Accuracy vs. Cost in Parallel Fixed-Precision Low-Rank Approximations of Sparse Matrices

Robert Ernstbrunner
University of Vienna
Faculty of Computer Science
Vienna, Austria
robert.ernstbrunner@univie.ac.at

Viktoria Mayer
University of Vienna
Faculty of Computer Science
Vienna, Austria
viktoria.mayer@univie.ac.at

Wilfried Gansterer
University of Vienna
Faculty of Computer Science
Vienna, Austria
wilfried.gansterer@univie.ac.at

2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 978-1-6654-8106-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/IPDPS53621.2022.00051

***Abstract*—We study a randomized and a deterministic algorithm for the fixed-precision low-rank approximation problem of large sparse matrices. The Randomized QB Factorization (RandQB_EI) constructs a reduced and dense representation of the originally sparse matrix based on randomization. The representation resulting from the deterministic Truncated LU Factorization with Column and Row Tournament Pivoting (LU_CRTP) is sparse, but fill-in introduced in the factorization process can affect sparsity and performance. We therefore attempt to mitigate fill-in with an incomplete LU_CRTP variant with thresholding (ILUT_CRTP). We analyze this approach and identify potential problems that may arise in practice. We design parallel implementations of RandQB_EI, LU_CRTP and ILUT_CRTP. We experimentally evaluate strong scaling properties for different problems and the runtime required for achieving a given approximation quality. Our results show that LU_CRTP tends to be particularly competitive for low approximation quality. However, when a lot of fill-in occurs, LU_CRTP is outperformed by RandQB_EI especially for higher approximation quality. ILUT_CRTP outperforms both LU_CRTP and RandQB_EI and can achieve speedups up to 40 over LU_CRTP, depending on the amount of fill-in.**

***Index Terms*—Randomized low-rank approximations, Deterministic low-rank approximations, Randomized QB Factorization, Truncated LU Factorization with Tournament Pivoting.**

I. Introduction

Consider a large sparse input matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ with numerical rank $l = \min(m, n)$. The Eckart-Young Theorem states that the truncated Singular Value Decomposition (SVD) minimizes $||\mathbf{A} - \hat{\mathbf{A}}_k||$ in both spectral and Frobenius norm, where $\hat{\mathbf{A}}_k$ is any rank-k matrix. However, the computation of the truncated SVD is often prohibitively expensive in terms of runtime and memory. Several algorithms have emerged that produce competitive results at lower computational cost.

Low-rank matrix approximations are important in a broad range of problem domains, see, for example, [15]. *Fixed-rank* algorithms are used for problems where the rank of the approximation is known *a priori*. *Fixed-precision* problems on the other hand determine a rank $K \ll l$ such that

$$\min_{K=\text{rank}(\hat{\mathbf{A}}_K)} \left\| \mathbf{A} - \hat{\mathbf{A}}_K \right\| < \tau \left\| \mathbf{A} \right\| \tag{1}$$

Supported partially by the Vienna Science and Technology Fund (WWTF) through project ICT15-113.

for a tolerance τ given by the user. Methods addressing this problem are known as *fixed-precision* algorithms.

We surveyed fixed-precision methods and found that, while the literature on this topic is extensive and rich, the set of algorithms that are suitable for large sparse matrices is rather small. We consider and compare two representative fixed-precision methods which are based on fundamentally different concepts, the *randomized* QB Factorization (RandQB_EI) [20] and the *deterministic* Truncated LU Factorization with Column and Row Tournament Pivoting (LU_CRTP) [10].

A. Related Work

Bach et al. [1] state that "the majority of research and software implementations of randomized low-rank approximation methods have so far focused on the fixed-rank problem". We focus on fixed-precision methods, where the rank is determined by the algorithm. In particular, we are interested in algorithms that can handle large sparse matrices.

Halko et al. [12] give a comprehensive overview of randomized algorithms for computing low-rank approximations. The Randomized Range Finder (RRF) constitutes the basic idea of probabilistic fixed-rank algorithms. The method uses *randomized sketching* [12] to derive a small basis from the large input matrix $\mathbf{A}$, such that this basis captures the essential information of $\mathbf{A}$. The Adaptive Randomized Range Finder (ARRF) uses the same concept to incrementally and adaptively construct a low-rank approximation that satisfies (1). Martinsson et al. [16] propose a blocked version of ARRF called RandQB_b, whose stopping criterion is more precise than the estimator suggested in [12]. However, the input matrix is updated with a dense matrix in each iteration and the algorithm is therefore not suitable for sparse matrices. Bach et al. [1] introduce a randomized fixed-precision algorithm based on ARRF. Like RandQB_b, this algorithm is not suited for sparse problems. The randomized SVD (RSVD) [12] is another concept derived from RRF. The algorithm computes a randomized SVD with an initial estimated rank k. If the error of the approximation is too large, another RSVD with a larger k is computed. This is continued until the error is small enough. RandUBV has been presented very recently in [13]. The method is based on block Lanzcos bidiagonalization and is closely related to RandQB_EI. Numerical experiments

1530-2075/22/$31.00 ©2022 IEEE
DOI 10.1109/IPDPS53621.2022.00051

suggest that RandUBV is competitive to RandQB_EI in terms of runtime and approximation quality.

LU_CRTP, a deterministic alternative to the truncated SVD, was first presented in [10] and enhanced in [2] with, inter alia, a novel strategy that discards some columns of $\mathbf{A}$ to reduce computational work. Grigori et al. [10] show that in theory and under certain conditions, the fixed-rank version of LU_CRTP outperforms the randomized algorithm presented in [3], one of the fastest fixed-rank algorithms known at the time. It is still an open question whether this holds in practice for the fixed-precision variant of LU_CRTP and RandQB_EI.

Techniques for finding low-rank approximations have also been investigated for *data-sparse* hierarchical matrices [11, 17]. We focus on algorithms for general sparse matrices.

B. Contributions

Based on the parallel implementation of LU_CRTP for sparse fixed-rank problems by Grigori et al. [10], we develop a parallel version of LU_CRTP for sparse *fixed-precision* problems. We also develop a parallel implementation of RandQB_EI and compare the two parallel algorithms in terms of number of iterations, runtime, accuracy and parallel scalability. Uniform termination criteria are required to establish a meaningful comparison between the methods. Moreover, we include an analysis of the asymptotic arithmetic complexity and runtime performance of the most expensive kernels and present results with a sequential implementation of RandUBV [13].

We illustrate that LU_CRTP tends to produce fill-in that can significantly slow down the algorithm. To counter this problem, we adapt LU_CRTP by dropping small enough entries. We call this new version of the algorithm *Incomplete* LU_CRTP with Thresholding (ILUT_CRTP). We analyze our approach and identify potential limitations that may arise in practice. Numerical experiments illustrate that our adaptation can significantly improve runtime performance and lower the number of non-zeros in the factors produced.

C. Synopsis

In Section II we review the randomized algorithm RandQB_EI and the deterministic algorithm LU_CRTP. Section III introduces ILUT_CRTP, an adaption of LU_CRTP which reduces the fill-in. In Section IV we analyze the asymptotic arithmetic complexities for RandQB_EI, RandUBV and LU_CRTP. In Section V we discuss the parallelization of RandQB_EI and LU_CRTP. We summarize numerical experiments in Section VI, and we conclude in Section VII.

II. Fixed-precision low-rank approximation

Using Matlab-like notation, we refer to the submatrix of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ formed by rows i to j and columns v to w as $\mathbf{A}(i:j, v:w)$. Indices start with 1. If not explicitly defined, $\|\cdot\|$ denotes either the Frobenius or spectral norm.

First, we summarize and highlight important aspects of RandQB_EI and LU_CRTP. Both methods aim to satisfy (1). RandQB_EI iteratively computes a condensed subspace for the range of the input matrix $\mathbf{A}$. LU_CRTP iteratively produces truncated block LU factors based on the Rank Revealing QR factorization with Tournament Pivoting (QR_TP). QR_TP is briefly addressed in this Section and in Section V. A more detailed explanation can be found in [10] and the references therein. The algorithmic backgrounds of RandQB_EI and LU_CRTP are fundamentally different, but the individual approximation-generating processes can be generalized to illustrate the basic concept of these algorithms. In the first iteration, both methods produce approximation matrices $\mathbf{H}_k^{(1)} \in \mathbb{R}^{m \times k}$ and $\mathbf{W}_k^{(1)} \in \mathbb{R}^{k \times n}$ for $\mathbf{A}$, such that

$$\mathbf{H}_k^{(1)} \mathbf{W}_k^{(1)} = \hat{\mathbf{A}}_k \approx \mathbf{A}. \tag{2}$$

If (2) does not satisfy (1) then in successive iterations i, the matrices $\mathbf{H}_k^{(i)}$ and $\mathbf{W}_k^{(i)}$ are generated and concatenated to yield the matrices $\mathbf{H}_K$ and $\mathbf{W}_K$, i.e.,

$$\mathbf{H}_K = [\mathbf{H}_k^{(1)}, \ldots, \mathbf{H}_k^{(i)}], \quad \mathbf{W}_K = [\mathbf{W}_k^{T(1)}, \ldots, \mathbf{W}_k^{T(i)}]^T,$$

with overestimated rank $K = ik$ and resulting approximation

$$\mathbf{H}_K \mathbf{W}_K = \hat{\mathbf{A}}_K \approx \mathbf{A}.$$

If RandQB_EI is used as the underlying algorithm, $\mathbf{H}_K$ and $\mathbf{W}_K$ are inherently *dense*. As a consequence, if $\mathbf{A}$ has, e.g., only up to kl non-zeros, the amount of non-zeros in $\mathbf{H}_k^{(1)}$ and $\mathbf{W}_k^{(1)}$ already surpasses the amount of non-zeros in $\mathbf{A}$. If the underlying algorithm is LU_CRTP, $\mathbf{H}_K$ and $\mathbf{W}_K$ are potentially *sparse*. However, the sparsity of $\mathbf{H}_K$ and $\mathbf{W}_K$ depends on the properties of $\mathbf{A}$ and, in the worst case, these matrices might be dense as well. Moreover, factors with higher density not only require more memory, but, in case of LU_CRTP, are also produced at higher computational cost. We therefore adapt LU_CRTP later on such that the number of non-zeros in $\mathbf{H}_K$ and $\mathbf{W}_K$ are potentially reduced and the overall factorization process is likely to be less costly.

