Title

A 3D Parallel Algorithm for QR Decomposition

Authors

Ballard, Grey

Demmel, James

Grigori, Laura

et al.

Publication Date

2018-07-11

DOI

10.1145/3210377.3210415

Peer reviewed

A 3D Parallel Algorithm for QR Decomposition

Grey Ballard
Wake Forest University
Winston Salem, NC, USA
ballard@wfu.edu

James Demmel
University of California
Berkeley, CA, USA
demmel@berkeley.edu

Laura Grigori
INRIA Paris-Rocquencourt
Paris, France
laura.grigori@inria.fr

Mathias Jacquelin
Lawrence Berkeley Natl. Lab.
Berkeley, CA, USA
mjacquelin@lbl.gov

Nicholas Knight
NYU–Courant
New York, NY, USA
nknight@nyu.edu

ABSTRACT

Interprocessor communication often dominates the runtime of large matrix computations. We present a parallel algorithm for computing QR decompositions whose bandwidth cost (communication volume) can be decreased at the cost of increasing its latency cost (number of messages). By varying a parameter to navigate the bandwidth/latency tradeoff, we can tune this algorithm for machines with different communication costs.

1. INTRODUCTION

A common task in numerical linear algebra, especially when solving least-squares and eigenvalue problems, is *QR-decomposing* a matrix into a unitary Q-factor times an upper trapezoidal R-factor. We present a QR decomposition algorithm, 3D-CAQR-EG, whose bandwidth and latency costs demonstrate a tradeoff.

We model the cost of a parallel algorithm in terms of the number of arithmetic operations, the number of words moved between processors, and the number of messages in which these words are moved. These three quantities, measured along critical paths in a parallel schedule, characterize the algorithm's arithmetic cost, bandwidth cost, and latency cost, respectively.

THEOREM 1. *An $m \times n$ matrix, $m \geq n$, can be QR-decomposed on P processors with these asymptotic costs:*

# operations	# words	# messages
mn^2/P	$n^2/(nP/m)^\delta$	$(nP/m)^\delta (\log P)^2$

(1)

where δ can be chosen from $[1/2, 2/3]$, assuming

$$P/(\log P)^4 = \Omega(m/n), \quad \text{and}$$
$$P \cdot (\log P)^2 = O\left(m^{\frac{\delta}{1+\delta}} \cdot n^{\frac{1-\delta}{1+\delta}}\right). \tag{2}$$

This arithmetic cost is optimal [DGHL12]. For the smallest $\delta = 1/2$, the latency cost is optimal, and for the largest $\delta = 2/3$, the bandwidth cost is optimal [BCD+14]. However, these bandwidth and latency lower bounds are not attained simultaneously: the bandwidth-latency product is $O(n^2(\log P)^2)$. We conjecture that this product must be $\Omega(n^2)$, meaning the tradeoff is inevitable.

Our main contribution is the presentation and analysis of 3D-CAQR-EG, which extends Elmroth-Gustavson's recursive algorithm [EG00] to the distributed-memory setting and uses communication-efficient subroutines. The inductive cases feature *3D matrix multiplication* (3DMM) [ABG+95], which incurs a smaller bandwidth cost than conventional (2D) approaches. The base cases feature a new variant of *communication-avoiding QR* (CAQR) [DGHL12]. CAQR incurs a smaller latency cost than conventional (Householder) QR. Our variant further improves the bandwidth cost. We chose the name '3D-CAQR-EG' to reflect this lineage.

For tall-and-skinny matrices whose aspect ratio is at least P, it's best to directly invoke 3D-CAQR-EG's base-case subroutine, 1D-CAQR-EG. 1D-CAQR-EG also demonstrates a bandwidth/latency tradeoff, albeit less drastic, which we can navigate to derive the following bounds.

THEOREM 2. *An $m \times n$ matrix can be QR-decomposed on $P \geq m/n \geq 1$ processors with these asymptotic costs:*

# operations	# words	# messages
mn^2/P	n^2	$(\log P)^2$

(3)

assuming $P \cdot (\log P)^2 = O(n^2)$.

The rest of this work is organized as follows. We start by summarizing relevant mathematical background on computing QR decompositions (Section 2). We then introduce our parallel machine model, formalizing how we quantify the costs of communication and computation (Section 3). Next we review the communication-efficient subroutines mentioned above, 3DMM (Section 4) and CAQR (Section 5). With this background in place, we present and analyze the new algorithms, 1D-CAQR-EG (Section 6) and 3D-CAQR-EG (Section 7), proving Theorems 1 and 2, in reverse order. We conclude by discussing limitations and extensions and comparing with related work (Section 8).

