{A}$ and $\mathcal{B}$ are tensors with dimensions $I_1 \times \cdots I_n \times R_n$ and $R_n \times I_{n+1} \times \cdots \times I_N$, respectively. We visualize these relationships using a tensor network diagram [36] in Fig. 1a. In these diagrams, a node represents a tensor, edges represent modes (so that the degree of a node is its dimension), and adjacent nodes represent contractions. To perform the truncation, we first compute $\mathbf{A}^\top\mathbf{A}$ and $\mathbf{B}^\top\mathbf{B}$ as described in §IV-B. Then we follow the approach as described in §III-B2 and finally compute $\hat{\mathbf{A}}$ and $\hat{\mathbf{B}}$ by updating only $\mathcal{V}(\mathcal{T}_{\mathcal{X},n})$ and $\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})$, leaving the TT cores that constitute $\mathcal{Q}$ and $\mathcal{Z}$ unchanged.

B. Structured Gram Matrix Computation

Considering $\mathbf{A}_{(1:n)} = (\mathbf{I}_{I_n} \otimes \mathbf{Q}_{(1:n-1)})\mathcal{V}(\mathcal{T}_{\mathcal{X},n})$ as the matrix $\mathbf{A}$ in our matrix product example, our goal is to compute $\mathbf{A}^\top\mathbf{A}$ exploiting the structure of $\mathbf{A}$ (and the internal structure of $\mathbf{Q}_{(1:n-1)}$). This can also be seen as a contraction between $\mathcal{A}$, a tensor of dimension $n+1$, and itself in the first n modes.

The structure is easiest to understand in the form of a tensor network diagram, as we show in Fig. 1b. In the figure, we have $n = 3$, so that $\mathcal{A}$ is a 4-way tensor composed of 3 TT cores. To visualize contracting $\mathcal{A}$ with itself and compute $\mathbf{G}_3^L = \mathbf{A}_{(1:3)}{}^\top \mathbf{A}_{(1:3)}$, we draw $\mathcal{A}$ twice and connect edges corresponding to the modes with dimensions I_1, I_2, and I_3. After all connected modes are contracted, we are left with 2 un-contracted modes, each of dimension R_3, corresponding to a square output matrix (which is also symmetric). We use the notation $\mathbf{G}_3^L$ to signify that $\mathcal{A}$ is composed of left-most cores and has dimension $R_3 \times R_3$.

The most efficient way to perform the contractions to compute $\mathbf{G}_n^L = \mathbf{A}_{(1:n)}{}^\top \mathbf{A}_{(1:n)}$ is to work left to right, first contracting the mode with dimension I_1. Because the operation involves two tensors with dimension 2, it corresponds to the (symmetric) matrix multiplication $\mathbf{G}_1^L = \mathcal{V}(\mathcal{T}_{\mathcal{X},1})^\top \mathcal{V}(\mathcal{T}_{\mathcal{X},1})$, where we use the notation $\mathbf{G}_1^L$ because the result is the contraction between the left-most cores and has dimension $R_1 \times R_1$. The next step is to contract the two $\mathcal{T}_{\mathcal{X},2}$ nodes with $\mathbf{G}_1^L$ to compute $\mathbf{G}_2^L$. These two contractions can be performed in either order or simultaneously, exploiting symmetry as we describe below. We continue this process of computing each symmetric Gram matrix from the previous mode's, finally computing $\mathbf{G}_n^L$ from $\mathbf{G}_{n-1}^L$ and the two $\mathcal{T}_{\mathcal{X},n}$ cores. Figure 1c shows the structure of the tensor network before $\mathbf{G}_3^L$ is computed from $\mathbf{G}_2^L$ and the two $\mathcal{T}_{\mathcal{X},3}$ cores.

The key to the efficiency of the structured Gram matrix computation in the context of TT-Rounding is the fact that we obtain *all* Gram matrices $\{\mathbf{G}_n^L\}$ as a by-product of computing the last one, $\mathbf{G}_{N-1}^L$. In this way, we have performed the $\mathbf{A}^\top\mathbf{A}$-analogue computations for truncating all TT ranks with one left-to-right pass over the TT representation of the tensor. In order to compute the $\mathbf{B}^\top\mathbf{B}$-analogue quantities, we make a similar pass from right to left to obtain $\{\mathbf{G}_n^R\}$ for $1 \leq n \leq N-1$. Note that $\mathbf{G}_n^R$ is the contraction between the right-most cores to the right of (and not including) the nth core, so that $\mathbf{G}_n^L$ and $\mathbf{G}_n^R$ are the Gram matrices associated with the truncation of the nth TT rank and are both $R_n \times R_n$.

We now consider two ways of computing $\mathbf{G}_n^L$ from $\mathbf{G}_{n-1}^L$ and two $\mathcal{T}_{\mathcal{X},n}$ cores, which we refer to as non-symmetric and symmetric approaches. Computations for $\mathbf{G}_n^R$ from $\mathbf{G}_{n+1}^R$ are analogous. In the nonsymmetric approach, we contract $\mathbf{G}_{n-1}^L$ with one of the cores, letting $\mathcal{T}_{\mathcal{C},n}$ represent the temporary result as illustrated in Fig. 1c. Here we consider $\mathcal{C}$ to be a TT-

format tensor with the same dimensions and ranks as $\mathcal{X}$ for convenient notation. This contraction is a tensor-times-matrix operation and can be expressed as $\mathcal{T}_{\mathfrak{e},n} = \mathcal{T}_{\mathcal{X},n} \times_1 \mathbf{G}_{n-1}^L$ and computed as $\mathcal{H}(\mathcal{T}_{\mathfrak{e},n}) = \mathbf{G}_{n-1}^L \mathcal{H}(\mathcal{T}_{\mathcal{X},n})$. After the first contraction, $\mathcal{T}_{\mathfrak{e},n}$ and the remaining $\mathcal{T}_{\mathcal{X},n}$ share two modes, and the second contraction is across both modes. This operation can be performed via $\mathbf{G}_n^L = \mathcal{V}(\mathcal{T}_{\mathcal{X},n})^\top \mathcal{V}(\mathcal{T}_{\mathfrak{e},n})$. Note that while the result is symmetric in exact arithmetic, this approach does not assume symmetry, and the result will not be bit-wise symmetric due to roundoff error.