A. Randomized QB Factorization

RandQB_EI [20] (see Algorithm 1) is based on RandQB_b [16] and computes an approximate basis matrix $\mathbf{Q}_K \in \mathbb{R}^{m \times K}$ for the range of the input matrix $\mathbf{A}$, and a matrix $\mathbf{B}_K \in \mathbb{R}^{K \times n}$ such that $\mathbf{Q}_K \mathbf{B}_K \approx \mathbf{A}$. The matrices $\mathbf{Q}_K$ and $\mathbf{B}_K$ are constructed incrementally, where $\mathbf{B}_K = \mathbf{Q}_K^T \mathbf{A}$ can be used to compute an approximate factorization of $\mathbf{A}$. The resulting approximation error is given as

$$e_{rand} = \|\mathbf{A} - \mathbf{Q}_K \mathbf{B}_K\|. \tag{3}$$

1) Algorithm: In each iteration, a random matrix $\Omega_k \in \mathbb{R}^{n \times k}$ is constructed in line 4 of Algorithm 1, where k is the block size. Ω_k is used to compute a new block $\mathbf{Q}_k$ of $\mathbf{Q}_K$ in line 5, which is then reorthogonalized in line 10 against the previously computed parts of $\mathbf{Q}_K$ to reduce round-off errors. New rows of $\mathbf{B}_K$ are computed in line 11. To improve approximation quality, the algorithm can work on a similar matrix $\mathbf{K} = (\mathbf{A}\mathbf{A}^T)^p \mathbf{A}$ with the same singular vectors as $\mathbf{A}$ and related singular values $\sigma_j(\mathbf{K}) = \sigma_j(\mathbf{A})^{2p+1}$, with $1 \leq j \leq l$ and $0 \leq p \leq 3$. This modification exponentially accelerates singular value decay and is known as the power

scheme [12] (lines 6-9). Finally, in line 12 the matrices $\mathbf{Q}_K$ and $\mathbf{B}_K$ are expanded with $\mathbf{Q}_k$ and $\mathbf{B}_k$.

2) Termination Criterion: The Frobenius norm of (3) can be computed at small cost for the so far computed approximation in each iteration of RandQB_EI. [20] show that $\|\mathbf{A} - \mathbf{Q}_K\mathbf{B}_K\|_F^2 = \|\mathbf{A}\|_F^2 - \|\mathbf{B}_K\|_F^2$ for any factorization of the form $\mathbf{Q}_K\mathbf{B}_K \approx \mathbf{A}$, given that $\mathbf{Q}_K$ is orthonormal. Since $\|\mathbf{B}_K\|_F^2 = \sum_{j=1}^{i} \left\|\mathbf{B}_k^{(j)}\right\|_F^2$, with $\mathbf{B}_k^{(j)}$ being the blocks of $\mathbf{B}_K$ computed in the first i iterations, we can indicate the error with an update to $\|\mathbf{A}\|_F^2$ by subtracting $\left\|\mathbf{B}_k^{(i)}\right\|_F^2$ in each iteration i. The *error indicator* $E^{(i)}$ for RandQB_EI is defined as

$$E_{rand}^{(i)} := \sqrt{\|\mathbf{A}\|_F^2 - \sum_{j=1}^{i} \left\|\mathbf{B}_k^{(j)}\right\|_F^2}. \tag{4}$$

The method converges if $E_{rand}^{(i)} < \tau \|\mathbf{A}\|_F$ (line 14 of Algorithm 1). Theorem 3 of [20] shows that (4) fails in double precision floating point arithmetic for $\tau < 2.1 \cdot 10^{-7}$.

Algorithm 1 RandQB_EI

Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, block size k, power parameter $p \geq 0$, tolerance τ
Output: $\mathbf{Q}_K \in \mathbb{R}^{m\times K}$, $\mathbf{B}_K \in \mathbb{R}^{K\times n}$, rank K, such that $\|\mathbf{A} - \mathbf{Q}_K\mathbf{B}_K\|_F < \tau \|\mathbf{A}\|_F$

1: $\mathbf{Q}_K = [\,], \mathbf{B}_K = [\,], E = \|\mathbf{A}\|_F^2$
2: **for** $i = 1, 2, \ldots$ **do**
3: $\quad j = (i-1)k + 1, \quad K = ik$
4: $\quad \Omega_k = \text{randn}(n, k)$
5: $\quad \mathbf{Q}_k = \text{orth}(\mathbf{A}\Omega_k - \mathbf{Q}_K(\mathbf{B}_K\Omega_k))$
6: $\quad$ **for** $r = 1 : p$ **do** ▷ power scheme
7: $\quad\quad \hat{\mathbf{Q}}_k = \text{orth}(\mathbf{A}^T\mathbf{Q}_k - \mathbf{B}_K^T(\mathbf{Q}_K^T\mathbf{Q}_k))$
8: $\quad\quad \mathbf{Q}_k = \text{orth}(\mathbf{A}\hat{\mathbf{Q}}_k - \mathbf{Q}_K(\mathbf{B}_K\hat{\mathbf{Q}}_k))$
9: $\quad$ **end for**
10: $\quad \mathbf{Q}_k = \text{orth}(\mathbf{Q}_k - \mathbf{Q}_K(\mathbf{Q}_K^T\mathbf{Q}_k))$ ▷ re-orthogonalization
11: $\quad \mathbf{B}_k = \mathbf{Q}_k^T\mathbf{A}$
12: $\quad \mathbf{Q}_K(:, j:K) = \mathbf{Q}_k, \quad \mathbf{B}_K(j:K, :) = \mathbf{B}_k$
13: $\quad E = E - \|\mathbf{B}_k\|_F^2$
14: $\quad$ **if** $\sqrt{E} < \tau \|\mathbf{A}\|_F$ **then stop**
15: **end for**

B. Truncated LU Factorization with Tournament Pivoting

LU_CRTP [10] (see Algorithm 2) performs a block LU factorization with column and row pivoting and computes truncated LU factors $\mathbf{L}_K \in \mathbb{R}^{m\times K}$ and $\mathbf{U}_K \in \mathbb{R}^{K\times n}$ and permutation matrices $\mathbf{P}_r \in \mathbb{R}^{m\times m}$ and $\mathbf{P}_c \in \mathbb{R}^{n\times n}$ such that $\mathbf{L}_K\ \mathbf{U}_K \approx \mathbf{P}_r\ \mathbf{A}\ \mathbf{P}_c$ with resulting approximation error

$$e_{det} = \|\mathbf{P}_r\mathbf{A}\mathbf{P}_c - \mathbf{L}_K\mathbf{U}_K\|. \tag{5}$$

1) Algorithm: In each iteration i, the approximation

$$\mathbf{L}_k\mathbf{U}_k \approx \bar{\mathbf{A}} = \mathbf{P}_r^{(i)}\mathbf{A}^{(i)}\mathbf{P}_c^{(i)}, \tag{6}$$

is computed, where $\mathbf{P}_r^{(i)} \in \mathbb{R}^{(m-z)\times(m-z)}$ and $\mathbf{P}_c^{(i)} \in \mathbb{R}^{(n-z)\times(n-z)}$ are permutation matrices, with $z = (i-1)k$, and $\mathbf{A}^{(1)} = \mathbf{A}$. $\mathbf{L}_k \in \mathbb{R}^{(m-z)\times k}$ and $\mathbf{U}_k \in \mathbb{R}^{k\times(n-z)}$ are truncated factors from the factorization

$$\bar{\mathbf{A}} = \begin{pmatrix} \bar{\mathbf{A}}_{11} & \bar{\mathbf{A}}_{12} \\ \bar{\mathbf{A}}_{21} & \bar{\mathbf{A}}_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \\ \bar{\mathbf{A}}_{21}\bar{\mathbf{A}}_{11}^{-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \bar{\mathbf{A}}_{11} & \bar{\mathbf{A}}_{12} \\ & S(\bar{\mathbf{A}}_{11}) \end{pmatrix}$$
$$\approx \begin{pmatrix} \mathbf{I} \\ \bar{\mathbf{A}}_{21}\bar{\mathbf{A}}_{11}^{-1} \end{pmatrix} \begin{pmatrix} \bar{\mathbf{A}}_{11} & \bar{\mathbf{A}}_{12} \end{pmatrix} = \mathbf{L}_k\mathbf{U}_k,$$

where

$$S(\bar{\mathbf{A}}_{11}) = \bar{\mathbf{A}}_{22} - \bar{\mathbf{A}}_{21}\bar{\mathbf{A}}_{11}^{-1}\bar{\mathbf{A}}_{12} \tag{7}$$

denotes the Schur complement of $\bar{\mathbf{A}}_{11} \in \mathbb{R}^{k\times k}$. This factorization has the property that $\bar{\mathbf{A}}_{11}$ reveals the k largest singular values of $\mathbf{A}^{(i)}$ and $S(\bar{\mathbf{A}}_{11})$ reveals the $l-K$ smallest singular values of $\mathbf{A}^{(i)}$ due to the fact that the permutation matrices $\mathbf{P}_r^{(i)}$ and $\mathbf{P}_c^{(i)}$ in (6) emerged from a rank revealing QR factorization. QR_TP is a rank revealing block QR factorization algorithm that finds the k "most linearly independent" columns of the input matrix and permutes them to the k leftmost positions. $\mathbf{P}_c^{(i)}$ is obtained from QR_TP in line 5 of Algorithm 2. The factorization has the form

$$\mathbf{A}^{(i)}\mathbf{P}_c^{(i)} = \begin{pmatrix} \mathbf{Q}_{11} & \mathbf{Q}_{12} \\ \mathbf{Q}_{21} & \mathbf{Q}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ & \mathbf{R}_{22} \end{pmatrix}, \tag{8}$$

