Introduction to communication avoiding algorithms for direct methods of factorization in Linear Algebra

Laura Grigori

Abstract Modern, massively parallel computers play a fundamental role in a large and rapidly growing number of academic and industrial applications. However, extremely complex hardware architectures, which these computers feature, effectively prevent most of the existing algorithms to scale up to a large number of processors. Part of the reason behind this is the exponentially increasing divide between the time required to communicate a floating-point number between two processors and the time needed to perform a single floating point operation by one of the processors. Previous investigations have typically aimed at overlapping as much as possible communication with computation. While this is important, the improvement achieved by such an approach is not sufficient. The communication problem needs to be addressed also directly at the mathematical formulation and the algorithmic design level. This requires a shift in the way the numerical algorithms are devised, which now need to reduce, or even minimize when possible, the number of communication instances. Communication avoiding algorithms provide such a perspective on designing algorithms that minimize communication in numerical linear algebra. In this document we describe some of the novel numerical schemes employed by those communication avoiding algorithms, with a particular focus on direct methods of factorization.

1 Introduction

This document discusses one of the main challenges in high performance computing which is the increased communication cost, the fact that the time needed to communicate a floating-point number between two processors exceeds by huge factors the time required to perform a single floating point operation by one of the proces-

Laura Grigori
Inria Paris, Alpines, and UPMC Univ Paris 06, CNRS UMR 7598, Laboratoire Jacques-Louis Lions, Paris, France, e-mail: Laura.Grigori@inria.fr

sors. Several works have shown that this gap has been increasing exponentially (see e.g. [37]) and it is predicted that it will continue to do so in the foreseeable future! The memory wall problem, the disparity between the time required to transfer data between different levels of the memory hierarchy and the time required to perform floating point operations, was predicted already in 1995 by Wulf and McKee [72]. However, we are also facing now the inter-processor communication wall. Because of this, most of the algorithms are not able to scale to a large number of processors of these massively parallel machines. The slow rate of improvement of latency is mainly due to physical limitations, and it is not expected that the hardware research will find a solution to this problem soon. Hence the communication problem needs to be addressed also at the algorithmic and software level.

The communication gap is already seen and felt in the current, highly optimised applications, as illustrated by the top panel of figure 1, which displays the performance of a linear solver based on iterative methods used in the cosmic microwave background (CMB) data analysis application from astrophysics. This performance result is extracted from [44] [1] where a more detailed description of the algorithms can be found. It shows the cost of a single iteration of conjugate gradient iterative solver preconditioned by a block diagonal preconditioner, together with the time spent on computation and communication. These runs were performed on a Cray XE6 system, each node of the system is composed of two twelve-cores AMD MagnyCours. It can be seen that the communication becomes quickly very costly, potentially dominating the runtime of the solver when more than 6000 cores are used (each MPI process uses 6 cores). The bottom part of figure 1 displays the performance estimated on a model of an exascale machine of a dense solver based on Gaussian elimination with partial pivoting (GEPP) factorization [2] (see also [42]). The plot displays the computation to communication ratio as a function of the problem size, vertical axis, and the number of used nodes, horizontal axis. The plot shows two regimes, at the top left corner this is the computation which dominates the run time, while at the bottom right this is the communication. The white region marks the regime where the problem is too large to fit in memory. We note that the communication-dominated regime is reached very fast, even for such a computationally intensive operation requiring $O(n^3)$ floating point operations (flops) as shown here (where the matrix to be factored is of size $n \times n$).

1.1 Communication avoiding algorithms

New communication avoiding algorithms have been introduced in the recent years that minimize communication and are as stable as classic algorithms. We describe in more details the communication complexity of direct methods of factorization in section 2. Then in the following sections we describe communication avoiding al-

[1] Courtesy of M. Szydlarski.

[2] Courtesy of M. Jacquelin.

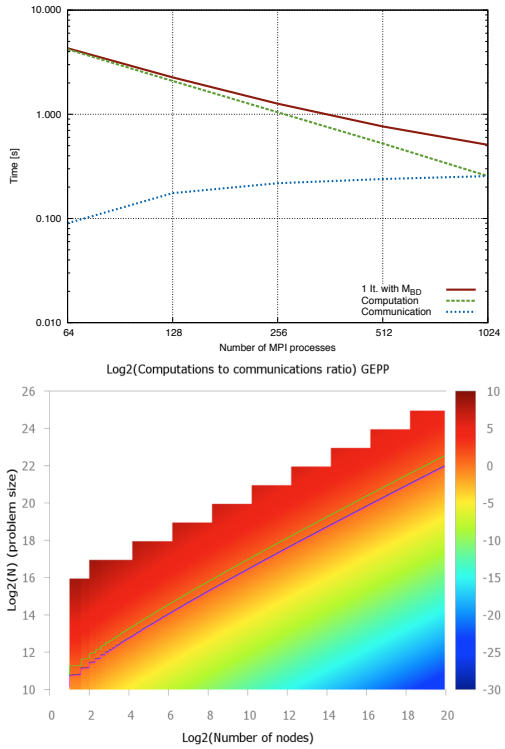


Fig. 1 Communication bottleneck of two algorithms, a dense linear solver based on LU factorization with partial pivoting (bottom figure) and a sparse iterative solver applied to the map-making problem in astrophysics (top figure, using data extracted from [44]).

gorithms for direct methods of factorization that attain the lower bounds on communication discussed in section 2 (up to polylogarithmic factors). Section 3 describes CALU, a communication avoiding LU factorization. Section 4 presents CAQR, a communication avoiding QR factorization, while section 5 discusses a communication avoiding rank revealing factorization. Section 5.3 focuses on computing a low rank matrix approximation. We assume for simplicity real matrices, but the algorithms can be generalized to complex matrices.

This document follows the presentation of the communication avoiding algorithms from the original papers that introduced them. The communication avoiding LU factorization is introduced in [40, 41], while the communication avoiding QR factorization is introduced in [24], and with many more details in the technical report [21]. A follow-up paper [4] allows to reconstruct Householder vectors such that it is sufficient to replace the panel factorization in a QR factorization to obtain a communication avoiding algorithm. A communication avoiding rank revealing QR factorization is presented in [20], while an LU factorization more stable than Gaussian elimination with partial pivoting is presented in [55]. When executed in

parallel, these algorithms reduce significantly the number of messages exchanged with respect to classic algorithms as for example implemented in LAPACK [1] and ScaLAPACK [12]. They sometimes perform redundant computations, however these computations represent lower order terms with respect to the computational complexity of classic algorithms. In practice, when used with advanced scheduling techniques, the new algorithms lead to important speedups over existing algorithms [25, 26].

We cite here several other communication avoiding algorithms that were introduced in the recent years, but they are not described in this document. Communication avoiding algorithms for singular value decomposition (SVD) and eigenvalue problems are described in [3]. Bounds on communication for fast matrix multiplication are introduced in [7] and communication optimal algorithms for Strassen matrix multiplication are discussed in [6]. For sparse matrices, the communication complexity of the Cholesky factorization is studied in [39], while a communication optimal sparse matrix matrix multiplication algorithm is presented in [2].

Let's now give an example of classic algorithms that do not attain the lower bounds on communication. Several direct methods of factorization require some form of pivoting to preserve numerical stability, or reveal the rank of a matrix. The classic pivoting schemes, as partial pivoting in LU factorization or column pivoting in rank revealing QR, imply that the subsequent parallel algorithm cannot attain the lower bounds on communication. For a machine with one level of parallelism, the number of messages exchanged is on the order of $n \log P$, where n is the number of columns of the matrix and P is the number of processors used in the algorithm. For square matrices and when the memory per processor is of size $O(n^2/P)$, the lower bound on number of messages is $\Omega(\sqrt{P})$ (see equation (4) in section 2). Hence in this case minimizing communication requires to invent novel pivoting schemes. There are examples in the literature of pivoting schemes, as for example proposed by Barron and Swinnerton-Dyer in their notable work [10], that minimize communication on sequential machines. At that time the matrices were of dimension 100×100 and the pivoting scheme was stable. But as shown in [40], this method can become unstable for sizes of the matrices we encounter nowadays. The solution that we have developed for LU factorization is described in section 3.

For iterative methods of factorization, most of the research around communication avoiding algorithms focuses on Krylov subspace methods. Those methods, as Conjugate Gradient (CG) [48], Generalized Minimal RESidual (GMRES) [63], Bi-Conjugate Gradient STABilized (Bi-CGSTAB) [70] are the most used iterative methods for solving linear systems of the form $Ax = b$, where A is very large and sparse. Starting from an initial solution x_0 and an initial residual r_0, a new approximate solution x_k is computed at iteration k by minimizing a measure of the error over $x_0 + K_k(A, r_0)$, where $K_k(A, r_0) = \text{span}[r_0, Ar_0, \ldots, A^{k-1}r_0,]$ is the Krylov subspace of dimension k. Every iteration requires computing the product of A (and in some cases of A^T) with a vector and several other operations as dot products related to the orthogonalization of the vectors of the basis. In the parallel case, the input matrix and the vectors are distributed over processors. Hence every iteration requires point to point communications for multiplying A with a sparse vector and collective com-

munications for the dot products. On a large number of processors, the collective communications start dominating the overall cost of the iterative process. There are two main approaches used to reduce communication. The first approach relies on so called s-step methods [17, 31, 51, 15] that compute s vectors of the Krylov basis with no communication and then orthogonalize them against the previous vectors of the basis and against themselves. With this approach, the communication is performed every s iterations and this results in an overall reduction of the communication cost of the iterative method [15, 22, 51]. A second approach, described in [43], relies on enriching the subspace used in these methods that allows, at the cost of some extra computation, to reduce communication, while ensuring theoretically that the convergence is at least as fast as the convergence of the corresponding existing Krylov method. For this, first the problem is partitioned into P domains, and at each iteration of the iterative method, P dimensions are added to the search space instead of one dimension as in classic methods. Experimental results presented in [43] show that enlarged CG converges faster than CG on matrices arising from several different applications. This method is related to block Krylov subspace methods [59]. There are few preconditioners developed in this context, one of them is the communication avoiding incomplete LU preconditioner described in [43].

1.2 Different previous approaches for reducing communication

Most of the approaches investigated in the past to address this problem rely on changing the schedule of the computation such that the communication is overlapped as much as possible with the computation. However such an approach can lead to limited improvements. Ghosting is a different technique for reducing communication, in which a processor ghosts some data and performs redundantly some computation, thus avoiding waiting to receive the results of this computation from other processors. But the dependency between computations in linear algebra operations prevents a straightforward application of ghosting. There are operations for which ghosting would require storing and performing on one processor an important fraction of the entire computation. Cache-oblivious algorithms represent a different approach introduced in 1999 for Fast Fourier Transforms [33], and then extended to graph algorithms, dynamic programming, etc. They were also applied to several operations in linear algebra (see e.g. [30, 46, 68]) as dense LU and QR factorizations. These cache-oblivious factorizations are computed through recursive calls of linear algebra operations on sub-blocks of the matrix to be factored. Since the sub-blocks become smaller and smaller, at some level of the recursion they fit in memory, and overall the amount of data transferred between different levels of the memory hierarchy is reduced. However there are cases in which the number of messages is not reduced and they perform asymptotically more floating-point operations.