In the symmetric approach, we can use the fact that every Gram matrix is symmetric and positive semi-definite. Thus, we can compute a (pivoted) Cholesky decomposition $\mathbf{G}_{n-1}^L = \mathbf{L}\mathbf{L}^\top$. Then we can contract each $\mathbf{L}$ factor with one of the $\mathcal{T}_{\mathcal{X},n}$ nodes, permuting slices of $\mathcal{T}_{\mathcal{X},n}$ if necessary. Here, one contraction is sufficient because they are equivalent operations, and we can exploit the triangular structure of $\mathbf{L}$ to save half the arithmetic of the tensor-times-matrix operation. Letting $\mathcal{T}_{\mathcal{D},n} = \mathcal{T}_{\mathcal{X},n} \times_1 \mathbf{L}$ represent the result, the second contraction is performed via $\mathbf{G}_n^L = \mathcal{V}(\mathcal{T}_{\mathcal{D},n})^\top \mathcal{V}(\mathcal{T}_{\mathcal{D},n})$ which can be performed symmetrically, again saving half the arithmetic and producing an exactly symmetric result.

As illustrated in Fig. 1, $\mathbf{G}_{n-1}^L$ is a matrix with dimension $R_{n-1} \times R_{n-1}$ and $\mathcal{T}_{\mathcal{X},n}$ has dimensions $R_{n-1} \times I_n \times R_n$. In the nonsymmetric approach, the first contraction requires $2I_n R_{n-1}^2 R_n$ operations, and the second contraction requires $2I_n R_{n-1} R_n^2$ operations. In the symmetric approach, the Cholesky decomposition requires $O(R_{n-1}^3)$ operations, and the two contractions together require $I_n R_{n-1}^2 R_n + I_n R_{n-1} R_n^2$ operations, not including any pivoting that must be performed. Despite the fact that the symmetric approach saves half the flops, we use the nonsymmetric approach in our later experiments because of the empirical performance benefits. We found that the superior performance of `gemm` over `trmm` and `syrk` (and the need to copy data for `trmm`) on our platform outweighs the reduction in arithmetic.

C. Algorithms

Given the approach to computing Gram matrices of the TT-structured matrices described in §IV-B, we now present algorithms for TT-Rounding using the Gram SVD approach. We follow the basic steps outlined in §III-B2: compute Gram matrices of factors, perform eigenvalue decompositions, truncate the combined results using SVD, then apply updates to factors to reduce their dimensions.

As described in §IV-B, with a left-to-right and right-to-left pass of the TT structure, we can obtain the Gram matrices associated with every TT rank truncation. Given its pair of Gram matrices, each TT rank can be truncated independently of all others. We call this approach the *simultaneous* variant to distinguish it from a more computationally efficient method that truncates ranks in *sequence* (described below). The simultaneous variant of the algorithm is given as Alg. 5. Line 2 to line 11 show the set of contractions used to obtain Gram matrices across all modes. Lines 15 to 17 perform the eigenvalue and singular value decompositions of small matrices. Finally, lines 18 and 19 update the TT cores and reduce their dimension. Note that the singular values are distributed evenly to each interior factor, as each is scaled by $\hat{\mathbf{\Sigma}}^{1/2}$, but this distribution is arbitrary.

Algorithm 5 TT-Rounding via Gram SVD (Simultaneous)

1: **function** $\mathcal{Y} = $ TT-ROUND-GRAM-SIM($\mathcal{X}, \varepsilon$)
2: $\quad\mathbf{G}_1^L = \mathcal{V}(\mathcal{T}_{\mathcal{X},1})^\top \mathcal{V}(\mathcal{T}_{\mathcal{X},1})$
3: $\quad$**for** $n = 2$ to $N-1$ **do**
4: $\quad\quad\mathcal{H}(\mathcal{T}_{\mathfrak{e},n}) = \mathbf{G}_{n-1}^L \mathcal{H}(\mathcal{T}_{\mathcal{X},n})$
5: $\quad\quad\mathbf{G}_n^L = \mathcal{V}(\mathcal{T}_{\mathcal{X},n})^\top \mathcal{V}(\mathcal{T}_{\mathfrak{e},n})$
6: $\quad$**end for**
7: $\quad\mathbf{G}_{N-1}^R = \mathcal{H}(\mathcal{T}_{\mathcal{X},N}) \mathcal{H}(\mathcal{T}_{\mathcal{X},N})^\top$
8: $\quad$**for** $n = N-1$ down to 1 **do**
9: $\quad\quad\mathcal{V}(\mathcal{T}_{\mathfrak{e},n}) = \mathcal{V}(\mathcal{T}_{\mathcal{X},n}) \mathbf{G}_n^R$
10: $\quad\quad\mathbf{G}_{n-1}^R = \mathcal{H}(\mathcal{T}_{\mathfrak{e},n}) \mathcal{H}(\mathcal{T}_{\mathcal{X},n})^\top$
11: $\quad$**end for**
12: $\quad$Compute $\|\mathcal{X}\| = (\mathbf{G}_0^R)^{1/2}$ and $\varepsilon_0 = \frac{\|\mathcal{X}\|}{\sqrt{N-1}}\varepsilon$
13: $\quad\mathcal{T}_{\mathcal{Y},1} = \mathcal{T}_{\mathcal{X},1}$
14: $\quad$**for** $n = 1$ to $N-1$ **do**
15: $\quad\quad[\mathbf{V}_L, \mathbf{\Lambda}_L] = $ EIG($\mathbf{G}_n^L$)
16: $\quad\quad[\mathbf{V}_R, \mathbf{\Lambda}_R] = $ EIG($\mathbf{G}_n^R$)
17: $\quad\quad[\hat{\mathbf{U}}, \hat{\mathbf{\Sigma}}, \hat{\mathbf{V}}] =$TSVD($\mathbf{\Lambda}_L^{1/2} \mathbf{V}_L^\top \mathbf{V}_R \mathbf{\Lambda}_R^{1/2}, \varepsilon_0$)
18: $\quad\quad\mathcal{V}(\mathcal{T}_{\mathcal{Y},n}) = \mathcal{V}(\mathcal{T}_{\mathcal{Y},n}) \cdot (\mathbf{V}_L \mathbf{\Lambda}_L^{-1/2} \hat{\mathbf{U}} \hat{\mathbf{\Sigma}}^{1/2})$
19: $\quad\quad\mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1}) = (\hat{\mathbf{\Sigma}}^{1/2} \hat{\mathbf{V}}^\top \mathbf{\Lambda}_R^{-1/2} \mathbf{V}_R^\top) \cdot \mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})$
20: $\quad$**end for**
21: **end function**