where the upper triangular $\mathbf{R}_{11} \in \mathbb{R}^{k\times k}$ reveals the $K-k+1$ to K largest and $\mathbf{R}_{22} \in \mathbb{R}^{(m-K)\times(n-K)}$ the $l-K$ smallest singular values of $\mathbf{A}$. In practice, only $\mathbf{P}_c^{(i)}$, $\mathbf{R}_{11}$, $\mathbf{Q}_{11}$ and $\mathbf{Q}_{21}$ are computed explicitly. $\left[\mathbf{Q}_{11}^T, \mathbf{Q}_{21}^T\right]^T$ resp. $\mathbf{R}_{11}$ in (8) corresponds to $\mathbf{Q}_k$ resp. $\mathbf{R}^{(i)}$ in line 6 of Algorithm 2. The QR_TP factorization on $\left[\mathbf{Q}_{11}^T, \mathbf{Q}_{21}^T\right]^T$ in line 7 yields $\mathbf{P}_r^{(i)}$. $\bar{\mathbf{Q}}_{11}$ from $\mathbf{P}_r^{(i)}\mathbf{Q}_k = \left[\bar{\mathbf{Q}}_{11}^T, \bar{\mathbf{Q}}_{21}^T\right]^T$ ensures through the relation $\bar{\mathbf{A}}_{11} = \bar{\mathbf{Q}}_{11}\mathbf{R}_{11}$ that the singular values of $\bar{\mathbf{A}}_{11}$ stay close to the singular values of $\mathbf{R}_{11}$. The method sets $\mathbf{A}^{(i+1)} := S(\bar{\mathbf{A}}_{11})$ in line 12 and recurses on $\mathbf{A}^{(i+1)}$ in successive iterations. The truncated factors $\mathbf{L}_k$ and $\mathbf{U}_k$ are concatenated in line 11 to incrementally build the factors $\mathbf{L}_K$ and $\mathbf{U}_K$.

2) Termination Criterion: Grigori et al. [10] terminate LU_CRTP if $\left|\mathbf{R}^{(i)}(k,k)\right|$ is smaller than a tolerance, which does not guarantee that (1) is satisfied. Similar to the error indicator (4), (5) can be computed efficiently as long as $\mathbf{A}^{(i+1)}$ is sparse. Since $\|\mathbf{P}_r\mathbf{A}\mathbf{P}_c - \mathbf{L}_K\mathbf{U}_K\| = \left\|\mathbf{A}^{(i+1)}\right\|$, the error indicator for LU_CRTP is given as

$$E_{det}^{(i)} := \left\|\mathbf{A}^{(i+1)}\right\|_F. \tag{9}$$

In contrast to (4), (9) also works in double precision for $\tau < 2.1 \cdot 10^{-7}$. We terminate LU_CRTP in line 13 of Algorithm 2 if $E_{det}^{(i)} < \tau \|\mathbf{A}\|_F$. This allows for a fair comparison to RandQB_EI. Note that evaluating (9) in the last iteration includes line 12 of Algorithm 2. The termination criterion in [10] does not require this additional computation.

3) The fill-in problem: The formation of (7) potentially produces fill-in that increases computational cost in successive iterations and also affects the sparsity of $\mathbf{L}_K$ and $\mathbf{U}_K$. In the worst case, $\mathbf{A}^{(2)}$ is already dense. In the best case, the matrices $\mathbf{A}^{(i+1)}$ remain sparse throughout the factorization process.

In a sparse QR factorization without pivoting, column reorderings can be applied to maintain sparsity. Row permutations, on the other hand, have no effect [10]. For the classical QR factorization with column pivoting, *a priori* column reorderings are meaningless due to value-dependent pivoting in the factorization process. This is not the case for QR_TP for which fill-reducing column reordering algorithms can be used. Therefore, Grigori et al. [10] first permute the input matrix with the column approximate minimum degree algorithm COLAMD [4]. $\mathbf{L}_K$ can also be computed with a method, that benefits numerical stability, but introduces additional small values (see [10]). If this computation is required, it further contributes to the amount of fill-in introduced.

Algorithm 2 LU_CRTP

Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$, block size k, tolerance τ
Output: $\mathbf{L}_K \in \mathbb{R}^{m \times K}, \mathbf{U}_K \in \mathbb{R}^{K \times n}, \mathbf{P}_r \in \mathbb{R}^{m \times m}, \mathbf{P}_c \in \mathbb{R}^{n \times n}$, rank K, s.t. $\|\mathbf{P}_r \mathbf{A} \mathbf{P}_c - \mathbf{L}_K \mathbf{U}_K\|_F < \tau \|\mathbf{A}\|_F$

$\mathbf{L}_K = [\,], \mathbf{U}_K = [\,], \mathbf{P}_r = \mathbf{I} \in \mathbb{R}^{m \times m}, \mathbf{P}_c = \mathbf{I} \in \mathbb{R}^{n \times n}$
$\mathbf{A}^{(1)} = \mathbf{A}$
for $i = 1, 2, \ldots$ **do**
  $j = (i-1)k + 1, \quad K = ik$
  $\mathbf{P}_c^{(i)} = \text{QR\_TP}(\mathbf{A}^{(i)}, k)$
  $[\mathbf{Q}_k, \mathbf{R}^{(i)}] = \text{qr}((\mathbf{A}^{(i)} \mathbf{P}_c^{(i)})(:, 1:k))$
  $\mathbf{P}_r^{(i)} = \text{QR\_TP}(\mathbf{Q}_k^T, k)^T$
  $\bar{\mathbf{A}} = \mathbf{P}_r^{(i)} \mathbf{A}^{(i)} \mathbf{P}_c^{(i)} = \begin{pmatrix} \bar{\mathbf{A}}_{11} & \bar{\mathbf{A}}_{12} \\ \bar{\mathbf{A}}_{21} & \bar{\mathbf{A}}_{22} \end{pmatrix}$
  Update $\mathbf{P}_r$, $\mathbf{P}_c$, $\mathbf{L}_K$ and $\mathbf{U}_K$ using $\mathbf{P}_r^{(i)}$ and $\mathbf{P}_c^{(i)}$.
  $\mathbf{L}_k = \begin{pmatrix} \mathbf{I} \\ \bar{\mathbf{A}}_{21} \bar{\mathbf{A}}_{11}^{-1} \end{pmatrix}, \qquad \mathbf{U}_k = \begin{pmatrix} \bar{\mathbf{A}}_{11} & \bar{\mathbf{A}}_{12} \end{pmatrix}$
  $\mathbf{L}_K(j:m, j:K) = \mathbf{L}_k, \quad \mathbf{U}_K(j:K, j:n) = \mathbf{U}_k$
  $\mathbf{A}^{(i+1)} = \bar{\mathbf{A}}_{22} - \bar{\mathbf{A}}_{21} \bar{\mathbf{A}}_{11}^{-1} \bar{\mathbf{A}}_{12}$ ▷ $S(\bar{\mathbf{A}}_{11})$
  if $\|\mathbf{A}^{(i+1)}\|_F < \tau \|\mathbf{A}\|_F$ **then stop**
end for

III. Incomplete LU_CRTP with Thresholding

As mentioned in the previous section, LU_CRTP potentially produces fill-in in the matrices $\mathbf{A}^{(i+1)}$. Our numerical experiments show that performance can be significantly reduced due to fill-in. We may resolve this problem by dropping small enough elements in $\mathbf{A}^{(i+1)}$ based on a threshold μ. This adaptation leads to better runtime performance and also reduces the non-zeros of the LU factors for almost no loss in approximation quality (see Section VI).

A. Analysis of thresholding

When ILUT_CRTP (see Algorithm 3) is run to completion, we obtain block LU factors of

$$\mathbf{P}_r \tilde{\mathbf{A}} \mathbf{P}_c = \mathbf{P}_r \mathbf{A} \mathbf{P}_c + \mathbf{T}^{(l/k-1)}, \tag{10}$$

where the threshold matrix $\mathbf{T}^{(l/k-1)}$ is a sum of permuted perturbation matrices $\tilde{\mathbf{T}}^{(i)}$, that arise from thresholding.

$$\mathbf{T}^{(l/k-1)} = \sum_{i=1}^{l/k-1} \left(\prod_{j=i+1}^{l/k} \tilde{\mathbf{P}}_r^{(j)} \tilde{\mathbf{T}}^{(i)} \prod_{j=i+1}^{l/k} \tilde{\mathbf{P}}_c^{(j)} \right) \tag{11}$$

(10) is a derivation from a result for classical (scalar) ILU factorization algorithms [19]. The bottom right corner submatrices of $\tilde{\mathbf{P}}_r^{(i)}$, $\tilde{\mathbf{P}}_c^{(i)}$, resp. $\tilde{\mathbf{T}}^{(i)}$ in (11) contain $\mathbf{P}_r^{(i)}$, $\mathbf{P}_c^{(i)}$, resp. $\bar{\mathbf{T}}^{(i)}$ from Algorithm 3. From Corollary 8.6.2 and Theorem 8.6.4 in [9], we have

$$\left| \sigma_i(\mathbf{A}) - \sigma_i(\tilde{\mathbf{A}}) \right| \leq \left\| \mathbf{T}^{(l/k-1)} \right\|_2 \quad i = 1, \ldots, l, \tag{12}$$

$$\sqrt{\sum_i (\sigma_i(\mathbf{A}) - \sigma_i(\tilde{\mathbf{A}}))^2} \leq \left\| \mathbf{T}^{(l/k-1)} \right\|_F. \tag{13}$$

The following analysis examines how thresholding would affect a rank-revealing block LU factorization algorithm which produces singular value approximations of $\mathbf{A}$ resp. $\tilde{\mathbf{A}}$, with the same accuracy as the TSVD such that $\|\mathbf{A}^{(i+1)}\|_2 = \sigma_{K+1}(\mathbf{A})$.