1.3 Notations

We use Matlab like notation. We refer to the element of A at row i and column j as $A(i, j)$. The submatrix of A formed by rows from i to j and columns from k to s is referred to as $A(i : j, k : s)$. The matrix formed by concatenating two matrices A_1, A_2 stacked atop one another is referred to as $[A_1;A_2]$. The matrix formed by concatenating two matrices one next to another is referred to as $[A_1,A_2]$. The matrix formed by the absolute value of the elements of A is referred to as $|A|$. The identity matrix of size $n \times n$ is referred to as I_n.

To estimate the performance of an algorithm, we use the $\alpha - \beta - \gamma$ model. With this model, the time required for transferring one message of n words between two processors is estimated as $\beta \cdot n + \alpha$, where β is the interprocessor bandwidth cost per word and α is the interprocessor latency. Given the time required to compute one floating point operation (flop) γ, the time of a parallel algorithm is estimated as,

$$T = \gamma \cdot \#\text{ flops} + \beta \cdot \#\text{ words} + \alpha \cdot \#\text{ messages}, \tag{1}$$

where $\#flops$ represents the computation, $\#words$ the volume of communication, and $\#messages$ the number of messages exchanged on the critical path of the parallel algorithm.

2 Lower bounds on communication for dense linear algebra

In this section we review recent results obtained on the communication complexity of dense linear algebra operations. In the sequential case, these results consider a machine with two levels of memory, at the first level the memory has size M words, at the second level, the memory has infinite size but the access to the data is much slower. In the parallel case, they assume one level of parallelism, that is a parallel machine with P processing units connected through a fast network. One notable previous theoretical result on communication complexity is a result derived by Hong and Kung [52] providing lower bounds on the volume of communication of dense matrix multiplication for sequential machines. These bounds are extended to dense parallel matrix multiplication in [53] (with a different approach used for the proofs). It was shown in [21] that these bounds hold for LU and QR factorizations (under certain assumptions) and that they can be used to also identify lower bounds on the number of messages. General proofs that hold for almost all direct dense linear algebra operations are given in [8]. Consider a matrix of size $m \times n$ and a direct dense linear algebra algorithm as matrix multiplication, LU, QR, or rank revealing QR factorization, executed on a sequential machine with fast memory of size M words and slow memory of infinite size. The number of words and the number of messages transferred between slow and fast memory is bounded as,

$$\#\text{ words} \geq \Omega\left(\frac{mn^2}{M^{1/2}}\right),\ \#\text{ messages} \geq \Omega\left(\frac{mn^2}{M^{3/2}}\right). \tag{2}$$

The bounds can be obtained by using the Loomis-Whitney inequality, as proven in [8, 53], which allows to bound the number of flops performed given an amount of data available in the memory of size M. Equation (2) can be used to derive bounds for a parallel program executed on P processors. For simplicity we consider in the following square dense matrices of size $n \times n$. Assuming that at least one processor does n^3/P floating point operations, and that the size of the memory of each processor M has a value between n^2/P and $n^2/P^{2/3}$, the lower bounds become

$$\#\text{ words} \geq \Omega\left(\frac{n^3}{P \cdot M^{1/2}}\right),\ \#\text{ messages} \geq \Omega\left(\frac{n^3}{P \cdot M^{3/2}}\right). \tag{3}$$

When the memory of each processor is on the order of n^2/P, that is each processor has enough memory to store $/1/P$-th of the matrices involved in the linear algebra operation and there is no replication of the data, the lower bounds become

$$\#\text{ words} \geq \Omega\left(\frac{n^2}{\sqrt{P}}\right),\ \#\text{ messages} \geq \Omega\left(\sqrt{P}\right). \tag{4}$$

Algorithms that attain these bounds are referred to as $2D$ algorithms. Cannon's matrix multiplication [14] is such an algorithm that attains the lower bounds on communication from (4). The lower bounds from (3) become smaller when the memory size is increased, and this until M is on the order of $n^2/P^{2/3}$. Indeed, even in the case of infinite memory M, it is shown in e.g. [5] that at least one processor must communicate $\Omega(n^2/P^{2/3})$ words of data. This leads to the following lower bounds,

$$\#\text{ words} \geq \Omega\left(\frac{n^2}{P^{2/3}}\right),\ \#\text{ messages} \geq \Omega\left(1\right). \tag{5}$$

Algorithms that attain the lower bounds on communication in the case when M is larger than n^2/P are referred to as $3D$ algorithms. In these algorithms, the matrices are replicated over a 3D grid of processors.

These lower bounds on communication allow to identify that most of the existing algorithms as implemented in well-known numerical libraries as ScaLAPACK and LAPACK do not minimize communication. In the rest of this document we will discuss 2D algorithms that store only one copy of the matrices involved in the computation and use a memory on the order of n^2/P per processor (for square matrices). We discuss only the parallel case, however the algorithms can be adapted to minimize communication between two levels of memory in the sequential case.

3 Communication avoiding LU factorization

Given a matrix $A \in \mathbb{R}^{m\times n}$, $m \geq n$, the LU factorization decomposes the matrix A into the product $L \cdot U$, where L is a lower triangular matrix of size $m \times n$ with unit diagonal and U is an upper triangular matrix of size $n \times n$. This algorithm can be written as three nested loops, whose order can be interchanged. A so-called right-looking version of the algorithm is presented in Algorithm 1. To avoid division by small elements and preserve numerical stability, this algorithm uses partial pivoting. During the factorization, for each column k, the element of maximum magnitude in $A(k:n,k)$ is permuted to the diagonal position before the column is factored. Then, multiples of row k are added to all subsequent rows $k+1$ to m to annihilate all the nonzero elements below the diagonal. This algorithm requires $mn^2 - n^3/3$ flops. An in-place version can be easily obtained by overwriting the matrix A with the matrices L and U.

Algorithm 1 LU factorization with partial pivoting (GEPP)

Require: $A \in \mathbb{R}^{m\times n}$
1: Let $L \in \mathbb{R}^{m\times n}$ be initialized with identity matrix and $U \in \mathbb{R}^{n\times n}$ with zero matrix.
2: **for** $k = 1$ to $n-1$ **do**
3: Let $A(i,k)$ be the element of maximum magnitude in $A(k:m,k)$
4: Permute row i and row k
5: $U(k,k:n) = A(k,k:n)$
6: $L(k+1:m,k) = A(k+1:m,k)/A(k,k)$
7: **for** $i = k+1:m$ **do**
8: **for** $j = k+1:n$ **do**
9: $A(i,j) = A(i,j) - A(i,k)A(k,j)$
10: **end for**
11: **end for**
12: **end for**
13: $U(n,n) = A(n,n)$

Typically, this factorization is implemented by using a block algorithm, in which the matrix is partitioned into blocks of columns of size b. In the remaining of this document, without loss of generality, we consider that n and m are multiples of b. At the first iteration, the matrix A is partitioned as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \tag{6}$$

where A_{11} is of size $b \times b$, A_{21} is of size $(m-b) \times b$, A_{12} is of size $b \times (n-b)$, and A_{22} is of size $(m-b) \times (n-b)$. With a right looking approach, the block algorithm computes the LU factorization with partial pivoting of the first block-column (panel), it determines the block U_{12}, and then it updates the trailing matrix A_{22}. The factorization obtained after the first iteration is

$$\Pi_1 A = \begin{bmatrix} L_{11} \\ L_{21}\ I_{m-b} \end{bmatrix} \cdot \begin{bmatrix} U_{11}\ U_{12} \\ A_{22}^1 \end{bmatrix}, \tag{7}$$

where $A_{22}^1 = A_{22} - L_{21}U_{12}$. The algorithm continues recursively on the trailing matrix A_{22}^1.

Parallel block LU factorization

We describe now briefly a parallel block LU algorithm by following its implementation in ScaLAPACK (`PDGETRF` routine). The input matrix is distributed over a $P_r \times P_c$ grid of processors using a bidimensional (2D) block cyclic layout with blocks of size $b \times b$. As an example, with a 2×2 grid of processors, the blocks of the matrix are distributed over processors as

$$\begin{bmatrix} P_0\ P_1\ P_0\ P_1\ \dots \\ P_2\ P_3\ P_2\ P_3\ \dots \\ \vdots\ \vdots\ \vdots\ \vdots\ \ddots \end{bmatrix}.$$

Algorithm 2 LU factorization with partial pivoting using a block algorithm

Require: $A \in \mathbb{R}^{m \times n}$ distributed over a 2D grid of processors $P = P_r \times P_c$

1: Let $L \in \mathbb{R}^{m \times n}$ be initialized with identity matrix and $U \in \mathbb{R}^{n \times n}$ with zero matrix
2: **for** k = 1 to n/b **do**
3: $k_b = (k-1) \cdot b + 1$, $k_e = k_b + b - 1$
4: Compute panel factorization using partial pivoting (processors in the same column of the process grid)

$$\Pi_k A(k_b : m, k_b : k_e) = L(k_b : m, k_b : k_e) \cdot U(k_b : k_e, k_b : k_e)$$

5: Broadcast pivot information along the rows of the process grid, pivot by applying the permutation matrix Π_k on the entire matrix (all processors)

$$A = \Pi_k A$$

6: Broadcast right $L(k_b : k_e, k_b : k_e)$, compute block row of U (processors in the same row of the process grid)

$$U(k_b : k_e, k_e + 1 : n) = L(k_b : k_e, k_b : k_e)^{-1} A(k_b : k_e, k_e + 1 : n)$$

7: Broadcast along rows of the process grid $L(k_e + 1 : m, k_b : k_e)$, broadcast along columns of the process grid $U(k_b : k_e, k_e + 1 : n)$, update trailing matrix (all processors)

$$A(k_e + 1 : m, k_e + 1 : n) = A(k_e + 1 : m, k_e + 1 : n) - L(k_e + 1 : m, k_b : k_e) \cdot U(k_b : k_e, k_e + 1 : n)$$

8: **end for**

Algorithm 2 presents the main operations executed at each iteration of the block LU factorization. In terms of number of messages, it can be seen that, except for the panel factorization, all the other operations rely on collective communications which require exchanging $O(\log P_r)$ or $O(\log P_c)$ messages. Hence, the latency bottleneck lies in the panel factorization, where the LU factorization is performed column by column as in Algorithm 1. For each column, finding the element of maximum magnitude requires a reduce-type communication based on exchanging $\log P_r$ messages. In other words, partial pivoting requires performing a number of $O(n \log P_r)$ collective communications, which depends on n, the number of columns of the matrix. Since the lower bound on number of messages in equation (4) is $\Omega(\sqrt{P})$ for square matrices, LU factorization with partial pivoting as implemented in ScaLAPACK does not allow to minimize communication on a parallel machine. However we note that recently it has been shown that with a sophisticated data layout, it is possible to minimize data movement on a sequential machine for LU with partial pivoting [9].