Alternatively, we can truncate the TT ranks in sequence to save some arithmetic by exploiting orthonormality. Following the original approach of TT-Rounding via orthogonalization (Alg. 2), if we truncate the ranks from left to right and pass all singular values to the right, then we maintain orthonormality of the left-most cores. That is, when truncating the nth rank and considering eq. (4), we have that $\mathbf{Q}_{(1:n-1)}$ has orthonormal columns. Thus, truncating $\mathbf{X}_{(1:n)}$ is equivalent to truncating $\mathcal{V}(\mathcal{T}_{\mathcal{X},n}) \mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})(\mathbf{I}_{I_{n+1}} \otimes \mathbf{Z}_{(1)})$. In the standard approach, we also have that $\mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})(\mathbf{I}_{I_{n+1}} \otimes \mathbf{Z}_{(1)})$ has orthogonal rows, but that does not apply here. Instead, we use the analogue of $\mathbf{A} = \mathcal{V}(\mathcal{T}_{\mathcal{X},n})$ and $\mathbf{B}^\top = \mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})(\mathbf{I}_{I_{n+1}} \otimes \mathbf{Z}_{(1)})$. We note that $\mathbf{B}^\top$ is identical to the simultaneous case, so $\mathbf{B}^\top \mathbf{B}$ is exactly $\mathbf{G}_n^R$. The $\mathbf{A}$ matrix is different, but because it corresponds to a single core, the Gram matrix computation is much cheaper to compute: $\mathbf{G}_n^L = \mathcal{V}(\mathcal{T}_{\mathcal{X},n})^\top \mathcal{V}(\mathcal{T}_{\mathcal{X},n})$.

Thus, we can make a single right-to-left pass to precompute all Gram matrices corresponding to $\mathbf{B}^\top \mathbf{B}$, and then we can make a left-to-right truncation pass where we maintain orthonormality of the left-most cores by passing all singular values to the right factor and compute Gram matrices for $\mathbf{A}^\top \mathbf{A}$ in sequence. The other added benefit of this approach is that the nth core already has one dimension truncated (from the previous mode) when its Gram matrix is computed. This sequence variant is presented in Alg. 6.

We note that the sequence order is arbitrary. Algorithm 6 truncates ranks in left-to-right order, but it can also truncate right-to-left if the Gram matrix sweep is done left-to-right. Following prior work [25], we use the acronym RLR to signify a right-to-left Gram sweep followed by a left-to-right

Algorithm 6 TT-Rounding via Gram SVD (Sequence RLR)

1: **function** $\mathcal{Y} = \text{TT-ROUND-GRAM-SEQ}(\mathcal{X}, \varepsilon)$
2: $\quad \mathbf{G}_{N-1}^R = \mathcal{H}(\mathcal{T}_{\mathcal{X},N})\mathcal{H}(\mathcal{T}_{\mathcal{X},N})^\top$
3: $\quad$ **for** $n = N-1$ down to 1 **do**
4: $\quad\quad \mathcal{V}(\mathcal{T}_{\mathbf{e},n}) = \mathcal{V}(\mathcal{T}_{\mathcal{X},n})\mathbf{G}_n^R$
5: $\quad\quad \mathbf{G}_{n-1}^R = \mathcal{H}(\mathcal{T}_{\mathbf{e},n})\mathcal{H}(\mathcal{T}_{\mathcal{X},n})^\top$
6: $\quad$ **end for**
7: $\quad$ Compute $\|\mathcal{X}\| = (\mathbf{G}_0^R)^{1/2}$ and $\varepsilon_0 = \frac{\|\mathcal{X}\|}{\sqrt{N-1}}\varepsilon$
8: $\quad \mathcal{T}_{\mathcal{Y},1} = \mathcal{T}_{\mathcal{X},1}$
9: $\quad$ **for** $n = 1$ to $N-1$ **do**
10: $\quad\quad \mathbf{G}_n^L = \mathcal{V}(\mathcal{T}_{\mathcal{Y},n})^\top \mathcal{V}(\mathcal{T}_{\mathcal{Y},n})$
11: $\quad\quad [\mathbf{V}_L, \mathbf{\Lambda}_L] = \text{EIG}(\mathbf{G}_n^L)$
12: $\quad\quad [\mathbf{V}_R, \mathbf{\Lambda}_R] = \text{EIG}(\mathbf{G}_n^R)$
13: $\quad\quad [\hat{\mathbf{U}}, \hat{\mathbf{\Sigma}}, \hat{\mathbf{V}}] = \text{TSVD}(\mathbf{\Lambda}_L^{1/2}\mathbf{V}_L^\top \mathbf{V}_R \mathbf{\Lambda}_R^{1/2}, \varepsilon_0)$
14: $\quad\quad \mathcal{V}(\mathcal{T}_{\mathcal{Y},n}) = \mathcal{V}(\mathcal{T}_{\mathcal{Y},n}) \cdot (\mathbf{V}_L \mathbf{\Lambda}_L^{-1/2} \hat{\mathbf{U}})$
15: $\quad\quad \mathcal{H}(\mathcal{T}_{\mathcal{Y},n+1}) = (\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^\top \mathbf{\Lambda}_R^{-1/2} \mathbf{V}_R^\top) \cdot \mathcal{H}(\mathcal{T}_{\mathcal{X},n+1})$
16: $\quad$ **end for**
17: **end function**

truncation sweep, and we use LRL to signify left-to-right Gram sweep followed by a right-to-left truncation sweep.