Let the singular values $\sigma_1(\mathbf{A}), \ldots, \sigma_l(\mathbf{A})$ be ordered in descending order. Assume that for a fixed iteration $\hat{i}$, with $\hat{K} = \hat{i}k$ and $\hat{K} \ll l$, there is a $\hat{K} + 1^{st}$ singular value $\sigma_{\hat{K}+1}(\mathbf{A}) < \sigma_{\hat{K}}(\mathbf{A})$ and a precision τ such that $\sigma_{\hat{K}+1}(\mathbf{A}) < \tau \|\mathbf{A}\|_2$. For $\tilde{\mathbf{A}}$, we want to satisfy

$$\sigma_{\hat{K}+1}(\tilde{\mathbf{A}}) < \tau \|\mathbf{A}\|_2 . \tag{14}$$

From (12), we can bound $\left| \sigma_{\hat{K}+1}(\mathbf{A}) - \sigma_{\hat{K}+1}(\tilde{\mathbf{A}}) \right|$ with the spectral norm of $\mathbf{T}^{(\hat{i})}$. It must hold that

$$\left\| \mathbf{T}^{(\hat{i})} \right\|_2 < \tau \|\mathbf{A}\|_2 - \sigma_{\hat{K}+1}(\mathbf{A}). \tag{15}$$

to guarantee that (14) is satisfied in any case.

The following analysis on LU_CRTP and ILUT_CRTP is based on both the spectral and Frobenius norm, because the latter is easily evaluated in practice. With LU_CRTP, the singular values of $\mathbf{A}^{(i+1)}$ are approximations to the $l - K$ smallest singular values of $\mathbf{A}$. The bound on $\sigma_j(\mathbf{A}^{(i+1)})$ with respect to $\sigma_{K+j}(\mathbf{A})$ is exponential in the number of iterations,

$$1 \leq \frac{\sigma_j(\mathbf{A}^{(i+1)})}{\sigma_{K+j}(\mathbf{A})} \leq \prod_{v=0}^{i-1} q(m - vk, n - vk, k), \tag{16}$$

for any $1 \leq j \leq l - K$, with q being a polynomial depending on the size of $\mathbf{A}^{(i+1)}$, k and bounds obtained from QR_TP (see [10] for more details). For some iteration $\bar{i}$, with $\bar{i} \geq \hat{i}$ and $\bar{K} = \bar{i}k$, we want $\left\| \mathbf{A}^{(\bar{i}+1)} \right\| < \tau \|\mathbf{A}\|$ resp., in case of ILUT_CRTP,

$$\left\| \tilde{\mathbf{A}}^{(\bar{i}+1)} \right\| < \tau \|\mathbf{A}\| \tag{17}$$

to hold and bound

$$\left| \left\| \mathbf{A}^{(\bar{i}+1)} \right\| - \left\| \tilde{\mathbf{A}}^{(\bar{i}+1)} \right\| \right|. \tag{18}$$

(16) imposes an exponential bound on (18). However, [10] demonstrate that LU_CRTP can be effective in approximating the singular values of $\mathbf{A}$, especially in case of fast singular value decay (see the bound (34) in [10]). By an *effective approximation*, we mean that, in practice, the ratios of the singular values of $\mathbf{A}^{(i+1)}$ with respect to the corresponding

singular values of $\mathbf{A}$ are, on average, close to one. This suggests that by (12) resp. (13) and Lemma 5.1.2 from [14],

$$\left|\left\|\mathbf{A}^{(\bar{i}+1)}\right\|_2 - \left\|\tilde{\mathbf{A}}^{(\bar{i}+1)}\right\|_2\right| \lesssim \left\|\mathbf{T}^{(\bar{i})}\right\|_2,$$

$$\left|\left\|\mathbf{A}^{(\bar{i}+1)}\right\|_F - \left\|\tilde{\mathbf{A}}^{(\bar{i}+1)}\right\|_F\right| \leq$$

$$\sqrt{\sum_i \left(\sigma_i(\mathbf{A}^{(\bar{i}+1)}) - \sigma_i(\tilde{\mathbf{A}}^{(\bar{i}+1)})\right)^2} \lesssim \left\|\mathbf{T}^{(\bar{i})}\right\|_F,$$

and $\bar{i}$ is close or equal to $\hat{i}$ as long as the $l - \bar{K}$ smallest singular values of $\mathbf{A}$ resp. $\tilde{\mathbf{A}}$ are effectively approximated by the singular values of $\mathbf{A}^{(\bar{i}+1)}$ resp. $\tilde{\mathbf{A}}^{(\bar{i}+1)}$. In contrast to (15) with respect to (14),

$$\left\|\mathbf{T}^{(\bar{i})}\right\| < \tau \left\|\mathbf{A}\right\| - \left\|\mathbf{A}^{(\bar{i}+1)}\right\| \tag{19}$$

might not guarantee, that (17) is satisfied, especially when the singular values are not effectively approximated.

If any of the singular values larger than $\sigma_{\bar{K}+1}(\mathbf{A})$ are smaller than machine precision, LU_CRTP may break down due to numerical issues. The same applies to ILUT_CRTP with respect to the singular values of $\tilde{\mathbf{A}}$. The fact that $\bar{i} \geq \hat{i}$ further increases the risk of potential failure. From (12), $\tilde{\mathbf{A}}$ is guaranteed to have at least rank $K+1$ if

$$\left\|\mathbf{T}^{(\bar{i})}\right\| < \sigma_{\bar{K}+1}(\mathbf{A}), \tag{20}$$

which also implies

$$\left\|\mathbf{T}^{(\bar{i})}\right\| < \tau \left\|\mathbf{A}\right\|_2. \tag{21}$$

$\sigma_{\bar{K}+1}(\mathbf{A})$ can be arbitrarily small, e.g., due to a large singular value gap. If (20) does not hold, ILUT_CRTP may fail due to rank deficiency in $\tilde{\mathbf{A}}$.

In conclusion, the convergence of ILUT_CRTP depends on the magnitude of $\left\|\mathbf{T}^{(\bar{i})}\right\|$ in the unknown bound (19), which guarantees that ILUT_CRTP converges at most within the same number of iterations as LU_CRTP, provided that the singular values of $\mathbf{A}$ are effectively approximated, and the unknown bound (20), which guarantees that $\text{rank}(\tilde{\mathbf{A}}) \geq K+1$. We are concerned with establishing bounds on $\left\|\mathbf{T}^{(\bar{i})}\right\|$ that are easily evaluated and that work well in practice.

B. Bounding the threshold matrix

Explicit formulations of $\mathbf{T}^{(i)}$ may produce high memory cost, whereas processing and discarding implicit formulations of the perturbation matrices $\tilde{\mathbf{T}}^{(j)}$ from (11), with $1 \leq j \leq i$ is memory efficient. In line 10 of Algorithm 3 we bound the size of $\left\|\mathbf{T}^{(\bar{i})}\right\|$ with a parameter ϕ, such that

$$\left\|\mathbf{T}^{(i)}\right\| \leq \sqrt{\sum_{j=1}^{i} \left\|\tilde{\mathbf{T}}^{(j)}\right\|_F^2} < \phi. \tag{22}$$

The term in the middle can be easily evaluated and, for large i, the bound is more pessimistic. Ideally, we would set ϕ to the minimum of the bounds (19) and (20). Instead, we can derive a possibly more optimistic bound from (21) at the cost of an additional source for potential algorithm failure. $\|\mathbf{A}\|_2$ is approximated in the first iteration with $\left|\mathbf{R}^{(1)}(1,1)\right|$ in line 5 of Algorithm 3. Due to the rank revealing property of QR_TP,

$$\left|\mathbf{R}^{(1)}(1,1)\right| \leq \left\|\mathbf{A}\right\|_2. \tag{23}$$

Therefore, ϕ might be chosen such that $\phi \leq \tau \left|\mathbf{R}^{(1)}(1,1)\right|$.

C. Determining the threshold

A relevant question is the selection of the threshold μ. We want ILUT_CRTP to determine a reasonable value that effectively performs thresholding and is not likely to break down the algorithm. There are many ways to derive a formula for μ that will work in practice. The Frobenius norm of the individual perturbation matrices $\tilde{\mathbf{T}}^{(i)}$ in (11) is bounded with $\left\|\tilde{\mathbf{T}}^{(i)}\right\|_F \leq \mu\sqrt{\text{nnz}(\tilde{\mathbf{T}}^{(i)})}$. Consequently, a bound on the overall perturbation matrix $\mathbf{T}^{(i)}$ is given as $\left\|\mathbf{T}^{(i)}\right\| \leq \mu i\sqrt{\text{nnz}(\mathbf{T}^{(i)})}$. From (21) and (23) we derive $\mu < \tau \left|\mathbf{R}^{(1)}(1,1)\right| / (\bar{i}\sqrt{\text{nnz}(\mathbf{T}^{(i)})})$ and end up with a heuristic that can be evaluated in practice.

$$\mu := \frac{\tau \left|\mathbf{R}^{(1)}(1,1)\right|}{u\sqrt{\text{nnz}\,(\mathbf{A})}}, \tag{24}$$

where u is an estimation of $\bar{i}$ and the number of non-zeros in $\mathbf{T}^{(\bar{i})}$ is approximated by the number of non-zeros in $\mathbf{A}$. In practice, one might set u at most to the number of iterations such that $K \ll l$ still holds. μ may be too large in case u resp. $\sqrt{\text{nnz}\,(\mathbf{A})}$ was selected much smaller than $\bar{i}$ resp. $\sqrt{\text{nnz}(\mathbf{T}^{(\bar{i})})}$. In this case, (22) could act as a threshold control.