3.1 Tournament pivoting

Communication avoiding LU based on tournament pivoting was introduced in [40, 41] where a more detailed description can be found. We refer to this factorization as CALU. As in a classic LU factorization, the matrix is partitioned in blocks of b columns. At the first iteration, consider the matrix A partitioned as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

where A_{11} is of size $b \times b$, A_{21} is of size $(m-b) \times b$, A_{12} is of size $b \times (n-b)$, and A_{22} is of size $(m-b) \times (n-b)$. With tournament pivoting, the panel factorization is performed as following. A preprocessing step plays a tournament to find at low communication cost b pivots that can be used to factor the entire panel. The selected b rows are permuted into the leading positions and they are used as pivots for the LU factorization of the entire panel (which is performed hence with no permutation). The preprocessing step is performed as a reduction operation where at each node of the reduction tree Gaussian elimination with partial pivoting (GEPP) is used to select b pivot rows. This strategy has the property that the communication for computing the panel factorization does not depend on the number of columns, but depends only on the number of processors. We refer to this procedure for computing the LU factorization of the panel as TSLU. The communication avoiding LU algorithm computes then the block U_{12}, updates the trailing matrix A_{22}, and a factorization as in equation (7) is obtained. It then continues recursively on the updated block A^1_{22}.

We explain now in more details tournament pivoting. Given P processors, the panel is partitioned into P block rows. We consider here the simple case $P = 4$, a binary reduction tree, and we suppose that m is a multiple of 4. The first panel is partitioned as $A(:, 1:b) = [A_{00}\ ;\ A_{10}\ ;\ A_{20}\ ;\ A_{30}]$. Each processor p has associated

a block row A_{p0}. At the first step of the reduction, b rows are selected from each block A_{p0} by using GEPP. The selected rows correspond to the pivot rows used during the LU factorization. After this step we obtain 4 sets of b candidate rows. In the second step, the sets are combined two by two, we obtain two matrices of size $2b \times b$ each. From each matrix we select b rows by using again GEPP. In the last step of tournament pivoting, the two sets of candidate rows form a new matrix of size $2b \times b$ from which the final b rows are selected. This algorithm is illustrated in figure 2 from [40], where the function $f(A_{ij})$ computes the GEPP factorization of A_{ij} and returns the b pivot rows used by partial pivoting. The input matrix A_{ij} of dimension $2b \times b$ is formed by the two sets of candidate rows selected by the previous steps of tournament pivoting.

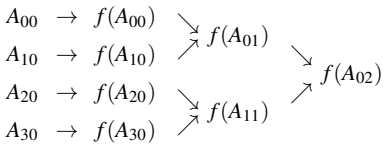


Fig. 2 TSLU with binary tree based tournament pivoting. This figure is from [40]. Copyright

Algorithm 3 Parallel TSLU factorization

Require: P processors, i is my processor's index, all reduction tree with height $L = \log P$
Require: $A \in \mathbb{R}^{m \times n}, m \gg n$, distributed in block row layout; $A_{i,0}$ is the block of rows belonging to my processor i
1: Compute $\Pi_{i,0} A_{i,0} = L_{i,0} U_{i,0}$ using GEPP
2: **for** each level k in the reduction tree from 1 to L **do**
3: $s = \lfloor i/2^k \rfloor$, $f = 2^k \lfloor i/2^k \rfloor$, $j = f + (i + 2^{k-1}) \mod 2^k$
4: $s_i = \lfloor i/2^{k-1} \rfloor$, $s_j = \lfloor j/2^{k-1} \rfloor$
5: Non-blocking send $(\Pi_{s_i,k-1} A_{s_i,k-1})(1:n, 1:n)$ to processor j
6: Non-blocking receive $(\Pi_{s_j,k-1} A_{s_j,k-1})(1:n, 1:n)$ from processor j
7: Wait until the previous send and receive have completed
8: Form the matrix $A_{s,k}$ of size $2n \times n$ as $A_{s,k} = \begin{bmatrix} (\Pi_{s_i,k-1} A_{s_i,k-1})(1:n, 1:n) \\ (\Pi_{s_j,k-1} A_{s_j,k-1})(1:n, 1:n) \end{bmatrix}$
9: Compute $\Pi_{s,k} A_{s,k} = L_{s,k} U_{s,k}$ using GEPP
10: **end for**
11: Determine the final permutation Π, such that $(\Pi A)(1:n, :)$ are the k selected rows at the end of tournament
12: All P processors compute the Gaussian elimination with no pivoting of their blocks, $\Pi A = LU$
Ensure: $U_{0,L}$ is the U factor obtained at step 12 for all processors i.

Algorithm 3 presents a pseudo-code for the parallel implementation of TSLU on P processors. It follows the presentation of TSLU in [40], where a more detailed description can be found. For simplicity, we consider that P is a power of 2. We consider here an all reduction tree based on a butterfly scheme, whose height is

$L = \log P$. The matrix A is distributed block row-wise over processors. The levels of the tree are numbered from 0 to L, where the first level 0 corresponds to the phase with no communication and each leaf node represents a processor. At the first level $k = 1$, each node s has associated two processors i and $i-1$, where i is an odd number. The two processors exchange their set of candidate rows. Then each processor forms a matrix with the two sets of candidate rows and selects a new set of candidate rows using GEPP. In general, at a given level k, processor i participates to the computation associated with node numbered $s = \lfloor i/2^k \rfloor$. The first processor associated with this node is $f = 2^k \lfloor i/2^k \rfloor$ and the processor exchanging information with this processor is numbered $f + 2^{k-1}$. Processor i exchanges information with processor $j = f + (i + 2^{k-1}) \mod 2^k$. They exchange the candidate rows that were selected at the previous level $k-1$ in the reduction tree at the children nodes s_i and s_j.

TSLU requires exchanging $\log P$ messages among processors. This allows the overall CALU algorithm to attain the lower bounds on communication in terms of both number of messages and volume of communication. When the LU factorization of a matrix of size $n \times n$ is computed by using CALU on a grid of $P = P_r \times P_c$ processors, as shown in [40] where a more detailed description can be found, the parallel performance of CALU in terms of number of messages, volume of communication, and flops, is

$$\begin{aligned} T_{CALU}(m,n,P) \approx \gamma \cdot & \left(\frac{1}{P}\left(mn^2 - \frac{n^3}{3}\right) + \frac{1}{P_r}\left(2mn - n^2\right) b + \frac{n^2 b}{2P_c} + \frac{nb^2}{3}(5\log_2 P_r - 1) \right) \\ & + \beta \cdot \left(\left(nb + \frac{3n^2}{2P_c} \right) \log_2 P_r + \frac{1}{P_r}\left(mn - \frac{n^2}{2} \right) \log_2 P_c \right) \\ & + \alpha \cdot \left(\frac{3n}{b} \log_2 P_r + \frac{3n}{b} \log_2 P_c \right). \end{aligned} \tag{8}$$

To attain the lower bounds on communication, an optimal layout can be chosen with $P_r = P_c = \sqrt{P}$ and $b = \log^{-2}\left(\sqrt{P}\right) \cdot \frac{n}{\sqrt{P}}$. The blocking parameter b is chosen such that the number of messages attains the lower bound on communication from equation (4), while the number of flops increases only by a lower order term. With this layout, the performance of CALU becomes,

$$\begin{aligned} T_{CALU}(m,n,P = \sqrt{P} \times \sqrt{P}) \approx \gamma \cdot & \left(\frac{1}{P}\frac{2n^3}{3} + \frac{5n^3}{2P\log^2 P} + \frac{5n^3}{3P\log^3 P} \right) \\ & + \beta \cdot \frac{n^2}{\sqrt{P}} \left(2\log^{-1} P + 1.25 \log P \right) \\ & + \alpha \cdot 3\sqrt{P}\log^3 P. \end{aligned} \tag{9}$$

We note that GEPP as implemented for example in ScaLAPACK (`PDGETRF` routine) has the same volume of communication as CALU, but requires exchanging a factor on the order of b more messages than CALU.

3.2 Pivoting strategies and numerical stability

The backward stability of the LU factorization depends on the growth factor g_W, defined as,

$$g_W = \frac{\max_{i,j,k} |A^{(k)}(i,j)|}{\max_{ij} |A(i,j)|}, \tag{10}$$

where $A^{(k)}(i,j)$ denotes the entry in position (i,j) obtained after k steps of elimination. This is illustrated by the following Lemma 1.

Lemma 1 (Lemma 9.6, section 9.3 of [49]). *Let $A = LU$ be the Gaussian elimination without pivoting of A. Then $\||L||U|\|_\infty$ is bounded using the growth factor g_W by the relation $\||L||U|\|_\infty \leq (1+2(n^2-n)g_W)\|A\|_\infty$.*

A comparison of the upper bound of the growth factors obtained by different pivoting strategies is given in Table 1. All the results discussed in this section hold in exact arithmetic. The growth factor of CALU is obtained by using the fact that performing CALU on a matrix A is equivalent with performing GEPP on a larger matrix formed by blocks from the original matrix A and blocks of zeros. In addition to partial pivoting (GEPP) and CALU, we also include in this table the growth factor of the LU factorization with panel rank revealing pivoting (LU_PRRP) and its communication avoiding version (CALU_PRRP), presented in [55]. We observe that the upper bound of the growth factor is larger for CALU than for GEPP. However many experiments presented in [40] show that in practice CALU is as stable as GEPP. There is one particular case of nearly singular matrices in which CALU can lead to a large growth factor, and a solution to this case is presented in a paper in preparation [23].

Table 1 Bounds for the growth factor g_W obtained from different pivoting strategies for a matrix of size $m \times n$. CALU_PRRP and LU_PRRP select pivots using strong rank revealing QR (that uses a parameter τ typically equal to 2). The reduction tree used during tournament pivoting is of height $\log P$.

	CALU	GEPP	CALU_PRRP	LU_PRRP
Upper bound	$2^{n(\log P+1)-1}$	2^{n-1}	$(1+\tau b)^{(n/b-1)\log P} \cdot 2^{b-1}$	$(1+\tau b)^{n/b-1} \cdot 2^{b-1}$

3.3 Selection of references for LU factorization

The LU factorization has been largely studied in the litterature, and we give here only several references. One of the first references (if not the first) to a block algorithm is [10], a paper by Barron and Swinnerton-Dyer. The authors were interested in solving a linear system of equations on EDSAC 2 computer, by using a

magnetic-tape store. Hence they were interested in using as much as possible the data in main store, and reduce the number of transfers between magnetic tape and main store. They introduce two algorithms, the first one uses a pivoting strategy referred to nowadays as pairwise pivoting, the second one is the block LU factorization presented at the beginning of this section. The numerical stability of the LU factorization is studied for example in [49, 50, 69, 71]. Techniques as pairwise pivoting and block pivoting are studied in [65, 69]. In [69] it is shown experimentally that two factors are important for the numerical stability of the LU factorization, the elements of L are bounded in absolute value by a small number and the correction introduced at each step of the factorization is of rank 1. The latter property is satisfied by GEPP, CALU, LU_PRRP, and CALU_PRRP. Pairwise pivoting, parallel pivoting and their block versions do not satisfy this property, and block parallel pivoting can lead to an exponential growth factor [69]. As shown in [40], for matrices with more than 2^{12} rows and columns, block pairwise pivoting leads to a growth of g_W which is faster than linear. Potentially this pivoting strategy can become unstable for very large matrices.