D. Parallelization

Algorithms 5 and 6 are presented as sequential algorithms. We describe the parallel version of the algorithm in words here, as we have chosen the algorithm for its ease of parallelization. We follow the same parallel distribution as prior work on TT-Rounding via orthogonalization [25] described in §II-D, with each TT core distributed across all processors and each processor owning a subset of the slices in 1D-distribution fashion.

There are two main parallel operations to consider in these algorithms: (1) a TT-core times a small matrix in one mode (e.g., line 4 in Alg. 5), and (2) the contraction of two TT cores across two modes (e.g., line 5 in Alg. 5). Given the parallel distribution, a TT-core times a small matrix in one mode, which is expressed as pre-multiplication of the horizontal unfolding or post-multiplication of the vertical unfolding by a small matrix) can be performed independently, with no communication, if all processors have access to the small matrix. Also, the contraction of two TT cores (expressed as the transpose of a vertical unfolding times another vertical unfolding or a horizontal unfolding times the transpose of another horizontal unfolding) can be performed via parallel reduction with a small matrix as output: after local contraction, a single all-reduce computes and stores the result across all processors.

In the simultaneous variant (Alg. 5), computing the left and right Gram matrices consists of alternating these two operations. Consider line 2 to line 6: if each $\mathbf{G}_n^L$ contraction operation uses an all-reduce, then the subsequent core-times-matrix operation requires only local computation and no communication. The same pattern applies to computing the $\{\mathbf{G}_n^R\}$ matrices. Given that the Gram matrices are all available on all processors, the EVD and SVD operations can be performed redundantly so that the update operations in lines 18 and 19 also require no communication. We note that in the simultaneous variant, the EVD and SVD operations are independent across modes. It is thus possible to distribute these computations across processors, allowing N different processors to work simultaneously on all modes. In this case, the processors need to broadcast their results in order to perform the update operations. This optimization improves scalability at the expense of slightly higher communication costs. We have not implemented this approach because the sequence variant of the algorithm outperforms the simultaneous variant in our experiments.

In the sequence variant (Alg. 6), we pre-compute only one set of Gram matrices. Computing these Gram matrices is parallelized the same as in the simultaneous variant. The unique operation for the sequence variant is line 10, which is a contraction of a TT core with itself, which is performed via local computation and an all-reduce. As before, the EVD and SVD operations are performed redundantly and the updates require no communication.

E. Complexity Analysis

We perform complexity analysis using the simplifying assumptions that all tensor dimensions are equivalent, all ranks are equivalent, and all reduced ranks are equivalent. That is, we assume that $I_n = I$ for $1 \leq n \leq N$ and that original and reduced ranks $R_n = R$ and $L_n = L$ for $1 \leq n \leq N-1$. For comparison, the parallel cost of TT-Rounding via orthogonalization (Alg. 2) is given by

$$\gamma \cdot \left(NIR\frac{3R^2 + 6RL + 4L^2}{P} + O(NR^3 \log P) \right) + \beta \cdot O(NR^2 \log P) + \alpha \cdot O(N \log P),$$

where γ, β, and α are the costs per flop, word, and message, respectively [25, Eq. (3.6)].

Algorithm 5 (the simultaneous variant) performs two passes to compute Gram matrices. For each mode, the local computation involves the multiplication between a local tensor core of dimension $R \times (I/P) \times R$ with an $R \times R$ matrix, for a cost of $2IR^3/P$ flops, and a contraction between two cores, which requires $2IR^3/P$ flops. Thus, the total arithmetic cost of the Gram matrix computations is $8NIR^3/P$. As described in §IV-B, by exploiting symmetry we can reduce the constant factor from 8 to 4. The EVD and SVD operations are performed on $R \times R$ matrices for a total cost of $O(NR^3)$ flops (note there is no parallelism in these operations). The updates of the cores are multiplications of the cores with two $R \times L$ matrices. The first multiplication costs $2IR^2L/P$ flops, while the second costs $2IRL^2/P$ because it involves a core with one mode of already reduced dimension. Thus, the total arithmetic cost for the updates is $2NIR^2L/P + 2NIRL^2/P$.

The communication cost of Alg. 5 is that of two all-reduces for each mode (one for each direction of Gram matrix computation). Thus, the communication costs across all modes

are $\beta \cdot O(NR^2) + \alpha \cdot O(N \log P)$, and the total parallel cost for Alg. 5 (assuming symmetry is exploited) is

$$\gamma \cdot \left(NIR \frac{4R^2 + 2RL + 2L^2}{P} + O(NR^3) \right) + \beta \cdot O(NR^2) + \alpha \cdot O(N \log P).$$

Algorithm 5 (the sequence variant) performs only one pass to compute Gram matrices, for an arithmetic cost of $4NIR^3/P$ flops across all modes, or $2NIR^3/P$ flops if we use the symmetric approach. Computing the Gram matrix for the nth TT core in line 10 costs IR^2L/P flops, because its first mode has already been reduced in dimension from R to L. The EVD and SVD operations and the updates of the cores are the same as in the simultaneous variant. The communications costs are identical to the simultaneous variant as well: there is one all-reduce for each mode in the Gram pass and one all-reduce in each mode for line 10. Thus, the total parallel cost for Alg. 6 (assuming symmetry is exploited) is

$$\gamma \cdot \left(NIR \frac{2R^2 + 3RL + 2L^2}{P} + O(NR^3) \right) + \beta \cdot O(NR^2) + \alpha \cdot O(N \log P).$$

We note that, compared to the orthogonalization approach, the Gram SVD approaches have reduced constants on the leading arithmetic terms and smaller bandwidth terms (by a factor of $O(\log P)$). We will see in the numerical results that the reduced arithmetic provides significant speedup in practice, in part because the performance of the operations (which are all based on `gemm` for Gram SVD) also improves. At higher processor counts, the simplified communication structure (using a single well-optimized collective) also provides speedup over the more complicated communication of Tall-Skinny QR of the orthogonalization approach.