D. Termination criterion

ILUT_CRTP computes $\tilde{\mathbf{L}}_K\ \tilde{\mathbf{U}}_K \approx \mathbf{P}_r\ \tilde{\mathbf{A}}\ \mathbf{P}_c$. The resulting approximation error is given as

$$\tilde{e}_{det} = \left\|\mathbf{P}_r\mathbf{A}\mathbf{P}_c - \tilde{\mathbf{L}}_K\tilde{\mathbf{U}}_K\right\|. \tag{25}$$

The termination criterion of ILUT_CRTP is similar to the criterion of LU_CRTP with the caveat that $\left\|\tilde{\mathbf{A}}^{(i+1)}\right\|$ indicates $\left\|\mathbf{P}_r\tilde{\mathbf{A}}\mathbf{P}_c - \tilde{\mathbf{L}}_K\tilde{\mathbf{U}}_K\right\|$ and *estimates* (25), with

$$\left\|\mathbf{P}_r\mathbf{A}\mathbf{P}_c - \tilde{\mathbf{L}}_K\tilde{\mathbf{U}}_K\right\| \leq \left\|\tilde{\mathbf{A}}^{(i+1)}\right\| + \left\|\mathbf{T}^{(i)}\right\|.$$

The error *estimator* of ILUT_CRTP is defined as

$$\tilde{E}_{det}^{(i)} := \left\|\tilde{\mathbf{A}}^{(i+1)}\right\|_F. \tag{26}$$

IV. Analysis of Asymptotic Arithmetic Complexity

The sequential asymptotic complexity of RandQB_EI at iteration $\bar{i}$ is $\mathcal{O}\big(2K\text{nnz}\,(\mathbf{A}) + \frac{1}{2}(3m+n)K^2 + \frac{2}{\bar{i}}mK^2 + p\left(2K\text{nnz}\,(\mathbf{A}) + (m+n)K^2 + \frac{1}{\bar{i}}(m+n)K^2\right)\big)$[13]. Thus, the cost roughly increases proportional to $p+1$. From [13], the sequential asymptotic complexity of RandUBV at iteration $\bar{i}$ is approximately $\mathcal{O}\left(2K\text{nnz}\,(\mathbf{A}) + \frac{3}{2\bar{i}}(m+n)K^2 + 2nK^2\right)$. For LU_CRTP, the cost is dominated by QR_TP on the columns of $\mathbf{A}^{(i)}$. QR_TP uses a reduction tree to find

Algorithm 3 ILUT_CRTP

Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, block size k, tolerance τ, estimated number of iterations u

Output: $\tilde{\mathbf{L}}_K \in \mathbb{R}^{m\times K}, \tilde{\mathbf{U}}_K \in \mathbb{R}^{K\times n}, \mathbf{P}_r \in \mathbb{R}^{m\times m}, \mathbf{P}_c \in \mathbb{R}^{n\times n}$, rank K, s.t. $\left\|\mathbf{P}_r\mathbf{A}\mathbf{P}_c - \tilde{\mathbf{L}}_K\tilde{\mathbf{U}}_K\right\|_F \lesssim \tau \left\|\mathbf{A}\right\|_F$

1: $\tilde{\mathbf{L}}_K = [\,], \tilde{\mathbf{U}}_K = [\,], \mathbf{P}_r = \mathbf{I} \in \mathbb{R}^{m\times m}, \mathbf{P}_c = \mathbf{I} \in \mathbb{R}^{n\times n}$
2: $\tilde{\mathbf{A}}^{(1)} = \mathbf{A}$, $t = 0$
3: **for** $i = 1, 2, \ldots$ **do**
4: Perform lines 4-11 from Algorithm 2 on $\tilde{\mathbf{A}}^{(i)}$, i.e., obtain $\tilde{\mathbf{L}}_k$, $\tilde{\mathbf{U}}_k$, $\mathbf{P}_r^{(i)}$ and $\mathbf{P}_c^{(i)}$, update $\mathbf{P}_r$ resp. $\mathbf{P}_c$, and enhance $\tilde{\mathbf{L}}_K$ resp. $\tilde{\mathbf{U}}_K$.
5: **if** $i == 1$ **then** $\mu = \tau \left|\mathbf{R}^{(1)}(1,1)\right| / (u\sqrt{\text{nnz}\,(\mathbf{A})})$, determine a bound ϕ with respect to $\tau \left|\mathbf{R}^{(1)}(1,1)\right|$.
6: $\tilde{\mathbf{A}}^{(i+1)} = \bar{\mathbf{A}}_{22} - \bar{\mathbf{A}}_{21}\bar{\mathbf{A}}_{11}^{-1}\bar{\mathbf{A}}_{12}$
7: **if** $\left\|\tilde{\mathbf{A}}^{(i+1)}\right\|_F < \tau \left\|\mathbf{A}\right\|_F$ **then stop**
8: Remove elements in $\mathbf{A}^{(i)}$ that are smaller than μ in absolute value and obtain the threshold matrix $\bar{\mathbf{T}}^{(i)}$.
9: $t = t + \left\|\bar{\mathbf{T}}^{(i)}\right\|_F^2$
10: **If** $\sqrt{t} \geq \phi$ **then** undo steps 9 and 10, set $\mu = 0$.
11: **end for**

the k "most linearly independent" columns of $\mathbf{A}^{(i)}$. Using either a flat or a binary tree, the asymptotic complexity of QR_TP is $\mathcal{O}\left(16k^2\text{nnz}(\mathbf{A}^{(i)})\right)$[10]. Thus, the overall asymptotic complexity of LU_CRTP at iteration $\bar{i}$ is $\mathcal{O}\left(\frac{16}{\bar{i}}K^2\max(\text{nnz}(\mathbf{A}^{(i)}))\right)$, with $1 \leq i \leq \bar{i}$. Due to potential fill-in introduced in $\mathbf{A}^{(i)}$ during the factorization process, $\text{nnz}(\mathbf{A}^{(i)})$ tends to grow with i in a way which is hard to predict.

For square $n \times n$ matrices $\mathbf{A}$, the complexity of RandUBV is very similar to the complexity of RandQB_EI with power parameter $p = 0$. If $\text{nnz}\,(\mathbf{A}) \leq tn$ where $t \ll n$, then LU_CRTP is faster than RandQB_EI at iteration $\bar{i}$ if $\text{nnz}(\mathbf{A}^{(i)}) < \mathcal{O}\big((p+1)\frac{t+(\bar{i}+1)k}{8kt}\text{nnz}\,(\mathbf{A})\big)$. The overall cost depends on the number of iterations needed.

V. Parallel Fixed-Precision Low-Rank Approximation

RandQB_EI performs dense linear algebra operations on tall and skinny resp. short and wide matrices that are widely supported by parallel libraries like, e.g., Elemental [18]. In LU_CRTP most of the parallelism can be exploited in QR_TP. For parallelization on P processes, QR_TP uses a reduction tree to find the k "most linearly independent" columns of a matrix. Suppose that each process P owns $2k$ columns of $\mathbf{A}$. The reduction tree consists of two stages - the local and the global reduction stage. In the local reduction, each process finds the k "most linearly independent" columns among the columns that they own. This step is done completely without communication between processes. In the second stage the k selected columns of each process compete against each other for "most linear independence". For a binary reduction tree, $\log(P)$ reduce operations are performed. At each reduction stage two neighboring processes combine their k columns to find the k "most linearly independent" columns among these $2k$ columns. The last reduction operation finds the k "most linearly independent" columns of the entire input matrix.

We implemented the parallel algorithms in C++ and MPI, using SuiteSparseQR [6] for sparse QR factorizations in the deterministic algorithms. For RandQB_EI we incorporate the Elemental framework. Elemental scatters dense matrices among processes via an elemental distribution. We call `El::Gemm` for dense matrix-matrix multiplication and `El::Multiply` for sparse, 1D row-distributed matrices that are multiplied with dense tall and skinny matrices. `El::qr::ExplicitTS` performs a tall and skinny QR factorization [7] in the orthogonalization procedure. Our LU_CRTP implementation uses a (cyclic) block-column distribution for $\mathbf{A}^{(i)}$ and $\mathbf{U}_K$ and a (cyclic) block-row distribution for $\mathbf{L}_K$. The input matrix was first permuted using COLAMD followed by a postorder traversal of its column elimination tree. COLAMD is a local, intrinsically sequential reordering heuristic. Thus, we apply COLAMD as a preprocessing step. After performing QR_TP on the columns of $\mathbf{A}^{(i)}$ (line 5 of Algorithm 2), the k "most linearly independent" columns of $\mathbf{A}^{(i)}$ are located in the leftmost columns of $\mathbf{A}^{(i)}\mathbf{P}_c^{(i)}$. These columns are then orthogonalized with a sparse QR factorization on the process owning them and scattered among all processes so that QR_TP can be performed (line 7 of Algorithm 2). To efficiently solve the linear system $\bar{\mathbf{A}}_{21}\bar{\mathbf{A}}_{11}^{-1}$ (line 10 of Algorithm 2), $\bar{\mathbf{A}}_{21}$ is scattered and $\bar{\mathbf{A}}_{11}$ is broadcast to all participating processes. An Allgather operation sends the resulting matrix to all processes for the computation of the Schur complement (line 12 of Algorithm 2).

VI. Numerical experiments

We first summarize sequential MATLAB 2020a experiments that evaluate the effectiveness of thresholding with ILUT_CRTP on a large set of small matrices. We then compare runtime, approximation quality and parallel scaling of parallel versions of RandQB_EI, LU_CRTP and ILUT_CRTP for some large sparse matrices on the Vienna Scientific Cluster (VSC4), a HPC system with 700 nodes and 48 cores per node. We used Intel C++ compiler 19.0.5, Intel MPI 2019 Update 7 and Intel MKL 2019 Update 5 with flags: "-Wall -g -O3 -std=gnu++11 -mkl=sequential".

A. Results for ILUT_CRTP

We considered a set of 197 matrices from the San Jose State University Singular Value Database [8], a subset of the 261 matrices used in [10] to test the numerical stability of LU_CRTP. We only employed matrices stored in sparse representation and we omitted 28 matrices for which thresholding could not be performed. 3 out of these 28 matrices are diagonal matrices. The remaining are matrices with integer entries that might be handled with other algorithms. We ordered the matrices in ascending order based on their numerical rank. We stopped the factorization at the numerical rank as in [10] to avoid numerical issues. We used $k = 8$, tolerances $\tau \in \{10^{-3}, 10^{-6}, 10^{-9}\}$ and threshold control $\phi = \tau \left|R^{(1)}(1,1)\right|$.

We computed μ using (24). The estimated number of iterations was set to the iteration at which LU_CRTP terminated in a previous run for the same parameter setting.