4 Communication avoiding QR factorization

The QR factorization decomposes a matrix $A \in \mathbb{R}^{m\times n}$ as $A = QR$, where $Q \in \mathbb{R}^{m\times m}$ is orthogonal and $R \in \mathbb{R}^{m\times n}$. We can further decompose the factors into $Q_1 \in \mathbb{R}^{m\times n}$, $Q_2 \in \mathbb{R}^{m\times(m-n)}$, and the upper triangular matrix $R_1 \in \mathbb{R}^{n\times n}$ to obtain the factorization

$$A = QR = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1. \tag{11}$$

If A is full rank, the thin factorization Q_1R_1 is unique (modulo signs of diagonal elements of R). We consider in this document the QR factorization based on Householder transformations. Algorithm 4 presents such a factorization. A Householder transformation is a symmetric and orthogonal matrix of the form $H = I - \frac{2}{y^T y} yy^T$, which is independent of the scaling of the vector y. When applied to a vector x, it reflects x through the hyperplane $\text{span}(y)^{\perp}$.

At each iteration k of the QR factorization from Algorithm 4, the Householder matrix $H_k = I - \tau_k y_k y_k^T$ is chosen such that all the elements of $A(k:m,k)$ are annihilated, except the first one, $H_k A(k:m,k) = \pm\|A(k:m,k)\|_2 e_1$. For more details on how to compute the Householder matrix, the reader can refer to [35, 49] or to the LAPACK implementation [1]. We obtain

$$\begin{aligned} Q^T A &= H_n H_{n-1} \ldots H_1 A = R, \\ Q &= (I - \tau_1 y_1 y_1^T) \ldots (I - \tau_n y_n y_n^T). \end{aligned}$$

A block version of this algorithm can be obtained by using a storage efficient representation of Q [64],

Algorithm 4 QR factorization based on Householder transformations

Require: $A \in \mathbb{R}^{m\times n}$

1: Let $R \in \mathbb{R}^{n\times n}$ be initialized with zero matrix and $Y \in \mathbb{R}^{m\times n}$ with identity matrix
2: **for** k = 1 to n **do**
▷ Compute Householder matrix $H_k = I - \tau_k y_k y_k^T$ s.t. $H_k A(k:m,k) = \pm\|A(k:m,k)\|_2 e_1$. Store y_k in $Y()$ and τ_k in $\mathscr{T}(k)$
3: $R(k,k) = -sgn(A(k,k)) \cdot \|A(k:m,k)\|_2$
4: $Y(k+1:m,k) = \frac{1}{R(k,k)-A(k,k)} \cdot A(k+1:m,k)$ ▷ vector y_k
5: $\mathscr{T}(k) = \frac{R(k,k)-A(k,k)}{R(k,k)}$ ▷ scalar τ_k
▷ Update trailing matrix $A(k:m,k+1:n)$
6: $A(k:m,k+1:n) = (I - Y(k+1:m,k)\mathscr{T}(k)Y(k+1:m,k)^T) \cdot A(k:m,k+1:n)$
7: $R(k,k+1:n) = A(k,k+1:n)$
8: **end for**

Ensure: $A = QR$, where $Q = H_1 \dots H_n = (I - \tau_1 y_1 y_1^T) \dots (I - \tau_n y_n y_n^T)$, the Householder vectors y_k are stored in Y, and $\mathscr{T}$ is an array of size n.

$$Q = (I - \tau_1 y_1 y_1^T) \dots (I - \tau_n y_n y_n^T) = I - YTY^T, \tag{12}$$

where Y is the matrix containing the Householder vectors as obtained in Algorithm 4 and T is computed from Y and the scalars τ_k stored in $\mathscr{T}$. As example, for $n = 2$, the compact representation is obtained as follows,

$$Y = [y_1, y_2], \quad T = \begin{bmatrix} \tau_1 & -\tau_1 y_1^T y_2 \tau_2 \\ 0 & \tau_2 \end{bmatrix}.$$

The product of two compact representations can be represented by one compact representation as follows [30],

$$\begin{aligned} Q &= (I - Y_1 T_1 Y_1^T)(I - Y_2 T_2 Y_2^T) = (I - YTY^T), \\ Y &= [Y_1, Y_2], \\ T &= \begin{bmatrix} T_1 & -T_1 Y_1^T Y_2 T_2 \\ 0 & T_2 \end{bmatrix}. \end{aligned}$$

A block algorithm for computing the QR factorization is obtained by partitioning the matrix A of size $m \times n$ as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \tag{13}$$

where A_{11} is of size $b \times b$, A_{21} is of size $(m-b) \times b$, A_{12} is of size $b \times (n-b)$, and A_{22} is of size $(m-b) \times (n-b)$. The first step of the block QR factorization algorithm computes the QR factorization of the first b columns $[A_{11};A_{21}]$ to obtain the following factorization,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = Q_1 \begin{bmatrix} R_{11} & R_{12} \\ & A_{22}^1 \end{bmatrix}.$$

The algorithm continues recursively on the trailing matrix A_{22}^1. The algebra of block QR factorization is presented in Algorithm 5.

Algorithm 5 QR factorization based on Householder transformations using a block algorithm

Require: $A \in \mathbb{R}^{m\times n}$
1: Let $R \in \mathbb{R}^{m\times n}$ be initialized with zero matrix
2: **for** k = 1 to n/b **do**
3: $k_b = (k-1)\cdot b+1$, $k_e = k_b + b - 1$
4: Compute by using Algorithm 4 the factorization

$$A(k_b : m, k_b : k_e) = Q_k R(k_b : k_e, k_b : k_e)$$

5: Compute the compact representation $Q_k = I - Y_k T_k Y_k^T$
6: Apply Q_k^T on the trailing matrix

$$\begin{aligned} A(k_b : m, k_e+1 : n) &= (I - Y_k T_k^T Y_k^T) A(k_b : m, k_e+1 : n) \\ &= A(k_b : m, k_e+1 : n) - Y_k \left(T_k^T \left(Y_k^T \left(A(k_b : m, k_e+1 : n) \right) \right) \right) \end{aligned}$$

7: $R(k_b : k_e, k_e+1 : n) = A(k_b : k_e, k_e+1 : n)$
8: **end for**
Ensure: $A = QR$, where $Q = (I - Y_1 T_1 Y_1^T) \ldots (I - Y_{n/b} T_{n/b} Y_{n/b}^T)$

A parallel implementation of the QR factorization as implemented in ScaLAPACK, `PDGEQRF` routine, considers that the matrix A is distributed over a grid of processors $P = P_r \times P_c$. We do not describe here in detail the parallel algorithm. We note that similarly to the LU factorization, the latency bottleneck lies in the QR factorization of each panel, that is based on Algorithm 4. The computation of a Householder vector at each iteration k of Algorithm 4 requires computing the norm of column k. Given that the columns are distributed over P_r processors, computing the norm of each column requires a reduction among P_r processors. Hence overall a number of messages proportional to the number of columns of A needs to be exchanged during `PDGEQRF`. Such an algorithm cannot attain the lower bounds on the number of messages. We note however that `PDGEQRF` attains the lower bound on the volume of communication.

4.1 Communication avoiding QR factorization for a tall and skinny matrix: TSQR

Consider a matrix $A \in \mathbb{R}^{m\times n}$ for which $m \gg n$. TSQR is a QR factorization algorithm that allows to minimize communication between different processors or between different levels of the memory hierarchy. It is performed as a reduction operation, in which the operator used at each step of the reduction is a QR factorization. We describe here the parallel case, for more details the reader is referred to [21]. We

assume that the matrix A is distributed over P processors by using a block row distribution. We consider in the following that $P = 4$, m is a multiple of 4, and the matrix A is partitioned among processors as,

$$A = \begin{bmatrix} A_{00} \\ A_{10} \\ A_{20} \\ A_{30} \end{bmatrix}, \tag{14}$$

where $A_{i0}, i = 0, \ldots, 3$ is of dimension $m/4 \times n$. At the first step of TSQR, each processor computes locally a QR factorization,

$$A = \begin{bmatrix} A_{00} \\ A_{10} \\ A_{20} \\ A_{30} \end{bmatrix} = \begin{bmatrix} Q_{00}R_{00} \\ Q_{10}R_{10} \\ Q_{20}R_{20} \\ Q_{30}R_{30} \end{bmatrix} = \begin{bmatrix} Q_{00} & & & \\ & Q_{10} & & \\ & & Q_{20} & \\ & & & Q_{30} \end{bmatrix} \begin{bmatrix} R_{00} \\ R_{10} \\ R_{20} \\ R_{30} \end{bmatrix}. \tag{15}$$

At the second step, the upper triangular factors $R_{i0}, i = 1:4$ are grouped into pairs, and each pair is factored in parallel as,

$$\begin{bmatrix} R_{00} \\ \hline R_{10} \\ \hline R_{20} \\ R_{30} \end{bmatrix} = \begin{bmatrix} Q_{01}R_{01} \\ Q_{11}R_{11} \end{bmatrix} = \begin{bmatrix} Q_{01} & \\ & Q_{11} \end{bmatrix} \begin{bmatrix} R_{01} \\ R_{11} \end{bmatrix}. \tag{16}$$

At the last step the resulting upper triangular factors are factored as,

$$\begin{bmatrix} R_{01} \\ R_{11} \end{bmatrix} = Q_{02}R_{02}. \tag{17}$$

The QR factorization obtained by TSQR based on a binary tree is,

$$A = QR_{02}, \tag{18}$$

where

$$Q = \begin{bmatrix} Q_{00} & & & \\ & Q_{10} & & \\ & & Q_{20} & \\ & & & Q_{30} \end{bmatrix} \cdot \begin{bmatrix} Q_{01} & \\ & Q_{11} \end{bmatrix} \cdot Q_{02}. \tag{19}$$

The matrix Q is an orthogonal matrix formed by the product of three orthogonal matrices (the dimensions of the intermediate factors are chosen such that their product can be written as above). Unless it is required, the matrix Q is not formed explicitely, but it is stored implicitely. The QR factorization used at each step of TSQR can be performed by using Algorithm 4 or any other efficient sequential QR factorization (as recursive QR [30]).

By using an arrow notation similar to CALU, a binary tree based parallel TSQR factorization is represented in Figure 3. Algorithm 6 presents parallel TSQR by fol-

lowing its presentation from [21]. The notation used for the nodes and the levels of the all reduction tree is the same as in Algorithm 3. It can be easily seen that parallel TSQR requires exchanging only $\log P$ messages, and thus it minimizes communication. It exchanges the same volume of communication as the ScaLAPACK implementation of Householder QR (`PDGEQR2` routine), $(n^2/2)\cdot\log P$ words. In terms of floating point operations, TSQR performs $2mn^2/P+(2n^3/3)\cdot\log P$ flops, while `PDGEQR2` performs $2mn^2/P-(2n^3)/(3P)$ flops.

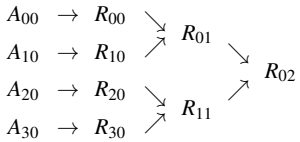


Fig. 3 Binary tree based TSQR. This figure is from [24]. Copyright ©[2012] Society for Industrial and Applied Mathematics. Reprinted with permission. All rights reserved.