V. Numerical Results

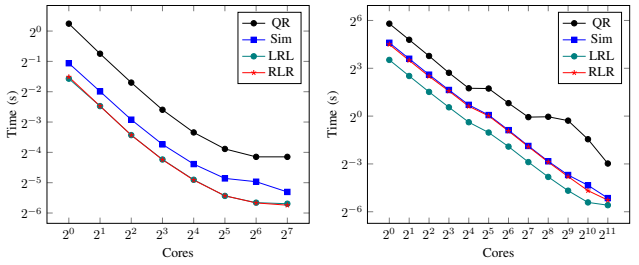
A. Experimental Setup

All parallel scaling experiments are performed on the Andes supercomputer at Oak Ridge Leadership Computing Facility. Andes is a 704-node Linux cluster. Each node contains 256 GB of RAM and 2 AMD EPYC 7302 16-Core processors for a total of 32 cores per node. We build our Gram rounding subroutines on top of the library MPI_ATTAC [37], and we use the OpenBLAS implementation for BLAS and LAPACK routines [38] and OpenMPI [39].

As described in Tab. I, we use 4 synthetic TT models for scaling experiments. Models 1-3 are analogous to the synthetic models used in prior work [25]. Model 4 is identical in shape to the problem we solve via TT-GMRES in the MATLAB implementation of TT-Rounding (see §V-D). We note that these models represent problems arising from parameter dependent PDEs and uncertainty quantification; see [26], [27], [28], [29]. In particular, model 1 mimics the dimensions arising from TT approximations of Gaussian random field correlations [27, §4], and model 4 has dimensions comparable to a previous study of the cookies problem [26, §4]. Models 2 and 3 have larger dimensions, demonstrating the ability to scale TT-Rounding to discretizations that are finer than current sequential libraries are able to process. For each model, we scale using the three Gram SVD algorithms described in §IV-C and the original QR-based TT-Rounding algorithm given by Alg. 2. All reported numbers are the minimum of 5 trials on 5 different allocations. The sequential experiments using MATLAB were performed on a machine with an Intel Xeon Gold 6226R CPU and 256 GB of RAM.

Model	Modes	Dimensions	Memory
1	50	$2K \times \ldots \times 2K$	77 MB
2	16	$100M \times 50K \times \ldots \times 50K \times 1M$	8 GB
3	30	$2M \times \ldots \times 2M$	45 GB
4	10	$10K \times 20 \times \ldots \times 20$	930 KB

TABLE I: Synthetic TT models used for performance experiments. All formal ranks are 20 and are cut in half to 10 by the TT-Rounding procedure.

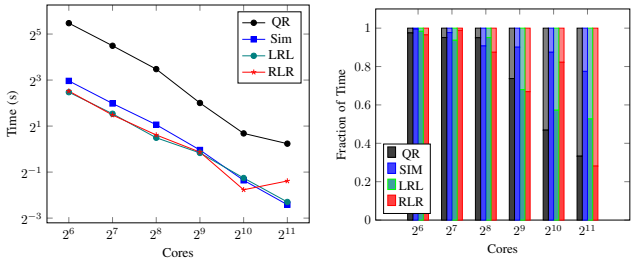


(a) Model 1 (b) Model 2

Fig. 2: Strong scaling results

B. Parallel Scaling of TT-Rounding

Figures 2a, 2b, and 3a present strong scalability comparisons using models 1, 2, and 3, respectively, among different rounding procedures. For the small model 1 problem in Figure 2a, we benchmark from 1 core to all 32 on a single node, and then scale to 4 nodes. We see that the on-node scaling of all algorithms is similar, achieving $14-17\times$ speedup from 1 core to 32 cores, and the sequence variants of the Gram-SVD approach are $3\times$ faster than QR-based rounding on 32 cores. The performance drops off when scaling beyond a single



(a) Strong Scaling (b) Timing Breakdown

Fig. 3: Performance results for Model 3. Dark signifies computation, and light signifies communication.

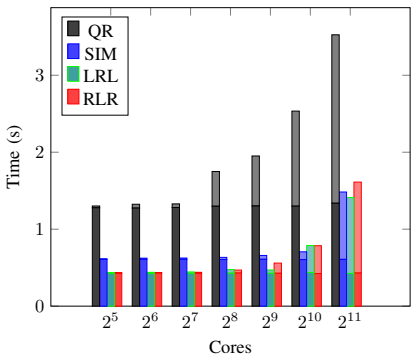
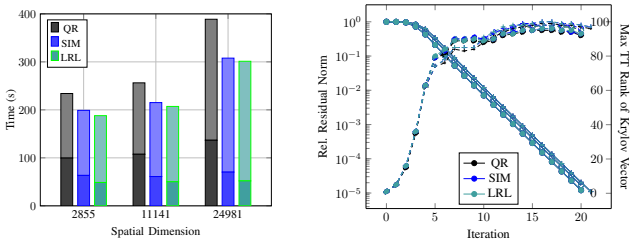


Fig. 4: Weak scaling time breakdowns for Model 1. Dark signifies computation, and light signifies communication.



(a) Timing (b) Residual error and max ranks

Fig. 5: TT-GMRES results for cookies problem with 3 spatial discretizations. In timing plot, dark signifies TT-Rounding, and light signifies other computation. In residual/ranks plot, markers signify different discretizations, solid lines correspond to relative residual norms, and dashed lines correspond to maximum TT rank of the Krylov vector computed at that iteration.

node, as the data footprint of this problem is only 77 MB. In Fig. 2b, we see that Gram-SVD-based rounding methods scale well to 32 nodes (1024 cores), with parallel speedups of $26\times$, $21\times$, and $21\times$ compared to 1 node and $576\times$, $491\times$, and $491\times$ compared to 1 core. The LRL variant is fastest, reaching a speedup of a factor of up to $21\times$ compared to the QR-based rounding. We note that since the mode sizes of the boundary modes are different, the computational complexity costs for the LRL and RLR variants become different, with LRL performing approximately half the flops of RLR. As expected, we see a performance difference between LRL and RLR of nearly $2\times$ when the performance is computation bound, and the run times converge as communication costs begin to dominate. The scalability limit is caused by the machine and is not inherent to the algorithm, as we explain in §V-C.

In the case of model 3, the mode sizes are all equal, and the complexity analysis in §IV-E tells us that the LRL and RLR approaches are about $2\times$ faster than the Gram-Sim approach. This analysis is confirmed by the experiment when the time is computation bound, as we see in Fig. 3a. Speedups of Gram SVD over QR range from $6\times$ to $8\times$, and the parallel speedups for the Gram SVD algorithms on 64 nodes are $42\times$, $27\times$, and $15\times$.