In all cases, the error was smaller than $\tau \|\mathbf{A}\|_F$ and agreed with the corresponding estimator. The threshold control was never triggered. The number of non-zeros in the truncated LU factors was significantly reduced for some matrices. Overall, ILUT_CRTP was effective for roughly 30% of the test cases. In some cases, the alternative computation of $\mathbf{L}_K$ that improves numerical stability (see Section II-B3) had to be done which exacerbated the fill-in problem. In 12 cases, the number of non-zeros in the factors from ILUT_CRTP was slightly higher for each of the three values of τ. Column and row pivoting is different for $\mathbf{A}$ resp. $\tilde{\mathbf{A}}$, and therefore, it is possible that more non-zeros in the factors from a factorization of $\tilde{\mathbf{A}}$ are introduced. This suggests that ILUT_CRTP is not a general solution for the fill-in problem. However, for a larger set of matrices, we observed that thresholding can be quite beneficial. The ratio of the total number of non-zeros in the truncated LU factors of LU_CRTP over the total number of non-zeros in the factors of ILUT_CRTP for all 197 matrices with $\tau = 10^{-6}$ is shown in Fig. 1 (left). For completeness, we included quantities that illustrate the maximum fill-in of $\mathbf{A}^{(i)}$ resp. $\tilde{\mathbf{A}}^{(i)}$. Moreover, we evaluated a more aggressive thresholding approach, for which similar or slightly better ratios are observed. In each iteration, values smaller than ϕ are sorted, and the smallest elements are dropped until the bound (22) would be violated otherwise. In 9, 37 resp. 4 cases with respect to the three values of τ, the error (25) was slightly larger than $\tau \|\mathbf{A}\|_F$ despite the estimator (26) indicating that the desired approximation quality was achieved. Due to space limitations, we do not show these results here. We also evaluated potential benefits of applying COLAMD for reordering. While COLAMD reduces fill-in slightly more when applied in every iteration, we found that COLAMD alone was not effective in reducing the number of non-zeros in the LU factors compared to thresholding (see Fig. 1).

B. Accuracy vs. Cost

We now compare the runtime cost per correct digit in parallel low-rank approximations for matrices in Table I. Table II shows the best combination of number of processes and block size for all experiments and all methods; except for ILUT_CRTP, for which we used the same parameters as for LU_CRTP. The threshold μ for ILUT_CRTP was determined using (24) with the estimated number of iterations set to the number of iterations needed by LU_CRTP. RandQB_EI with $p = 1$ provided a good trade-off between runtime and required number of iterations. For matrices `M3`, `M5` and `M6`, RandQB_EI with $p = 0$ did not converge within a reasonable number of iterations.

In sequential Matlab experiments, we also compared RandUBV with RandQB_EI in terms of runtime and number of iterations needed to attain a given approximation quality for matrices `M1`-`M5`. RandUBV was faster in all cases and performs roughly the same amount of work as RandQB_EI with $p = 0$ and same k while often requiring fewer iterations. The number of iterations for RandUBV are included in Table II. Although the runtime in Matlab experiments is not necessarily a sufficiently reliable indicator, these experiments still motivate the development of an efficient parallel implementation of RandUBV as the basis for future work. Despite reorthogonalization, RandQB_EI experienced a slight loss of orthogonality in the approximate basis $\mathbf{Q}_K$ over the iterations. With $i = 1$, the quantity $\left\|\mathbf{Q}_K^T\mathbf{Q}_K - \mathbf{I}\right\|_\infty$ was in the range 10^{-15} to 10^{-14} and increased by about one order of magnitude for matrices in Table I for the respective lowest value of τ in Table II.

TABLE I
TEST MATRICES FROM THE SUITESPARSE MATRIX COLLECTION [5]

label	matrix name	size	nnz	description
`M1`	bcsstk18	11948	149090	Structural Problem
`M2`	raefsky3	21200	1488768	Fluid Dynamics
`M3`	onetone2	36057	222596	Circuit Simulation
`M4`	rajat23	110355	555441	Circuit Simulation
`M5`	mac_econ_fwd500	206500	1273389	Economic Problem
`M6`	circuit5M_dc	3523317	14865409	Circuit Simulation

We also evaluated the minimum rank required to achieve a certain approximation quality with the singular values obtained from the TSVD (see Figs. 2 and 3). Due to the prohibitive computational cost of TSVD, we did not consider its runtime performance. For `M5`, the singular values could not be evaluated. With RandQB_EI, the exact rank approximation can also be determined at small cost [20]. In Fig. 2, a comparison of the minimum rank approximation from RandQB_EI with $p = 2$ to the minimum rank required suggests that the minimum rank required is reasonably approximated in Fig. 3.

Fill-in can severely impact the runtime of LU_CRTP. Fig. 1 (right) shows the fill-in in $\mathbf{A}^{(i)}$ produced throughout the iterations for different matrices. Figs. 2 and 3 show that, if fill-in occurs at a certain approximation quality, the performance of LU_CRTP decreases significantly from that point on. The work required to achieve a certain approximation quality depends on the singular values of $\mathbf{A}$. The right plot of Fig. 3 shows that the rank of the approximation has to be over 40% of the original matrix size to achieve an approximation error below $4 \cdot 10^{-5}$. Contrary to dense factors, sparse factors with a large rank might be still useful. Due to fill-in, LU_CRTP could not achieve the same approximation quality as RandQB_EI within a reasonable time. ILUT_CRTP effectively reduced fill-in and could attain the same quality as RandQB_EI in less time. The error (25) agreed with the estimator (26) in all cases. With respect to larger matrices, we could not find an example where ILUT_CRTP was unable to reduce fill-in, but we do not rule out this possibility due to the results in Fig. 1 (left).

C. Strong scaling

Speedups for three matrices from Table I with respect to different numbers of processes are shown in Fig. 4. All methods used the same block size k to compute low-rank approximations with the same approximation quality for the

TABLE II

RUNTIME PER CORRECT DIGIT FOR TEST MATRICES FROM TABLE I. THE FIRST TWO COLUMNS INDICATE THE LABEL OF THE TEST MATRIX AND THE APPROXIMATION TOLERANCE τ. THE NEXT 13 COLUMNS CONTAIN (WITH RESPECT TO τ) THE NUMBER OF ITERATIONS NEEDED (ITS), THE RUNTIME IN SECONDS, THE BLOCK SIZE k, AND THE NUMBER NP OF MPI PROCESSES USED (AT MOST 4096). ITS_{UBV} SHOWS THE NUMBER OF ITERATIONS NEEDED BY RANDUBV. THE SUBSCRIPTS p_i, $i \in \{0,1,2\}$ IN THE ITERATION AND RUNTIME COLUMN HEADERS OF RANDQB_EI INDICATE THE VALUE OF THE POWER PARAMETER p. NP AND BLOCK SIZE $k \in \{32, 64, 128, 192, 256, 512\}$ WERE SELECTED WITH BEST PERFORMANCE FOR THE HIGHEST APPROXIMATION QUALITY. THE LAST THREE COLUMNS SHOW THE RUNTIME OF ILUT_CRTP, THE NUMBER OF NON-ZEROS IN THE FACTORS OF LU_CRTP OVER THE NUMBER OF NON-ZEROS IN THE FACTORS OF ILUT_CRTP ($\text{RATIO}_{\text{NNZ}}$), AND THE THRESHOLD μ DETERMINED BY THE ALGORITHM. ILUT_CRTP USED THE SAME PARAMETER SETTING (NP AND k) AS LU_CRTP.

			RandQB_EI								LU_CRTP				ILUT_CRTP		
label	τ	its_{UBV}	its_{p_0}	time_{p_0}	its_{p_1}	time_{p_1}	its_{p_2}	time_{p_2}	k	np	its	time	k	np	time	$\text{ratio}_{\text{nnz}}$	μ
M1	10^{-1}	27	33	**1.6**	23	**1.5**	22	**2**	64	128	23	**4**	64	128	**0.8**	37.1	$3.3 \cdot 10^{5}$
	10^{-2}	60	72	**5**	56	**5**	55	**7**			55	**19**			**1.6**	41.6	$1.4 \cdot 10^{4}$
	10^{-3}	95	112	**11**	91	**12**	90	**17**			93	**45**			**3**	18.3	$8.1 \cdot 10^{2}$
M2	10^{-1}	26	33	**2.2**	24	**2.2**	23	**4**	64	256	45	**7**	32	512	**1.9**	265.9	$1.5 \cdot 10^{-5}$
	10^{-2}	50	59	**3.8**	47	**4.6**	46	**7**			92	**37**			**2.7**	376.5	$7.1 \cdot 10^{-7}$
	10^{-3}	80	85	**6.1**	75	**9**	74	**14**			148	**125**			**3.9**	334.3	$4.4 \cdot 10^{-8}$
	10^{-4}	92	99	**7.6**	90	**12**	90	**18**			180	**202**			**4.8**	255.9	$3.6 \cdot 10^{-9}$
M3	10^{-1}	233	164	**50**	152	**29**	151	**46**	64	512	38	**10**	256	128	**9**	2.2	$1.9 \cdot 10^{-2}$
	10^{-2}	256	211	**80**	172	**37**	171	**57**			45	**13**			**11**	2	$1.6 \cdot 10^{-3}$
	10^{-3}	273	-	-	259	**79**	257	**121**			76	**268**			**33**	6.1	$9.6 \cdot 10^{-5}$
M4	10^{-1}	1	1	**1.6**	1	**1.7**	1	**1.8**	192	512	1	**1**	128	512	**-**	-	-
	10^{-2}	5	8	**3.4**	4	**2.8**	4	**4**			5	**4**			**2.2**	3.7	1.5
	10^{-3}	84	110	**93**	72	**81**	69	**140**			104	**191**			**16**	34.4	$7.2 \cdot 10^{-3}$
M5	10^{-1}	69	81	**155**	56	**63**	54	**113**	192	1024	42	**23**	256	512	**22**	1.2	$3.1 \cdot 10^{-1}$
	10^{-2}	138	149	**425**	128	**249**	126	**485**			98	**107**			**52**	5.1	$1.3 \cdot 10^{-2}$
	10^{-3}	164	177	**575**	158	**371**	158	**696**			120	**242**			**79**	6.1	$1.1 \cdot 10^{-3}$
	10^{-4}	195	-	-	191	**532**	190	**953**			145	**611**			**137**	8.6	$8.9 \cdot 10^{-5}$
M6	10^{-3}	-	1	**12**	1	**13**	1	**13**	256	4096	1	**25**	512	4096	**-**	-	-
	10^{-4}	-	-	-	44	**331**	37	**563**			17	**236**			**228**	2.4	$2.6 \cdot 10^{-2}$