Algorithm 6 Parallel TSQR factorization

Require: P processors, i is my processor's index, all reduction tree with height $L=\log P$
Require: $A\in\mathbb{R}^{m\times n}, m\gg n$, distributed in a block row layout; $A_{i,0}$ is the block of rows belonging to my processor i
1: Compute QR factorization $A_{i,0}=Q_{i,0}R_{i,0}$
2: **for** each level k in the reduction tree from 1 to L **do**
3: $s=\lfloor i/2^k\rfloor$, $f=2^k\lfloor i/2^k\rfloor$, $j=f+(i+2^{k-1}) \mod 2^k$
4: $s_i=\lfloor i/2^{k-1}\rfloor$, $s_j=\lfloor j/2^{k-1}\rfloor$
5: Non-blocking send $R_{s_i,k-1}$ to processor j
6: Non-blocking receive $R_{s_j,k-1}$ from processor j
7: Wait until the previous send and receive have completed
8: Compute $\begin{bmatrix}R_{s_i,k-1}\\R_{s_j,k-1}\end{bmatrix}=Q_{s,k}R_{s,k}$
9: **end for**
Ensure: $A=QR_{0,L}$, $R_{0,L}$ is available on all processors i
Ensure: Q is implicitly represented by the intermediate Q factors $\{Q_{s,k}\}$, for each node s and each level k in the all reduction tree

We note also that it is possible to reconstruct the Householder vectors of the classic Householder QR factorization (Algorithm 4) from TSQR. Let $A=QR$ be the factorization obtained from Householder QR, where A is of size $m\times n$, and let

$$Q=I-YTY^T=I-\begin{bmatrix}Y_1\\Y_2\end{bmatrix}T\begin{bmatrix}Y_1^TY_2^T\end{bmatrix} \tag{20}$$

be the compact representation of Q, where Q is of size $m\times m$. Let $Q=[Q_1,Q_2]$, where Q_1 is formed by the first n columns of Q. This is also called a *basis-kernel* representation of an orthogonal matrix, and as described in [67], there are several

different basis-kernel representations. The reconstruction of Householder vectors introduced in [4] relies on the observation that

$$Q_1 - S = Y(-TY_1^T), \tag{21}$$

where S is a sign matrix which reflects the sign choice of the diagonal of R made in line 3 of Algorithm 4. Since Y is unit lower triangular and $(-TY_1^T)$ is upper triangular, this represents the unique LU decomposition of $Q_1 - S$. In other words, Y and T can be reconstructed by computing the LU decomposition of $Q_1 - S$. With this approach, denoted as TSQR-HR in [4], the performance of the algorithm becomes:

$$T_{TSQR-HR}(m,n,P) = \gamma \cdot \left(\frac{4mn^2}{P} + \frac{4n^3}{3} \log P \right) + \beta \cdot n^2 \log P + \alpha \cdot 2 \log P. \tag{22}$$

We note that this algorithm performs 2.5 times more floating point operations than TSQR. However, in practice it leads to a faster algorithm than `PDGEQR2`, as shown in [4]. It can also be used to obtain a communication avoiding QR factorization by only replacing the panel factorization in `PDGEQRF`. Faster approaches are possible, but they could be less stable. For example the Householder vectors can be reconstructed from the LU factorization of $A - R$, and this approach is stable when A is well conditioned.

4.2 Communication avoiding QR factorization

We consider now the case of general matrices. CAQR was introduced in [21] and it relies on using TSQR for its panel factorization. Each QR factorization performed during TSQR induces an update of the trailing matrix. Hence the update of the trailing matrix is driven by the reduction tree used during TSQR. CAQR exchanges the same volume of communication as `PDGEQRF`. But the number of messages with an optimal layout is $(3/8)\sqrt{P}\log^3 P$ for CAQR, while for `PDGEQRF` is $(5n/4)\log^2 P$. The number of floating point operations remains the same (only lower order terms change).

Another approach [4] consists in reconstructing the Householder vectors from TSQR. A communication avoiding version can be obtained by replacing the panel factorization in a classic algorithm such that the update of the trailing matrix does not change. This leads to a simpler algorithm to implement, and better performance on parallel machines, as described in [4].

5 Communication avoiding rank revealing factorization and low rank matrix approximation

In this section we consider the problem of estimating the singular values of a matrix or computing its numerical rank, a problem with many diverse applications in both scientific computing and data analytics, a detailed description can be found in [16]. One such application is computing the rank-k approximation $\tilde{A}_k$ of a matrix $A \in \mathbb{R}^{m\times n}$, $\tilde{A}_k = ZW^T$, where $Z \in \mathbb{R}^{m\times k}$, $W^T \in \mathbb{R}^{k\times n}$, and k is much smaller than m and n. Very often, this low rank approximation is used in the context of an iterative process which involves multiplying a matrix with a vector. Hence instead of computing the product Ax, which requires computing $2mn$ flops when A is dense, one could compute the product ZW^Tx with $2(m+n)k$ flops.

The best rank-k approximation of A is the rank-k truncated singular value decomposition (SVD) of A. The singular value decomposition of A is

$$A = U\Sigma V^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \cdot \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \cdot \begin{bmatrix} V_1 & V_2 \end{bmatrix}^T,$$

where U is $m \times m$ orthogonal matrix, the left singular vectors of A, U_1 is formed by the first k vectors, U_2 is formed by the last $m-k$ vectors. Σ is of dimension $m \times n$, its diagonal is formed by $\sigma_1(A) \geq \ldots \geq \sigma_n(A)$, Σ_1 is of dimension $k \times k$ and contains the first k singular values, Σ_2 is of dimension $(m-k) \times (n-k)$ and contains the last $n-k$ singular values. V is $n \times n$ orthogonal matrix, the right singular vectors of A, V_1 is formed by the first k vectors, V_2 is formed by the last $n-k$ vectors. The rank-k truncated singular value decomposition of A is $A_k = U_1\Sigma_1 V_1^T$. Eckart and Young [29] have shown that

$$\min_{\text{rank}(\tilde{A}_k)\leq k} ||A-\tilde{A}_k||_2 = ||A-A_k||_2 = \sigma_{k+1}(A), \tag{23}$$

$$\min_{\text{rank}(\tilde{A}_k)\leq k} ||A-\tilde{A}_k||_F = ||A-A_k||_F = \sqrt{\sum_{j=k+1}^{n} \sigma_j^2(A)}. \tag{24}$$

Since computing the SVD of a matrix is very expensive, several different approaches exist in the literature to approximate the singular value decomposition which trade-off accuracy for speed. Those include the Lanczos algorithm [18, 62], rank revealing factorizations as the rank revealing QR or LU factorizations, and more recently randomized algorithms. For an overview of randomized algorithms the reader can refer to [57].

5.1 Rank revealing QR factorization

In this section we consider the rank revealing QR factorization based on QR factorization with column pivoting. Given a matrix $A \in \mathbb{R}^{m\times n}$, its QR factorization with

column pivoting is

$$A\Pi_c = QR = Q \begin{bmatrix} R_{11} & R_{12} \\ & R_{22} \end{bmatrix}, \tag{25}$$

where Π_c is a column permutation matrix, $Q \in \mathbb{R}^{m\times m}$ is orthogonal, $R_{11} \in \mathbb{R}^{k\times k}$ is upper triangular, $R_{12} \in \mathbb{R}^{k\times(n-k)}$, $R_{22} \in \mathbb{R}^{(m-k)\times(n-k)}$. We say that this is a rank revealing factorization (RRQR) if the column permutation matrix Π_c is chosen such that

$$1 \leq \frac{\sigma_i(A)}{\sigma_i(R_{11})}, \frac{\sigma_j(R_{22})}{\sigma_{k+j}(A)} \leq q(k,n), \tag{26}$$

for any $1 \leq i \leq k$ and $1 \leq j \leq \min(m,n)-k$, where $q(k,n)$ is a low degree polynomial in n and k, and $\sigma_1(A) \geq \ldots \geq \sigma_n(A)$ are the singular values of A (we assume in this document that the singular values of A and R are all nonzero). In other words, the column permutation allows to identify a submatrix of k columns whose singular values provide a good approximation of the largest k singular values of A, while the singular values of R_{22} provide a good approximation of the $\min(m,n)-k$ smallest singular values of A. If $||R_{22}||_2$ is small and since $\sigma_{k+1}(A) \leq \sigma_{\max}(R_{22}) = ||R_{22}||_2$, then the numerical rank of A is k. Then $Q(:,1:k)$ forms an approximate orthogonal basis for the range of A. Since $A\Pi_c \begin{bmatrix} R_{11}^{-1}R_{12} \\ -I \end{bmatrix} = Q \begin{bmatrix} 0 \\ -R_{22} \end{bmatrix}$ then $\Pi_c \begin{bmatrix} R_{11}^{-1}R_{12} \\ -I \end{bmatrix}$ are approximate null vectors.

The usage of a QR factorization to reveal the rank of a matrix was introduced in [34] and the first algorithm to compute it was introduced in [13]. With this algorithm, the absolute value of the entries of $R_{11}^{-1}R_{12}$ is bounded by $O(2^k)$ and it might fail sometimes to satisfy (26), for example on the so-called Kahan matrix [54]. However, in most cases it provides a good approximation to the SVD and it is the method of choice for estimating the singular values of a matrix through a pivoted QR factorization. We refer to this algorithm as QRCP, which stands for QR with Column Pivoting. It chooses at each step of the QR factorization the column of maximum norm and permutes it to the leading position before proceeding with the factorization.

The *strong RRQR factorization* was introduced in [45]. For a given k and a parameter $f > 1$, the results in [45] show that there exists a permutation Π_c such that

$$\left(R_{11}^{-1}R_{12}\right)_{i,j}^2 + \omega_i^2(R_{11})\,\chi_j^2(R_{22}) \leq f^2, \tag{27}$$

for any $1 \leq i \leq k$ and $1 \leq j \leq n-k$, where $\omega_i(R_{11})$ denotes the 2-norm of the i-th row of R_{11}^{-1} and $\chi_j(R_{22})$ denotes the 2-norm of the j-th column of R_{22}. This inequality bounds the absolute values of the elements of $R_{11}^{-1}R_{12}$ and leads to the following bounds on singular values.

Theorem 1. *(Gu and Eisenstat [45]) Let the factorization in equation* (25) *satisfy inequality* (27)*. Then*

$$1 \leq \frac{\sigma_i(A)}{\sigma_i(R_{11})}, \frac{\sigma_j(R_{22})}{\sigma_{k+j}(A)} \leq \sqrt{1 + f^2 k(n-k)}, \tag{28}$$

for any $1 \leq i \leq k$ *and* $1 \leq j \leq \min(m,n) - k$, .

A strong RRQR factorization can be obtained by computing first a QR factorization with column pivoting to choose a rank k. For this rank k and a given f, additional permutations are performed until the inequality in (27) is satisfied, for a cost of $O(mnk)$ floating point operations [45].

When executed on a distributed memory computer, the matrix A is distributed over a 2D grid of processors $P = P_r \times P_c$. Finding the column of maximum norm at each step of QRCP requires a reduction operation among P_r processors, which costs $O(\log P_r)$ messages. After k steps of factorization, this requires exchanging $O(k \cdot \log P_r)$ messages. Hence, when run to completion, QRCP and its strong variant cannot attain the lower bound on communication $\Omega(\sqrt{P})$.