C. Time Breakdown of TT-Rounding

Figure 3b presents the relative communication/computation runtime of the strong scalability test using model 3, matching the data of Fig. 3a. We remark that the communication time is more significant when using the QR-based TT-Rounding. The communication costs for the QR-based are a factor $O(\log P)$ larger than the Gram rounding procedures in theory. Further, the Gram SVD variants use the `MPI_Allreduce` routine which seems to be more efficient than the TSQR implementation used in the QR-based rounding.

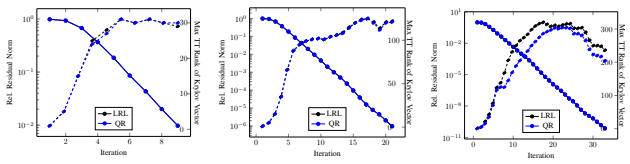
Figure 4 presents the communication/computation runtime breakdown of a weak scalability test using model 1 and different variants of TT-Rounding procedures. We remark that the computation time for each method is the same when increasing the number of processors, and the relative computation time affirms the theoretical analysis of the constant factors on the leading terms. The communication time of Gram rounding procedures shows a logarithmic increase up to 32 nodes (1024 cores) and increases significantly on 64 nodes. This behavior appears even earlier, at 256 processors, when using the QR-based TT-Rounding. In order to understand this behavior, we performed a scalability test on the `MPI_Allreduce` routine on Andes using a single scalar and observed similar behavior costs as in Fig. 4: the time increases like $\log P$ until 32 nodes and then begins to increase more quickly than theory suggests. Thus, we believe the scalability limit is reached due to an artifact of the machine rather than a limitation of the algorithm, whose latency costs should grow with $O(\log P)$.

D. TT-GMRES Performance

Here we consider a parameter dependent PDE model where we seek an increasingly accurate solution by refining the mesh in space. This mesh refinement will increase the size of mode 1 and leave the parameters modes' sizes the same.

1) MATLAB Performance for Small Problem: In this experiment, we use TT-GMRES to solve the cookies problem described in §II-C using $p = 4$ parameters. The values of each parameter are distributed logarithmically in the interval [0.1,10]. The discretization of the PDE is obtained by using FreeFem++ [40]. The refinement is performed by increasing the number of points on $\partial\Omega$ and on the circles defining the disks D_i. The number of points on each circle and each side of the square Ω is selected to be $\{50, 100, 150\}$. This yields 3 spatial discretizations (and corresponding first tensor mode) of dimensions 2855, 11141, and 24981. For each variant of TT-Rounding, we run TT-GMRES with a stopping criterion based on the residual norm being smaller than 10^{-5}. As a preconditioner for TT-GMRES we use the mean preconditioner [26].

Figure 5a shows the performance of the original TT-Rounding using QR in a MATLAB implementation of TT-GMRES compared to the Gram-Sim and Gram-Seq (LRL)



(a) Tol $\varepsilon = 10^{-2}$ (b) Tol $\varepsilon = 10^{-6}$ (c) Tol $\varepsilon = 10^{-10}$

Fig. 6: Accuracy of TT-GMRES using Gram- and QR-based TT-Rounding with varying GMRES tolerance.

implementations of TT-Rounding for the three spatial discretizations. We note that TT-Rounding is at nearly half of the runtime of TT-GMRES using QR and that the Gram-Seq gives at least a $2\times$ speedup over the QR implementation of TT-Rounding for an overall faster TT-GMRES algorithm.

From Figure 5b, we see the variations among relative residual norms obtained by different methods on different discretizations is negligible. We also note that the ranks of the TT representations of the Krylov vectors among the algorithms are also nearly identical, which demonstrates that there is no loss of accuracy in the TT-GMRES method by using the Gram-SVD approaches when the desired residual norm is above square root of machine precision.

2) Accuracy Comparison: To further explore the effects of the Gram-based TT-Rounding on TT-GMRES, we experiment with varying GMRES tolerance values (ε in the notation of Alg. 1). Figure 6 compares the computed residual norms and the maximal TT ranks across the TT-GMRES iterations using QR and Gram-Seq (LRL) for TT-Rounding and three different convergence tolerances: 10^{-2}, 10^{-6}, and 10^{-10}. In this experiment, we set I_1=1781 and $I_k = 10$ for $k = 2, \ldots, 5$.

Our first observation is that the computed residuals are nearly equivalent for all three values of ε, so the use of Gram rounding does not degrade the numerical accuracy of the method even for a tolerance below the square root of machine precision. For the tolerance $\varepsilon = 10^{-10}$, we see an increase in the ranks of the TT representations of each Krylov vector, as depicted by the deviation in maximum ranks in Fig. 6c. Note that the rounding tolerance δ, which is set in line 4 of Alg. 1, increases as the residual decreases, so it is tightest in the earliest iterations. Thus, the Gram method overestimates the ranks in the first several iterations because it does not accurately compute the smallest singular values that can be truncated, and this propagates to later iterations.

While Fig. 6 reports computed residual norms (from line 11 of Alg. 1), we also compute the true residual norms of the solutions. We observe slight deviation between computed and true residual norms but little difference between QR- and Gram-based methods. For $\varepsilon = 10^{-2}$, both approaches achieve true residual norms of $1.1 \cdot 10^{-2}$; for $\varepsilon = 10^{-6}$, both approaches achieve $3.6 \cdot 10^{-6}$; and for $\varepsilon = 10^{-10}$, QR-based rounding achieves $4.0 \cdot 10^{-9}$ and Gram-based rounding achieves $1.2 \cdot 10^{-9}$. We conclude that this deviation is due to the inexactness of the Krylov method rather than the TT-Rounding method, and we attribute the better accuracy of the Gram-based algorithm only to overestimation of the TT ranks.