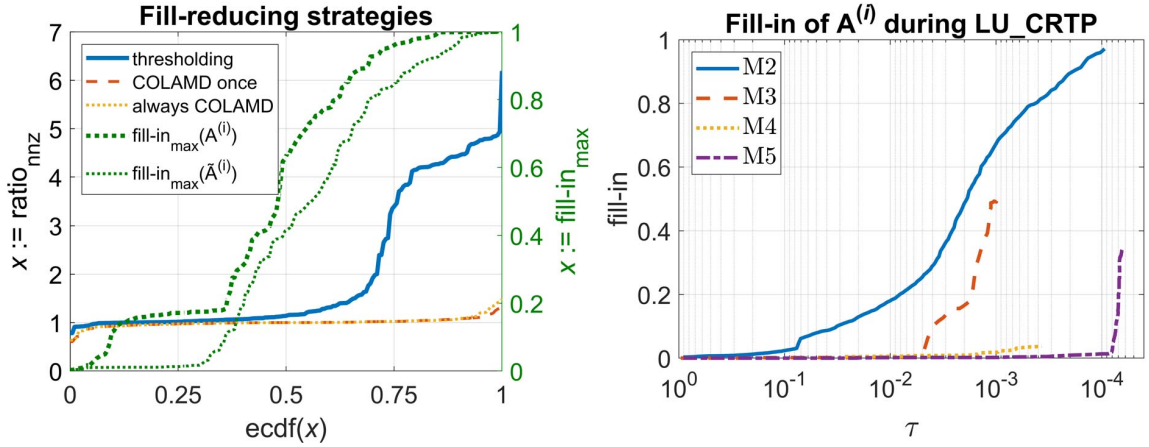

Fig. 1. **Left:** The x-axis shows the empirical distribution function for 197 matrices with $k = 8$ and $\tau = 10^{-6}$. The left y-axis shows the number of non-zeros in the factors of LU_CRTP over the number of non-zeros in the factors of ILUT_CRTP (blue solid line) and the number of non-zeros in the factors of LU_CRTP without COLAMD over the number of non-zeros in the factors of LU_CRTP with COLAMD applied in the first iteration (red dashed line) or with COLAMD applied in every iteration (yellow dotted line) – higher is better. The right y-axis shows fill-in as the maximum density ratio $\max(\text{nnz}(\mathbf{A}^{(i)})/\left(\#\text{rows}(\mathbf{A}^{(i)}) \cdot \#\text{cols}(\mathbf{A}^{(i)})\right))$ (green bold dotted line) resp. $\max(\text{nnz}(\tilde{\mathbf{A}}^{(i)})/(\#\text{rows}(\tilde{\mathbf{A}}^{(i)}) \cdot \#\text{cols}(\tilde{\mathbf{A}}^{(i)})))$ (green thin dotted line) – lower is better. **Right:** Fill-in progression for matrices M2 - M5 after each iteration in LU_CRTP. Block sizes and number of processes as in Table II. The y-axis shows fill-in as the density ratio $\text{nnz}(\mathbf{A}^{(i)})/\left(\#\text{rows}(\mathbf{A}^{(i)}) \cdot \#\text{cols}(\mathbf{A}^{(i)})\right)$.

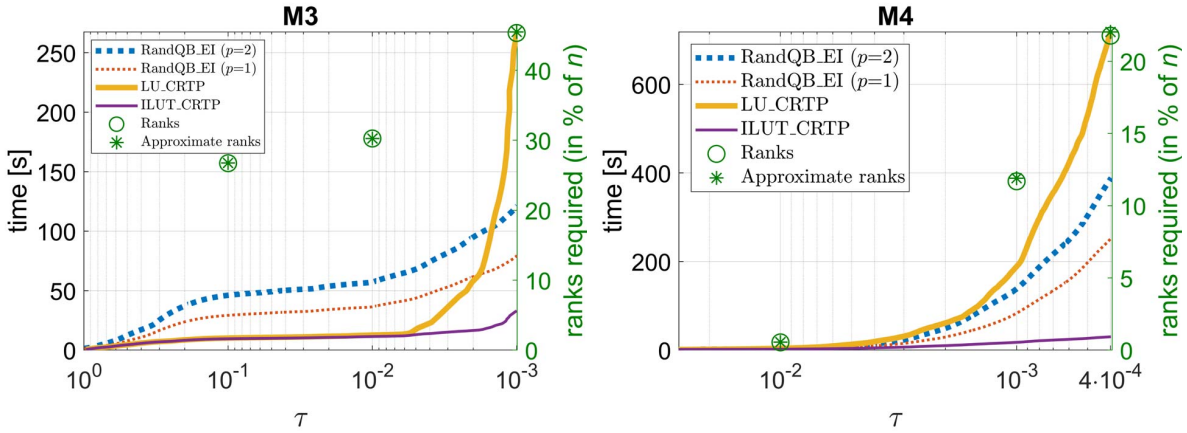

Fig. 2. Runtime vs. approximation quality for `M3` and `M4`. Block sizes and number of processes as in Table II. The left y-axis shows runtime and refers to the four lines. The right y-axis shows the minimum rank required (circles) and the approximated minimum rank (asterisks) to achieve a certain approximation quality as a percentage of the matrix size n.

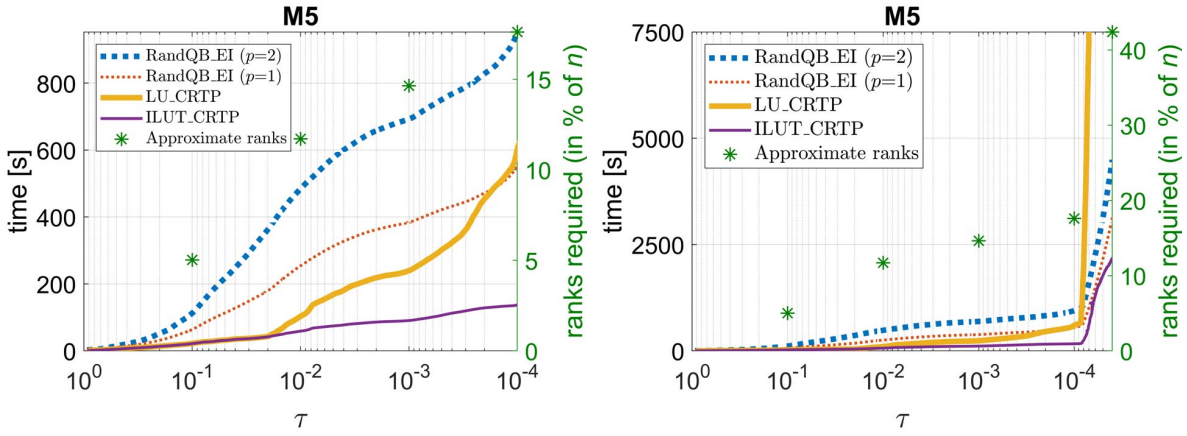

Fig. 3. Runtime vs. approximation quality for `M5`, cf. the caption of Fig. 2. The right plot presents the same data as the left, but with an extended range along the x-axis. Evaluating the minimum rank required to achieve a certain approximation quality was too time consuming.

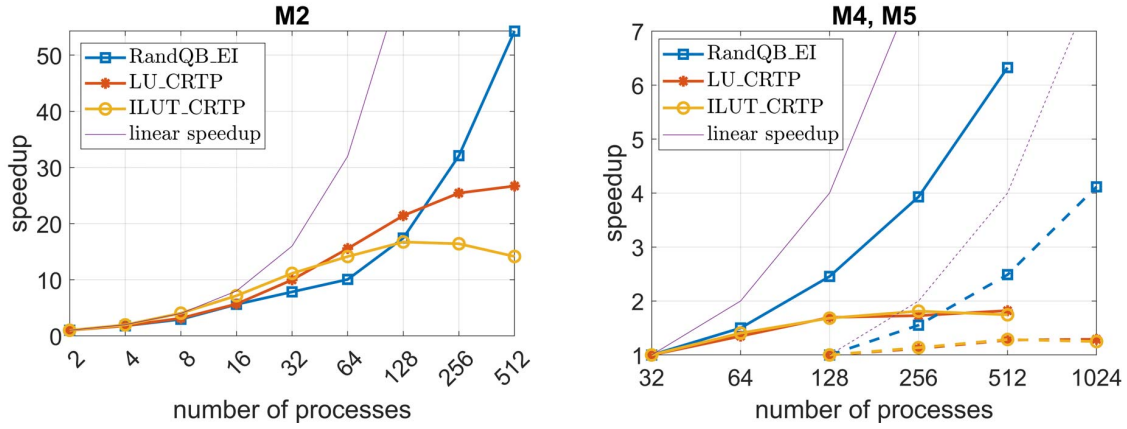

Fig. 4. **Left:** Speedup for matrix `M2` with $k = 32$, approximation error below 10^{-4}. **Right:** Speedup for matrices `M4` and `M5` with $k = 192$, approximation error below 10^{-3}. Solid lines refer to `M4`, dashed lines refer to `M5`.