5.2 Tournament pivoting for selecting a set of k columns

A communication avoiding rank revealing QR factorization, referred to as CARRQR, was introduced in [20]. This factorization is based on tournament pivoting, and performs a block algorithm which computes the factorization by traversing blocks of k columns (where k is small). At each iteration, it selects k columns that are as well conditioned as possible by using a tournament which requires only $O(\log P_r)$ messages. The selected columns are permuted to the leading positions before the algorithm computes k steps of a QR factorization with no more pivoting.

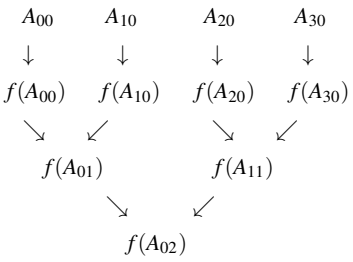


Fig. 4 Binary tree based QR factorization with tournament pivoting. This figure is from [20].

Algorithm 7 describes the selection of k columns from a matrix A by using binary tree based tournament pivoting. This selection is displayed in figure 4, in which the matrix A is partitioned into 4 subsets of columns, $A = [A_{00}, A_{10}, A_{20}, A_{30}]$. At the leaves of the reduction tree, for each subset of columns A_{0j}, $f(A_{0j})$ selects k columns by using strong rank revealing QR factorization of A_{0j}. Then at each node of the reduction tree, a new matrix A_{ij} is obtained by adjoining the columns selected

by the children of the node, and $f(A_{ij})$ selects k columns by using strong rank revealing QR factorization of A_{ij}.

Algorithm 7 QR_TP (A,k): Select k linearly independent columns from a matrix A by using QR factorization with binary tree based tournament pivoting

Require: $A \in \mathbb{R}^{m\times n}$, number of columns to select k

1: Partition the matrix $A = [A_{00},\ldots,A_{n/k,0}]$, where $A_{i0} \in \mathbb{R}^{m\times 2k}$, $i = 1,\ldots n/(2k)$ // Assume n is a multiple of $2k$
2: **for** each level in the reduction tree $j = 0$ to $\log_2 n/(2k) - 1$ **do**
3: **for** each node i in the current level j **do**
4: **if** $j = 0$ (at the leaves of the reduction tree) **then**
5: A_{i0} is the i-th block of $2k$ columns of A
6: **else** Form A_{ij} by putting next to each other the two sets of k column candidates selected by the children of node j
7: **end if**
8: Select k column candidates by computing $A_{ij} = Q_1 R_1$ and then computing a RRQR factorization of R_1, $R_1 P_{c_2} = Q_2 \begin{bmatrix} R_2 & * \\ & * \end{bmatrix}$
9: **if** j is the root of the reduction tree **then**
10: Return Π_c such that $(A\Pi_c)(:,1:k) = (A_{ij}\Pi_{c_2})(:,1:k)$
11: **else** Pass the k selected columns, $A\Pi_{c_2}(:,1:k)$ to the parent of i
12: **end if**
13: **end for**
14: **end for**

Ensure: Π_c such that $(A\Pi_c)(:,1:k)$ are the k selected columns

It is shown in [20] that the factorization as in equation (25) computed by CARRQR satisfies the inequality

$$\chi_j^2\left(R_{11}^{-1}R_{12}\right) + \left(\chi_j\left(R_{22}\right)/\sigma_{\min}\left(R_{11}\right)\right)^2 \leq F_{TP}^2, \text{ for } j = 1,\ldots,n-k, \tag{29}$$

where $\chi_j(B)$ denotes the 2-norm of the j-th column of B. This inequality is very similar to the one characterizing a strong RRQR factorization. The following Theorem 2 shows that CARRQR reveals the rank by satisfying an inequality similar to (27), where the constant f is replaced by F_{TP}, a quantity which depends on the number of columns n, the rank k, and the depth of the tree used during tournament pivoting. More details can be found in [20].

Theorem 2. *Assume that there exists a permutation Π_c for which the QR factorization*

$$A\Pi_c = Q\begin{bmatrix} R_{11} & R_{12} \\ & R_{22} \end{bmatrix}, \tag{30}$$

where R_{11} is $k\times k$ and satisfies (29)*. Then*

$$1 \leq \frac{\sigma_i(A)}{\sigma_i(R_{11})}, \frac{\sigma_j(R_{22})}{\sigma_{k+j}(A)} \leq \sqrt{1+F_{TP}^2(n-k)}, \tag{31}$$

for any $1 \leq i \leq k$ and $1 \leq j \leq \min(m,n)-k$.

If only one step of QR with binary tree based tournament pivoting is used to select k columns of the $m \times n$ matrix A, Corollaries 2.6 and 2.7 from [20] show that the rank of A is revealed by satisfying inequality (31), with bound

$$F_{TP-BT} \leq \frac{1}{\sqrt{2k}} \left(\sqrt{2} f k\right)^{\log_2(n/k)} = \frac{1}{\sqrt{2k}} (n/k)^{\log_2(\sqrt{2} f k)}. \tag{32}$$

Given that f is a small constant and k in general is small compared to n, this bound can be seen as a polynomial in n. If tournament pivoting uses a flat tree, then the bound becomes

$$F_{TP-FT} \leq \frac{1}{\sqrt{2k}} \left(\sqrt{2} f k\right)^{n/k}, \tag{33}$$

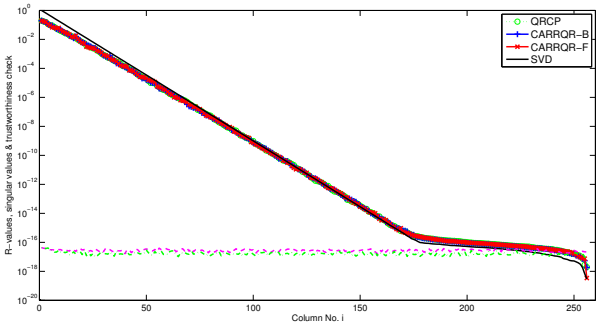
exponential in n/k. The exponent of both bounds has an additional factor on the order of n/k if multiple steps of QR with tournament pivoting are required to reveal the rank (which is hence larger than k). However, the extensive numerical experiments performed in [20] show that both binary tree and flat tree are effective in approximating the singular values of A. For a large set of matrices, the singular values approximated with CARRQR are within a factor of 10 of the singular values computed with the highly accurate routine `dgesvj` [27, 28]. Figure 5 shows the results obtained for two matrices (from [19]), EXPONENT, a matrix whose singular values follow an exponential distribution $\sigma_1 = 1$, $\sigma_i = \alpha^{i-1}$ $(i = 2, \ldots, n)$, $\alpha = 10^{-1/11}$ [11], and SHAW, a matrix from an 1D image restoration model [47]. The plots display the singular values computed by SVD and their approximations computed by QR factorizations with column permutations (given by the diagonal values of the R factor): QR with column pivoting (QRCP), CARRQR based on binary tree tournament pivoting (CARRQR-B), and flat tree tournament pivoting (CARRQR-F). The plots also display bounds for trustworthiness,

$$\varepsilon \min\{||(A\Pi_0)(:,i)||_2, ||(A\Pi_1)(:,i)||_2, ||(A\Pi_2)(:,i)||_2\} \tag{34}$$

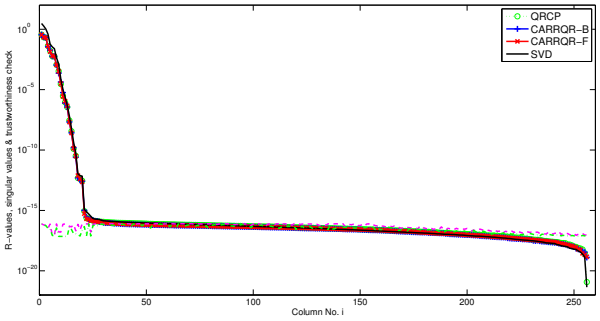
$$\varepsilon \max\{||(A\Pi_0)(:,i)||_2, ||(A\Pi_1)(:,i)||_2, ||(A\Pi_2)(:,i)||_2\} \tag{35}$$

where $\Pi_j (j = 0, 1, 2)$ are the permutation matrices obtained by QRCP, CARRQR-B, and CARRQR-F respectively, and ε is the machine precision. Those bounds display for each column an estimate of uncertainty in any entry of that column of R as computed by the three pivoting strategies.

On a distributed memory computer, CARRQR is implemented by distributing the matrix over a 2D grid of processors $P = P_r \times P_c$. By using an optimal layout, $P_r = \sqrt{mP/n}$, $P_c = \sqrt{nP/m}$, and $b = B \cdot \sqrt{mn/P}$, $B = 8^{-1} \log_2^{-1}(P_r) \log_2^{-1}(P_c)$, the overall performance of CARRQR (some lower order terms are ignored) is:



(a) EXPONENT



(b) SHAW

Fig. 5 Singular values as computed by SVD and approximations obtained from QRCP, CARRQR-B, and CARRQR-F.

$$
\begin{aligned}
T_{CARRQR}(m,n,P) \approx \gamma \cdot & \left(\frac{6mn^2 - 6n^3/3}{P} + cmn^2 \right) \\
& + \beta \cdot 2 \frac{\sqrt{mn^3}}{\sqrt{P}} \left(\log_2 \sqrt{\frac{mP}{n}} + \log_2 \sqrt{\frac{nP}{m}} \right) \\
& + \alpha \cdot 2^7 \sqrt{\frac{nP}{m}} \log_2^2 \sqrt{\frac{mP}{n}} \log_2^2 \sqrt{\frac{nP}{m}},
\end{aligned}
$$

where $c < 1$. This shows that parallel CARRQR performs three times more floating point operations than QRCP as implemented in ScaLAPACK (routine `pdgeqpf`), and it is communication optimal, modulo polylogarithmic factors.

5.3 Low rank matrix approximation for sparse matrices

In this section we focus on computing the low rank approximation of a sparse matrix by using rank revealing factorizations. In this case, the factors obtained by using Cholesky, LU, or QR factorizations have more nonzeros than the matrix A. The R factor obtained from the QR factorization is the Cholesky factor of A^TA, and since A^TA can be much denser than A, it is expected that its Cholesky factor has more nonzeros than the Cholesky factor of A. Hence, the QR factorization can lead to denser factors than the LU factorization. Similarly, a rank revealing QR factorization can be more expensive in terms of both memory usage and floating point operations than a rank revealing LU factorization. We present in the following LU_CRTP, a rank revealing LU factorization that also minimizes communication cost. A detailed presentation can be found in [38]. Given a desired rank k, the factorization is written as

$$\Pi_r A \Pi_c = \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \\ \bar{A}_{21} & \bar{A}_{22} \end{bmatrix} = \begin{bmatrix} I \\ \bar{A}_{21}\bar{A}_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \\ & S(\bar{A}_{11}) \end{bmatrix}, \tag{36}$$

where $A \in \mathbb{R}^{m\times n}$, $\bar{A}_{11} \in \mathbb{R}^{k,k}$, $S(\bar{A}_{11}) = \bar{A}_{22} - \bar{A}_{21}\bar{A}_{11}^{-1}\bar{A}_{12}$. The rank-k approximation matrix $\tilde{A}_k$ is

$$\tilde{A}_k = \begin{bmatrix} I \\ \bar{A}_{21}\bar{A}_{11}^{-1} \end{bmatrix} \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \end{bmatrix} = \begin{bmatrix} \bar{A}_{11} \\ \bar{A}_{21} \end{bmatrix} \bar{A}_{11}^{-1} \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} \end{bmatrix}. \tag{37}$$

The second formulation of $\tilde{A}_k$ from (37) is referred to as CUR decomposition (see [36, 57, 66] and references therein), since the first factor is formed by columns of A and the third factor is formed by rows of A. This decomposition is of particular interest for sparse matrices because its factors C and R remain sparse as the matrix A.