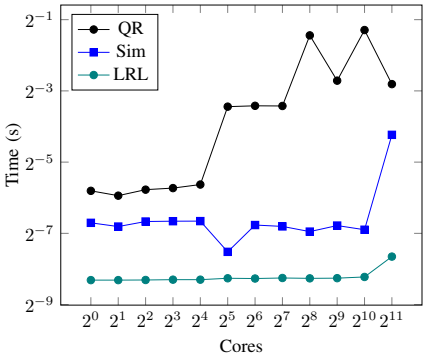


Fig. 7: Weak scaling results for Model 4.

3) Weak Scaling of TT-Rounding for Larger Problems: Using the model 4 tensor of comparable dimensions and ranks to the one used in §V-D1, we weakly scale the spatial dimension on Andes, keeping all other modes fixed, and report the results in Figure 7. We remark that the LRL variant does less computation than RLR, so we report only LRL performance, which we see weakly scales well until 2^{10} cores.

VI. Conclusion

We present in this work a parallel rounding procedure for low-rank TT tensors based on Gram SVD. In contrast with the orthogonalization-based rounding procedure that relies heavily on QR decomposition of tall and skinny matrices, this method relies on matrix multiplication. Not only does the Gram SVD approach reduce the computational complexity, but existing on-node implementations of matrix multiplication are typically more efficient than those for computing and multiplying by orthogonal matrices.

Our scalability experiments show that the proposed method scales as well as or better than the state of the art, in large part because all the communication is cast in terms of all-reduce collectives, and we observe consistent speedup over the previous work on a variety of tensor formats. Our numerical experiments also show that the loss of accuracy inherent in the Gram SVD does not affect the final accuracy of the solution when used in iterative low rank solvers such as TT-GMRES where aggressive truncation, hence low accuracy, can be used.

We consider simultaneous and sequence variants of the Gram SVD approach. The theoretical analysis and experimental results show that the reduced arithmetic of the sequence variants leads to shorter run times in almost all cases. Within the sequence variant, we observe that the LRL and RLR orderings are both possible and typically have comparable run times. We note that for some applications where the first mode size is much larger than the last mode size (which is common for parametrized PDE problems), the LRL approach should be used as it has lower computational complexity.

In the light of the numerical experiments, we plan in the future to study randomized methods to perform rounding procedures. Using randomized methods could outperform the

proposed procedures as they reduce arithmetic further and also rely on matrix multiplication. Encouraged by the results of the MATLAB implementation of TT-GMRES, we also plan to develop a scalable implementation of the TT-based linear solver that can use our parallel TT-Rounding algorithms.

REFERENCES

[1] L. Grasedyck, "Existence and computation of low Kronecker-rank approximations for large linear systems of tensor product structure," *Computing*, vol. 72, no. 3-4, pp. 247–265, 2004. [Online]. Available: https://doi.org/10.1007/s00607-003-0037-z

[2] D. Kressner and C. Tobler, "Krylov subspace methods for linear systems with tensor product structure," *SIAM J. Matrix Anal. Appl.*, vol. 31, no. 4, pp. 1688–1714, 2009/10. [Online]. Available: https://doi.org/10.1137/090756843

[3] W. Dahmen, R. DeVore, L. Grasedyck, and E. Süli, "Tensor-sparsity of solutions to high-dimensional elliptic partial differential equations," *Found. Comput. Math.*, vol. 16, no. 4, pp. 813–874, 2016. [Online]. Available: https://doi.org/10.1007/s10208-015-9265-9

[4] I. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, 2011. [Online]. Available: https://doi.org/10.1137/090752286

[5] Y. Zniyed, R. Boyer, A. F. de Almeida, and G. Favier, "A TT-based hierarchical framework for decomposing high-order tensors," *SIAM J. Sci. Comput.*, vol. 42, no. 2, pp. A822–A848, 2020. [Online]. Available: https://doi.org/10.1137/18M1229973

[6] L. Grigori and S. Kumar, "Parallel Tensor Train through Hierarchical Decomposition," INRIA, Tech. Rep. hal-03081555, 2021. [Online]. Available: https://hal.inria.fr/hal-03081555

[7] M. Röhrig-Zöllner, J. Thies, and A. Basermann, "Performance of low-rank approximations in tensor train format (TT-SVD) for large dense tensors," arXiv, Tech. Rep. 2102.00104, 2021. [Online]. Available: https://arxiv.org/abs/2102.00104

[8] S. V. Dolgov, "TT-GMRES: solution to a linear system in the structured tensor format," *Russian J. Numer. Anal. Math. Modelling*, vol. 28, no. 2, pp. 149–172, 2013. [Online]. Available: https://doi.org/10.1515/rnam-2013-0009

[9] P. Benner, S. Gugercin, and K. Willcox, "A survey of projection-based model reduction methods for parametric dynamical systems," *SIAM Review*, vol. 57, no. 4, pp. 483–531, 2015. [Online]. Available: https://doi.org/10.1137/130932715

[10] P. Benner, S. Dolgov, A. Onwunta, and M. Stoll, "Low-rank solvers for unsteady Stokes–Brinkman optimal control problem with random data," *Comput. Methods Appl. Mech. Engrg.*, vol. 304, pp. 26–54, 2016. [Online]. Available: https://doi.org/10.1016/j.cma.2016.02.004

[11] ——, "Low-rank solution of an optimal control problem constrained by random Navier-Stokes equations," *Internat. J. Numer. Methods Fluids*, vol. 92, no. 11, pp. 1653–1678, 2020. [Online]. Available: https://doi.org/10.1002/fld.4843

[12] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "SPLATT: Efficient and parallel sparse tensor-matrix multiplication," in *IPDPS '15*. IEEE Computer Society, 2015, pp. 61–70. [Online]. Available: http://doi.org/10.1109/IPDPS.2015.27

[13] S. Eswar, K. Hayashi, G. Ballard, R. Kannan, M. A. Matheson, and H. Park, "PLANC: Parallel low rank approximation with non-negativity constraints," arXiv, Tech. Rep. 1909.01149, 2019. [Online]. Available: https://arxiv.org/abs/1909.01149

[14] 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 Trans. Math. Software*, vol. 46, no. 2, 2020. [Online]. Available: https://doi.org/10.1145/3378445