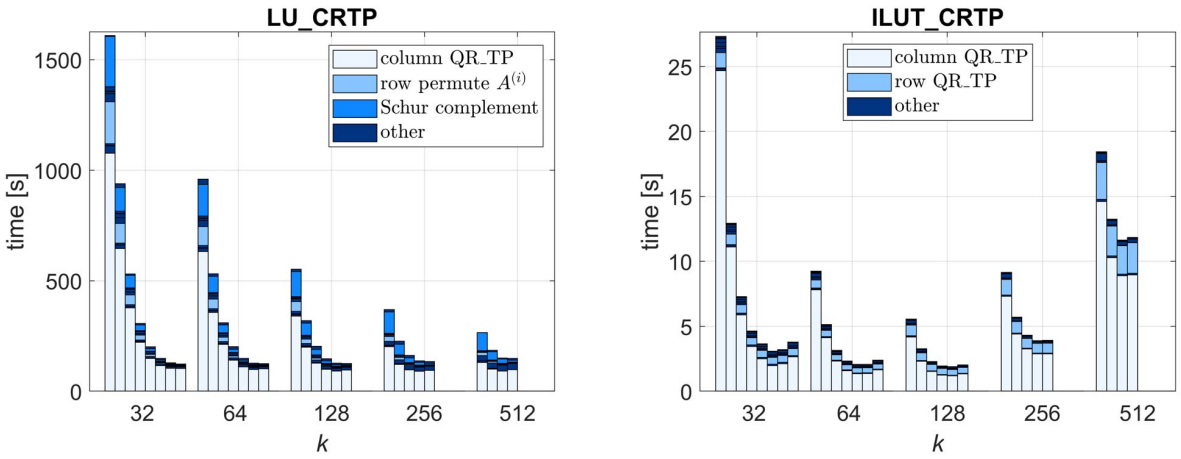

Fig. 5. Runtime of computational kernels in LU_CRTP and ILUT_CRTP with varying number of MPI processes (np) and block sizes k for matrix `M2` and $\tau = 10^{-3}$. LU_CRTP produces significant fill-in for `M2` (see Fig. 1). The runtime for each kernel was accumulated over the number of iterations and the maximum time among processes was selected. The left-most bar in each bar group shows the runtime with np = 4. The number of processes doubles as we move to the next bar within a bar group until the product np $\cdot$ k would be larger than the size of `M2` (21200). E.g., with $k = 32$, up to 512 processes are possible. With $k = 512$, at most 32 processes can be used. If significant fill-in is produced, the most expensive kernels, besides column QR_TP, are the computation of the Schur complement (line 12 of Algorithm 2) and the local row permutations of $\mathbf{A}^{(i)}$, that are performed right after row QR_TP in line 8 of Algorithm 2 (the column permutations that would require communication in line 8 can be done implicitly during column QR_TP). If only minor fill-in occurs or if fill-in is effectively reduced with ILUT_CRTP, the most expensive kernel besides column QR_TP is row QR_TP. Larger values for k or np require more communication. This leads to increased runtimes for ILUT_CRTP within and among bar groups when, at some point, the communication cost outweighs the benefit of distributing the workload among processes. LU_CRTP must process more non-zeros than ILUT_CRTP and therefore benefits more from workload distribution. This also implies that using a combination of values for k and np with best performance for LU_CRTP is not necessarily the best configuration for ILUT_CRTP.

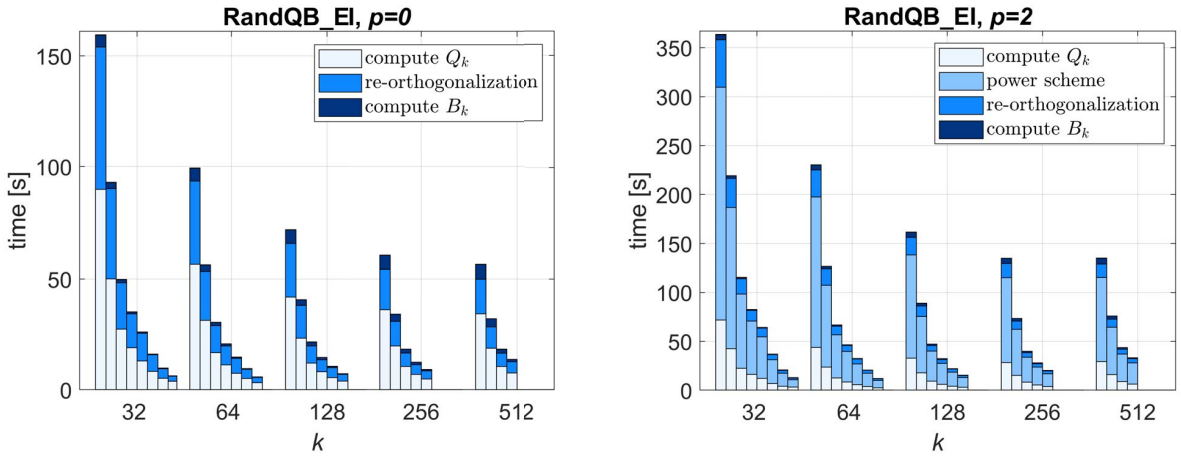

Fig. 6. Runtime of computational kernels in RandQB_EI with varying number of MPI processes and block sizes k for matrix `M2` and $\tau = 10^{-3}$, cf. the caption of Fig. 5. 170 resp. 148 iterations for $p = 0$ resp. $p = 2$ with $k = 32$. 11 resp. 10 iterations for $p = 0$ resp. $p = 2$ with $k = 512$.

respective matrix. In the left plot, we observe that, at some point, the deterministic methods do not scale anymore. Applying QR_TP on the columns of the input dominates the cost of LU_CRTP. The method scales well, as long as most of the time is spent in the local reduction tree of QR_TP. The method does not scale anymore when $\log(P)$ approaches the height of the tree, such that most of the time is spent in the global reduction. More parallelism is possible by reducing the block size k at the cost of more iterations for achieving the same precision. A similar trend is observed for two larger problems in the right plot of Fig. 4. Here, the speedup is shown with respect to a larger number of processes such that the deterministic methods do not scale well from the start. ILUT_CRTP does the least amount of work overall and at some point, is negatively affected by more parallelism. A breakdown of the most expensive kernels with respect to varying number of MPI processes and block sizes k is shown in Figs. 5 and 6.

VII. Conclusion

Two fixed-precision algorithms, a randomized and a deterministic one, for computing low-rank approximations of large sparse matrices were studied, RandQB_EI and LU_CRTP. For suitable comparison, we used the efficient error indicator of RandQB_EI and developed a similar error indicator for LU_CRTP. Due to potential fill-in in the factorization process, the runtime of LU_CRTP is less predictable than the one of RandQB_EI. To improve the runtime performance of the deterministic approach, we developed ILUT_CRTP. Experimental evaluation on a large set of test matrices showed that the new algorithm achieves the same approximation quality as LU_CRTP with, in some cases, significant performance improvements and reduced number of non-zeros in the LU factors. We developed distributed-memory parallelizations of RandQB_EI, LU_CRTP and ILUT_CRTP and compared them in terms of parallel scalability, runtime and accuracy on several large test matrices. While RandQB_EI overall exhibited better scalability, we found that LU_CRTP can still outperform RandQB_EI if fill-in is not significant. ILUT_CRTP effectively improved performance and reduced the number of non-zeros in the truncated LU factors, achieving speedups up to 40 over LU_CRTP.

References

[1] C. Bach, F. Duddeck, and L. Song. "Fixed-precision randomized low-rank approximation methods for nonlinear model order reduction of large systems". In: *Int. J. Numer. Methods Engrg.* 119.8 (2019), pp. 687–711.

[2] S. Cayrols. "Minimizing communication for incomplete factorizations and low-rank approximations on large scale computers". PhD thesis. Sorbonne Univ., 2019.

[3] K. L. Clarkson and D. P. Woodruff. "Low-Rank Approximation and Regression in Input Sparsity Time". In: *J. ACM* 63.6 (2017), pp. 1–45.

[4] T. Davis, J. Gilbert, S. Larimore, and E. Ng. "A column approximate minimum degree ordering algorithm". In: *ACM Trans. Math. Softw.* 30.3 (2004), pp. 353–376.

[5] T. Davis and Y. Hu. "The University of Florida Sparse Matrix Collection". In: *ACM Trans. Math. Softw.* 38.1 (2011), pp. 1–25.

[6] T. Davis. "Algorithm 915, SuiteSparseQR: Multifrontal Multithreaded Rank-Revealing Sparse QR Factorization". In: *ACM Trans. Math. Softw.* 38.1 (2011), pp. 1–22.

[7] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. "Communication-Optimal Parallel and Sequential QR and LU Factorizations". In: *SIAM J. Sci. Comput.* 34.1 (2012), pp. 206–239.

[8] L. Foster. *San Jose State University Singular Matrix Database*. URL: http://www.math.sjsu.edu/singular/matrices/ (visited on 10/14/2021).

[9] G. Golub and C. Van Loan. *Matrix Computations*. 4th edition. Johns Hopkins University Press, 2013.

[10] L. Grigori, S. Cayrols, and J. Demmel. "Low Rank Approximation of a Sparse Matrix Based on LU Factorization with Column and Row Tournament Pivoting". In: *SIAM J. Sci. Comput.* 40.2 (2018), pp. C181–C209.

[11] W. Hackbusch. *Hierarchical Matrices: Algorithms and Analysis*. Berlin, Germany: Springer, 2015.

[12] N. Halko, P.-G. Martinsson, and J. Tropp. "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions". In: *SIAM Rev.* 53.2 (2011), pp. 217–288.

[13] E. Hallman. "A Block Bidiagonalization Method for Fixed-Accuracy Low-Rank Matrix Approximation". In: *arXiv* abs/2101.01247 (2021).

[14] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 2012.

[15] I. Markovsky. *Low-Rank Approximation: Algorithms, Implementation, Applications*. Springer, 2019.

[16] P.-G. Martinsson and S. Voronin. "A Randomized Blocked Algorithm for Efficiently Computing Rank-revealing Factorizations of Matrices". In: *SIAM J. Sci. Comput.* 38.5 (2016), S485–S507.

[17] M. Mascagni, I. Yamazaki, A. Ida, R. Yokota, and J. Dongarra. "Distributed-Memory Lattice H-Matrix Factorization". In: *Int. J. High Perform. Comput. Appl.* 33.5 (2019), pp. 1046–1063.

[18] J. Poulson, B. Marker, R. A. Van de Geijn, J. R. Hammond, and N. A. Romero. "Elemental: A new framework for distributed memory dense matrix computations". In: *ACM Trans. Math. Softw.* 39.2 (2013), pp. 1–24.

[19] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003.

[20] W. Yu, Y. Gu, and Y. Li. "Efficient Randomized Algorithms for the Fixed-Precision Low-Rank Matrix Approximation". In: *SIAM J. Matrix Anal. Appl.* 39.3 (2018), pp. 1339–1359.