In LU_CRTP, the first k columns are selected by using QR with tournament pivoting of the matrix A. This leads to the factorization

$$A\Pi_c = Q \begin{bmatrix} R_{11} & R_{12} \\ & R_{22} \end{bmatrix} = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \\ & R_{22} \end{bmatrix}. \tag{38}$$

After tournament pivoting we have the QR factorization of the first k columns, $A(:,1:k) = Q(:,1:k)R_{11}$. The first k rows are then obtained by using QR factorization with tournament pivoting of the rows of the thin Q factor, $Q(:,1:k)^T$,

$$\Pi_r Q = \begin{bmatrix} \bar{Q}_{11} & \bar{Q}_{12} \\ \bar{Q}_{21} & \bar{Q}_{22} \end{bmatrix},$$

such that $||\bar{Q}_{21}\bar{Q}_{11}^{-1}||_{\max} \leq F_{TP}$ and bounds for the singular values of $\bar{Q}_{11}$ with respect to the singular values of Q are governed by a low degree polynomial. This leads to the factorization,

$$\begin{aligned}\Pi_r A \Pi_c &= \begin{bmatrix}\bar{A}_{11} & \bar{A}_{12}\\ \bar{A}_{21} & \bar{A}_{22}\end{bmatrix} = \begin{bmatrix} I \\ \bar{A}_{21}\bar{A}_{11}^{-1} & I\end{bmatrix}\begin{bmatrix}\bar{A}_{11} & \bar{A}_{12}\\ & S(\bar{A}_{11})\end{bmatrix}\\ &= \begin{bmatrix} I \\ \bar{Q}_{21}\bar{Q}_{11}^{-1} & I\end{bmatrix}\begin{bmatrix}\bar{Q}_{11} & \bar{Q}_{12}\\ & S(\bar{Q}_{11})\end{bmatrix}\begin{bmatrix}R_{11} & R_{12}\\ & R_{22}\end{bmatrix}\end{aligned} \tag{39}$$

where

$$\begin{aligned}\bar{Q}_{21}\bar{Q}_{11}^{-1} &= \bar{A}_{21}\bar{A}_{11}^{-1},\\ S(\bar{A}_{11}) &= S(\bar{Q}_{11})R_{22} = \bar{Q}_{22}^{-T}R_{22}.\end{aligned}$$

The following theorem from [38] shows that LU_CRTP (A,k) factorization reveals the singular values of A, and in addition also bounds the absolute value of the largest element of $S(\bar{A}_{11})$. This is important for the backward stability of the LU factorization.

Theorem 3 ([38]). *Let A be an $m \times n$ matrix. The LU_CRTP(A,k) factorization,*

$$\bar{A} = \Pi_r A \Pi_c = \begin{bmatrix}\bar{A}_{11} & \bar{A}_{12}\\ \bar{A}_{21} & \bar{A}_{22}\end{bmatrix} = \begin{bmatrix} I \\ \bar{Q}_{21}\bar{Q}_{11}^{-1} & I\end{bmatrix}\begin{bmatrix}\bar{A}_{11} & \bar{A}_{12}\\ & S(\bar{A}_{11})\end{bmatrix} \tag{40}$$

where

$$S(\bar{A}_{11}) = \bar{A}_{22} - \bar{A}_{21}\bar{A}_{11}^{-1}\bar{A}_{12} = \bar{A}_{22} - \bar{Q}_{21}\bar{Q}_{11}^{-1}\bar{A}_{12}, \tag{41}$$

satisfies the following properties

$$\rho_l(\bar{A}_{21}\bar{A}_{11}^{-1}) = \rho_l(\bar{Q}_{21}\bar{Q}_{11}^{-1}) \leq F_{TP}, \tag{42}$$

$$||S(\bar{A}_{11})||_{\max} \leq \min\left((1+F_{TP}\sqrt{k})||A||_{\max}, F_{TP}\sqrt{1+F_{TP}^2(m-k)}\sigma_k(A)\right) \tag{43}$$

$$1 \leq \frac{\sigma_i(A)}{\sigma_i(\bar{A}_{11})}, \frac{\sigma_j(S(\bar{A}_{11}))}{\sigma_{k+j}(A)} \leq q(m,n,k), \tag{44}$$

for any $1 \leq l \leq m-k$, $1 \leq i \leq k$, and $1 \leq j \leq \min(m,n)-k$. Here $\rho_l(B)$ denotes the 2-norm of the l-th row of B, F_{TP} is the bound obtained from QR with tournament pivoting, as in equation (32)*, and $q(m,n,k) = \sqrt{\left(1+F_{TP}^2(n-k)\right)\left(1+F_{TP}^2(m-k)\right)}$.*

The existence of a rank revealing LU factorization has been proven by Pan in [60], who shows that there are permutation matrices Π_r, Π_c such that the factorization from (36) satisfies

$$1 \leq \frac{\sigma_k(A)}{\sigma_{\min}(\bar{A}_{11})}, \frac{\sigma_{\max}(S(\bar{A}_{11}))}{\sigma_{k+1}(A)} \leq k(n-k)+1. \tag{45}$$

The existence of a stronger LU factorization has been proven by Miranian and Gu in [58], which in addition to (45) also upper bounds $||\bar{A}_{11}^{-1}\bar{A}_{12}||_{\max}$ by a low degree polynomial in k, n, and m. Pan also introduces two algorithms for computing such a factorization which are based on the notion of local maximum volume, where the

volume of a square matrix refers to the absolute value of its determinant. The first algorithm starts by performing LU factorization with conventional column pivoting (chooses as pivot the element of largest magnitude in the current row) followed by a block pivoting phase. The second algorithm relies on using the LU factorization of A^TA to perform symmetric pivoting. Experiments presented in [32] show that when there is a sufficiently large gap in the singular values of the matrix A, pivoting strategies as rook pivoting or complete pivoting produce good low rank approximations. However, they can fail for nearly singular matrices, as shown by examples given in [61].

The bounds on the approximation of singular values from (44) are worse than those from (45) showing the existence of a rank revealing LU factorization. However LU_CRTP is a practical algorithm that also minimizes communication. The bounds from (44) are also slightly worse than those obtained by CARRQR for which $q(m,n,k) = \sqrt{1+F_{TP}^2(n-k)}$ (see Theorem 2 for more details). But for sparse matrices, CARRQR requires significantly more computations and memory, as the experimental results in [38] show. A better bound than (44) can be obtained by using strong rank revealing QR for selecting the rows from the thin Q factor in equation (39), in which case $||\bar{A}_{21}\bar{A}_{11}^{-1}||_{\max} \leq f$, similar to the LU factorization with panel rank revealing pivoting from [56].

The bound on the growth factor from (43) is the minimum of two quantities. The first quantity has similarities with the bound on the growth factor obtained by the LU factorization with panel rank revealing pivoting from [56]. The second quantity is new and it relates the growth factor obtained after k steps of factorization to $\sigma_k(A)$.

Experiments reported in [38] show that LU_CRTP approximates well the singular values. For the matrices considered in that paper, the ratio of the singular values approximated by LU_CRTP to the singular values computed by SVD is at most 13 (and 27 for the devil's stairs, a more difficult matrix). Figure 6 [38] shows the results obtained for SHAW matrix, the 1D image restoration matrix also used in section 5.2.

References

1. E. ANDERSON, Z. BAI, C. BISCHOF, S. BLACKFORD, J. DEMMEL, J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, AND D. SORENSEN, *LAPACK Users' Guide*, SIAM, Philadelphia, PA, USA, 1999.
2. G. BALLARD, A. BULUC, J. DEMMEL, L. GRIGORI, O. SCHWARTZ, AND S. TOLEDO, *Communication optimal parallel multiplication of sparse random matrices*, in Proceedings of ACM SPAA, Symposium on Parallelism in Algorithms and Architectures, 2013.
3. G. BALLARD, J. DEMMEL, AND I. DUMITRIU, *Communication-optimal parallel and sequential eigenvalue and singular value algorithms*, Tech. Report EECS-2011-14, UC Berkeley, February 2011.
4. G. BALLARD, J. DEMMEL, L. GRIGORI, M. JACQUELIN, H. D. NGUYEN, AND E. SOLOMONIK, *Reconstructing Householder Vectors from Tall-Skinny QR*, in Proceedings of IEEE International Parallel and Distributed Processing Symposium IPDPS, 2014.
5. G. BALLARD, J. DEMMEL, O. HOLTZ, B. LIPSHITZ, AND O. SCHWARTZ, *Brief announcement: strong scaling of matrix multiplication algorithms and memory-independent communi-*

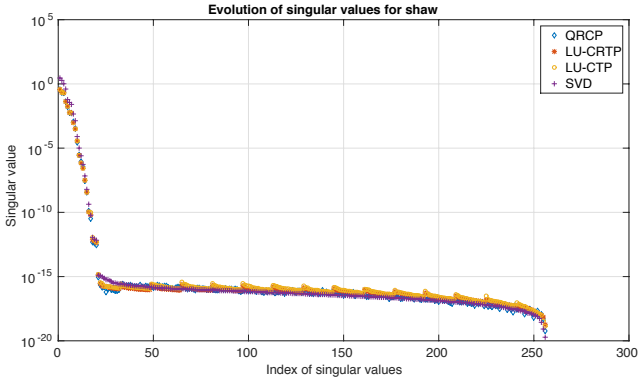


Fig. 6 Singular values as computed by SVD and as approximated by LU_CRTP (LU with column and row tournament pivoting) and LU_CTP (LU with column tournament pivoting and row partilal pivoting) for SHAW matrix.

cation lower bounds, in Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12, New York, NY, USA, June 2012, ACM, pp. 77–79.