[15] V. T. Chakaravarthy, J. W. Choi, D. J. Joseph, X. Liu, P. Murali, Y. Sabharwal, and D. Sreedhar, "On optimizing distributed Tucker decomposition for dense tensors," in *IPDPS '17*, 2017, pp. 1038–1047. [Online]. Available: https://doi.org/10.1109/IPDPS.2017.86

[16] O. Kaya and B. Uçar, "Scalable sparse tensor decompositions in distributed memory systems," in *SC '15*. ACM, 2015, pp. 77:1–77:11. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807624

[17] J. Choi, X. Liu, and V. Chakaravarthy, "High-performance dense tucker decomposition on GPU clusters," in *SC '18*. IEEE Press, 2018, pp. 42:1–42:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=3291656.3291712

[18] E. Solomonik, D. Matthews, J. R. Hammond, J. F. Stanton, and J. Demmel, "A massively parallel tensor contraction framework for coupled-cluster computations," *J. Parallel Distrib. Comput.*, vol. 74, no. 12, pp. 3176–3190, 2014. [Online]. Available: https://doi.org/10.1016/j.jpdc.2014.06.002

[19] E. Aprà, E. J. Bylaska *et al.*, "NWChem: Past, present, and future," *J. Chem. Phys.*, vol. 152, no. 18, p. 184102, 2020. [Online]. Available: https://doi.org/10.1063/5.0004997

[20] E. Stoudenmire and S. R. White, "ITensor: A C++ library for creating efficient and flexible physics simulations based on tensor product wavefunctions," 2016. [Online]. Available: http://itensor.org/

[21] I. Oseledets *et al.*, "TT-Toolbox." [Online]. Available: https://github.com/oseledets/TT-Toolbox

[22] A. Novikov, P. Izmailov, V. Khrulkov, M. Figurnov, and I. V. Oseledets, "Tensor Train decomposition on TensorFlow (T3F)," *J. Mach. Learn. Res.*, vol. 21, no. 30, pp. 1–7, 2020. [Online]. Available: https://www.jmlr.org/papers/v21/18-008.html

[23] S. Dolgov and D. Savostyanov, "Parallel cross interpolation for high-precision calculation of high-dimensional integrals," *Comput. Phys. Commun.*, vol. 246, p. 106869, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465519302565

[24] L. Grigori and S. Kumar, "Parallel tensor train through hierarchical decomposition," INRIA, Tech. Rep. hal-03081555, 2021. [Online]. Available: https://hal.inria.fr/hal-03081555

[25] H. A. Daas, G. Ballard, and P. Benner, "Parallel algorithms for tensor train arithmetic," *SIAM J. Sci. Comput.*, vol. 44, no. 1, pp. C25–C53, 2022. [Online]. Available: https://doi.org/10.1137/20M1387158

[26] D. Kressner and C. Tobler, "Low-rank tensor Krylov subspace methods for parametrized linear systems," *SIAM J. Matrix Anal. Appl.*, vol. 32, no. 4, pp. 1288–1316, 2011. [Online]. Available: https://doi.org/10.1137/100799010

[27] D. Kressner, R. Kumar, F. Nobile, and C. Tobler, "Low-rank tensor approximation for high-order correlation functions of Gaussian random fields," *SIAM/ASA J. Uncertain. Quantif.*, vol. 3, no. 1, pp. 393–416, 2015. [Online]. Available: https://doi.org/10.1137/140968938

[28] F. Bonizzoni, F. Nobile, and D. Kressner, "Tensor train approximation of moment equations for elliptic equations with lognormal coefficient," *Comput. Methods Appl. Mech. Engrg.*, vol. 308, pp. 349–376, 2016. [Online]. Available: https://doi.org/10.1016/j.cma.2016.05.026

[29] P. Benner, A. Onwunta, and M. Stoll, "Block-diagonal preconditioning for optimal control problems constrained by PDEs with uncertain inputs," *SIAM J. Matrix Anal. Appl.*, vol. 37, no. 2, pp. 491–518, 2016. [Online]. Available: https://doi.org/10.1137/15M1018502

[30] A.-H. Phan, P. Tichavsky, and A. Cichocki, "Fast alternating LS algorithms for high order CANDECOMP/PARAFAC tensor factorizations," *IEEE Trans. Signal Process.*, vol. 61, no. 19, pp. 4834–4846, 2013. [Online]. Available: https://doi.org/10.1109/TSP.2013.2269903

[31] L. Trefethen and D. Bau, *Numerical Linear Algebra*. SIAM, 1997.

[32] C. Tobler, "Low-rank tensor methods for linear systems and eigenvalue problems," Ph.D. dissertation, ETH Zurich, 2012. [Online]. Available: http://sma.epfl.ch/~anchpcommon/students/tobler.pdf

[33] J. Ballani and L. Grasedyck, "A projection method to solve linear systems in tensor format," *Numer. Linear Algebra Appl.*, vol. 20, no. 1, pp. 27–43, 2013. [Online]. Available: https://doi.org/10.1002/nla.1818

[34] R. Weinhandl, P. Benner, and T. Richter, "Low-rank linear fluid-structure interaction discretizations," *ZAMM Z. Angew. Math. Mech.*, vol. 100, no. 11, p. e201900205, 2020. [Online]. Available: https://doi.org/10.1002/zamm.201900205

[35] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, "Communication-optimal parallel and sequential QR and LU factorizations," *SIAM J. Sci. Comput.*, vol. 34, no. 1, pp. A206–A239, 2012. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/080731992

[36] R. Penrose, "Applications of negative dimensional tensors," in *Combinatorial mathematics and its applications*, 1971, pp. 221–244.

[37] H. A. Daas *et al.*, "MPI_ATTAC." [Online]. Available: https://gitlab.com/aldaas/mpi_attac

[38] Z. Xianyi *et al.*, "OpenBLAS." [Online]. Available: https://github.com/xianyi/OpenBLAS

[39] E. Gabriel, G. E. Fagg *et al.*, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *EuroMPI '04*, 2004, pp. 97–104. [Online]. Available: https://doi.org/10.1007/978-3-540-30218-6_19

[40] F. Hecht, "New development in FreeFem++," *J. Numer. Math.*, vol. 20, no. 3-4, pp. 251–265, 2012. [Online]. Available: https://freefem.org/