6. ———, *Communication-optimal parallel algorithm for Strassen's matrix multiplication*, in Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12, New York, NY, USA, June 2012, ACM, pp. 193–204.
7. G. BALLARD, J. DEMMEL, O. HOLTZ, AND O. SCHWARTZ, *Graph expansion and communication costs of fast matrix multiplication*, in Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '11, ACM, June 2011, pp. 1–12.
8. G. BALLARD, J. DEMMEL, O. HOLTZ, AND O. SCHWARTZ, *Minimizing communication in linear algebra*, SIAM J. Matrix Anal. Appl., (2011).
9. G. BALLARD, J. DEMMEL, B. LIPSHITZ, O. SCHWARTZ, AND S. TOLEDO, *Communication efficient gaussian elimination with partial pivoting using a shape morphing data layout*, in Proceedings of 25th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2013.
10. D. W. BARRON AND H. P. F. SWINNERTON-DYER, *Solution of Simultaneous Linear Equations using a Magnetic-Tape Store*, Computer Journal, 3 (1960), pp. 28–33.
11. C. H. BISCHOF, *A parallel QR factorization algorithm with controlled local pivoting*, SIAM J. Sci. Stat. Comput., 12 (1991), pp. 36–57.
12. L. S. BLACKFORD, J. CHOI, A. CLEARY, E. D'AZEVEDO, J. W. DEMMEL, I. DHILLON J. J. DONGARRA, S. HAMMARLING, G. HENRY, A. PETITET, K. STANLEY, D. WALKER, AND R. C. WHA LEY, *ScaLAPACK Users' Guide*, SIAM, Philadelphia, PA, USA, May 1997.
13. P. A. BUSINGER AND G. H. GOLUB, *Linear least squares solutions by Householder transformations*, Numer. Math., 7 (1965), pp. 269–276.
14. L. E. CANNON, *A cellular computer to implement the Kalman filter algorithm*, PhD thesis, Montana State University, 1969.
15. E. CARSON, *Communication-Avoiding Krylov Subspace Methods in Theory and Practice*, PhD thesis, EECS Department, University of California, Berkeley, Aug 2015.
16. T. F. CHAN AND P. C. HANSEN, *Some applications of the rank revealing QR factorization*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 727–741.
17. A. T. CHRONOPOULOS AND W. GEAR, *S-step iterative methods for symmetric linear systems*, J. of Comput. Appl. Math., 25 (1989), pp. 153–168.
18. J. K. CULLUM AND R. A. WILLOUGHBY, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, vol. I: Theory, SIAM, Philadelphia, 2002.

19. J. W. DEMMEL, L. GRIGORI, M. GU, AND H. XIANG, *Communication avoiding rank revealing QR factorization with column pivoting*, Tech. Report UCB/EECS-2013-46, EECS Department, University of California, Berkeley, May 2013.
20. J. W. DEMMEL, L. GRIGORI, M. GU, AND H. XIANG, *Communication-avoiding rank-revealing QR decomposition*, SIAM Journal on Matrix Analysis and its Applications, 36 (2015), pp. 55–89.
21. J. W. DEMMEL, L. GRIGORI, M. HOEMMEN, AND J. LANGOU, *Communication-optimal parallel and sequential QR and LU factorizations*, Tech. Report UCB/EECS-2008-89, UC Berkeley, 2008. LAPACK Working Note 204.
22. J. W. DEMMEL, M. HOEMMEN, M. MOHIYUDDIN, AND K. YELICK, *Avoiding communication in sparse matrix computations*, in IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1–12.
23. J. W. DEMMEL, L. GRIGORI, M. GU, AND H. XIANG, *TSLU for nearly singular matrices*. In preparation, July 2017.
24. J. W. DEMMEL, L. GRIGORI, M. HOEMMEN, AND J. LANGOU, *Communication-optimal parallel and sequential QR and LU factorizations*, SIAM Journal on Scientific Computing, (2012), pp. 206–239. short version of technical report UCB/EECS-2008-89 from 2008.
25. S. DONFACK, L. GRIGORI, W. D. GROPP, AND V. KALE, *Hybrid static/dynamic scheduling for already optimized dense matrix factorization*, IEEE International Parallel and Distributed Processing Symposium IPDPS, (2012).
26. S. DONFACK, L. GRIGORI, AND A. KUMAR GUPTA, *Adapting communication-avoiding LU and QR factorizations to multicore architectures*, Proceedings of IPDPS, (2010).
27. Z. DRMAČ AND K. VESELIC, *New fast and accurate Jacobi SVD algorithm I*, SIAM J. Matrix Anal. Appl., 29 (2008), pp. 1322–1342.
28. ———, *New fast and accurate Jacobi SVD algorithm II*, SIAM J. Matrix Anal. Appl., 29 (2008), pp. 1343–1362.
29. C. ECKART AND G. YOUNG, *The approximation of one matrix by another of lower rank*, Psychometrika, 1 (1936), pp. 211–218.
30. E. ELMROTH, F. GUSTAVSON, I. JONSSON, AND B. KAGSTROM, *Recursive blocked algorithms and hybrid data structures for dense matrix library software*, SIAM Review, 46 (2004), pp. 3–45.
31. J. ERHEL, *A parallel GMRES version for general sparse matrices*, Electronic Transactions on Numerical Analysis, 3 (1995), pp. 160–176.
32. L. V. FOSTER AND X. LIU, *Comparison of rank revealing algorithms applied to matrices with well defined numerical ranks*. www.math.sjsu.edu/ foster/rank/rank_revealing_s.pdf, 2006.
33. M. FRIGO, C. E. LEISERSON, H. PROKOP, AND S. RAMACHANDRAN, *Cache-oblivious algorithms*, In FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science, (1999). IEEE Computer Society.
34. G. H. GOLUB, *Numerical methods for solving linear least squares problems*, Numer. Math., 7 (1965), pp. 206–216.
35. G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations (3rd Ed.)*, Johns Hopkins University Press, Baltimore, MD, USA, 1996.
36. N. L. ZAMARASHKIN, S. A. GOREINOV, AND E. E. TYRTYSHNIKOV, *A theory of pseudoskeleton approximations*, Linear Algebra and Its Applications, 261 (1997), pp. 1–21.
37. S. L. GRAHAM, M. SNIR, AND C. A. PATTERSON, eds., *Getting Up to Speed: The Future of Supercomputing*, National Academies Press, Washington, D.C., USA, 2005.
38. L. GRIGORI, S. CAYROLS, AND J. W. DEMMEL, *Low rank approximation of a sparse matrix based on LU factorization with column and row tournament pivoting*, Research Report RR-8910, inria, Mar. 2016. submitted to SIAM Journal on Scientific Computing, in revision.
39. L. GRIGORI, P.-Y. DAVID, J. DEMMEL, AND S. PEYRONNET, *Brief announcement: Lower bounds on communication for direct methods in sparse linear algebra*, Proceedings of ACM SPAA, (2010).
40. L. GRIGORI, J. DEMMEL, AND H. XIANG, *CALU: a communication optimal LU factorization algorithm*, SIAM Journal on Matrix Analysis and Applications, 32 (2011), pp. 1317–1350.

41. L. GRIGORI, J. W. DEMMEL, AND H. XIANG, *Communication avoiding Gaussian elimination*, Proceedings of the ACM/IEEE SC08 Conference, (2008).
42. L. GRIGORI, M. JACQUELIN, AND A. KHABOU, *Performance predictions of multilevel communication optimal LU and QR factorizations on hierarchical platforms*, in Proceedings of International Supercomputing Conference, LNCS, 2014.
43. L. GRIGORI, S. MOUFAWAD, AND F. NATAF, *Enlarged Krylov subspace conjugate gradient methods for reducing communication*, SIAM Journal on Matrix Analysis and Applications, (2016). preliminary version published as Inria TR 8597.
44. L. GRIGORI, R. STOMPOR, AND M. SZYDLARSKI, *A parallel two-level preconditioner for cosmic microwave background map-making*, Proceedings of the ACM/IEEE Supercomputing SC12 Conference, (2012).
45. M. GU AND S. C. EISENSTAT, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869.
46. F. GUSTAVSON, *Recursion Leads to Automatic Variable Blocking for Dense Linear-Algebra Algorithms*, IBM Journal of Research and Development, 41 (1997), pp. 737–755.
47. P. C. HANSEN, *Regularization tools version 4.1 for matlab 7.3*.
48. M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems.*, Journal of research of the National Bureau of Standards., 49 (1952), pp. 409–436.
49. N. HIGHAM, *Accuracy and Stability of Numerical Algorithms*, SIAM, second ed., 2002.
50. N. HIGHAM AND D. J. HIGHAM, *Large growth factors in Gaussian elimination with pivoting*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 155–164.
51. M. F. HOEMMEN, *Communication-avoiding Krylov subspace methods*, PhD thesis, EECS Department, University of California, Berkeley, Apr 2010.
52. J.-W. HONG AND H. T. KUNG, *I/O complexity: The Red-Blue Pebble Game*, in STOC '81: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing, New York, NY, USA, 1981, ACM, pp. 326–333.
53. D. IRONY, S. TOLEDO, AND A. TISKIN, *Communication lower bounds for distributed-memory matrix multiplication*, J. Parallel Distrib. Comput., 64 (2004), pp. 1017–1026.
54. W. M. KAHAN, *Numerical linear algebra*, Canad. Math. Bull., 9 (1966), pp. 757 – 801.
55. A. KHABOU, J. DEMMEL, L. GRIGORI, AND M. GU, *Communication avoiding LU factorization with panel rank revealing pivoting*, SIAM Journal on Matrix Analysis and Applications, 34 (2013), pp. 1401–1429. preliminary version published as INRIA TR 7867.
56. A. KHABOU, J. W. DEMMEL, L. GRIGORI, AND M. GU, *Communication avoiding LU factorization with panel rank revealing pivoting*, SIAM Journal on Matrix Analysis and Applications, 34 (2013), pp. 1401 – 1429.
57. M. W. MAHONEY, *Randomized algorithms for matrices and data*, Found. Trends Mach. Learn., 3 (2011), pp. 123–224.
58. L. MIRANIAN AND M. GU, *Strong rank revealing LU factorizations*, Linear Algebra and its Applications, (2003), pp. 1–16.
59. D. P. O'LEARY, *The block conjugate gradient algorithm and related methods.*, Linear Algebra and Its Applications, 29 (1980), pp. 293–322.
60. C.-T. PAN, *On the existence and computation of rank-revealing LU factorizations*, Linear Algebra and its Applications, 316 (2000), pp. 199–222.
61. G. PETERS AND J. H. WILKINSON, *The least squares problem and pseudo-inverses*, Computer Journal, 13 (1970), pp. 309–316.
62. Y. SAAD, *Numerical Methods for Large Eigenvalue Problems, 2nd ed.*, SIAM, Philadelphia, 2011.
63. Y. SAAD AND M. H. SCHULTZ, *GMRES: a generalized minimal residual algorithm for solving non-symmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.
64. R. SCHREIBER AND C. VAN LOAN, *A storage-efficient WY representation for products of Householder transformations*, SIAM Journal on Scientific and Statistical Computing, 10 (1989), pp. 53–57.
65. D. C. SORENSEN, *Analysis of pairwise pivoting in Gaussian elimination*, IEEE Transactions on Computers, 3 (1985), pp. 274 –278.

66. G.W. STEWART, *Four algorithms for the efficient computation of truncated QR approximations to a sparse matrix*, Numer. Math., 83 (1999), pp. 313–323.
67. X. SUN AND C. BISCHOF, *A basis-kernel representation of orthogonal matrices*, SIAM Journal on Matrix Analysis and Applications, 16 (1995), pp. 1184–1196.
68. S. TOLEDO, *Locality of reference in LU Decomposition with partial pivoting*, SIAM J. Matrix Anal. Appl., 18 (1997).
69. L. N. TREFETHEN AND R. S. SCHREIBER, *Average-case stability of Gaussian elimination*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 335–360.
70. H. A. VAN DER VORST, *Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 631–644.
71. J. H. WILKINSON, *Error analysis of direct methods of matrix inversion*, J. Assoc. Comput. Mach., 8 (1961), pp. 281–330.
72. W. WULF AND S. MCKEE, *Hitting the wall: Implications of the obvious*, ACM SIGArch Computer Architecture News, 23 (1995), pp. 20–24.