2. QR DECOMPOSITION

In Section 2, we summarize the relevant background concerning computing QR decompositions.

After formalizing the problem in Section 2.1, we present a recursive template algorithm, called REC-QR (Algorithm 1 in

ACM ISBN .
DOI:

Section 2.2), which includes many well-known algorithms as special cases. We then specialize REC-QR to utilize compact matrix representations (Section 2.3) and a simpler recursive splitting strategy (Section 2.4). The result of these specializations, called QR-EG (Algorithm 2), serves as a template for our two new algorithms, 1D-CAQR-EG and 3D-CAQR-EG.

2.1 QR Preliminaries

A *QR decomposition* of a matrix $\mathbf{A}$ is a matrix pair $(\mathbf{Q}, \mathbf{R})$ such that $\mathbf{A} = \mathbf{QR}$, the *Q-factor* $\mathbf{Q}$ is unitary, meaning $\mathbf{Q}^H\mathbf{Q} = \mathbf{QQ}^H = \mathbf{I}$, and the *R-factor* $\mathbf{R}$ is upper trapezoidal, meaning all entries below its main diagonal equal zero. To specialize for real-valued $\mathbf{A}$, simply substitute $(\cdot)^T$ for $(\cdot)^H$ and 'orthogonal' for 'unitary'.

We will always assume that $\mathbf{A}$ has at least as many rows as columns. (This implies that $\mathbf{R}$ has the same dimensions as $\mathbf{A}$ and is upper triangular.) When $\mathbf{A}$ has more columns than rows, we can obtain a QR decomposition by splitting $\mathbf{A} = \begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2\end{bmatrix}$ with square $\mathbf{A}_1$, decomposing $\mathbf{A}_1 = \mathbf{QR}_1$, and computing $\mathbf{R} = \begin{bmatrix}\mathbf{R}_1 & \mathbf{Q}^H\mathbf{A}_2\end{bmatrix}$.

2.2 Recursive QR Decomposition

We consider QR decomposition algorithms based on REC-QR (Algorithm 1), which split $\mathbf{A}$ vertically (Line 4), QR-decompose the left panel (Line 5), update the right panel (Line 6), QR-decompose the lower part of the (updated) right panel (Line 7), and then assemble a QR decomposition from the smaller ones (Lines 8 and 9).

Algorithm 1 $(\mathbf{Q}, \mathbf{R}) = \text{REC-QR}(\mathbf{A})$

1: **if** BASE-CONDITION **then**
2: $\quad(\mathbf{Q}, \mathbf{R}) = \text{BASE-QR}(\mathbf{A})$.
3: **else**
4: $\quad\text{SPLIT } \mathbf{A} = \begin{bmatrix}\mathbf{A}_{11} & \mathbf{A}_{12}\\ \mathbf{A}_{21} & \mathbf{A}_{22}\end{bmatrix}$ with $\mathbf{A}_{11}$ square.
5: $\quad(\mathbf{Q}_\text{L}, \mathbf{R}_\text{L}) = \text{REC-QR}\left(\begin{bmatrix}\mathbf{A}_{11}\\ \mathbf{A}_{21}\end{bmatrix}\right)$.
6: $\quad\begin{bmatrix}\mathbf{B}_{12}\\ \mathbf{B}_{22}\end{bmatrix} = \mathbf{Q}_\text{L}^H \cdot \begin{bmatrix}\mathbf{A}_{12}\\ \mathbf{A}_{22}\end{bmatrix}$.
7: $\quad(\mathbf{Q}_\text{R}, \mathbf{R}_\text{R}) = \text{REC-QR}(\mathbf{B}_{22})$.
8: $\quad\mathbf{Q} = \mathbf{Q}_\text{L} \cdot \begin{bmatrix}\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{Q}_\text{R}\end{bmatrix}$.
9: $\quad\mathbf{R} = \begin{bmatrix}\mathbf{R}_\text{L} & \begin{bmatrix}\mathbf{B}_{12}\\ \mathbf{R}_\text{R}\end{bmatrix}\end{bmatrix}$.
10: **end if**

We call REC-QR a 'template' because it leaves several details unspecified. To instantiate this template and obtain an algorithm, we must pick a base-case condition (BASE-CONDITION, Line 1), a base-case QR-decomposition subroutine (BASE-QR, Line 2), and a splitting strategy (SPLIT, Line 4). Additionally, we must specify how the operations are scheduled and how the data are distributed.

2.3 Compact Representations

In practice, a QR decomposition $(\mathbf{Q}, \mathbf{R})$ of an $m \times n$ matrix $(m \geq n)$ is typically not represented as a pair of explicit matrices. In Section 2.3 we specialize REC-QR to represent $\mathbf{Q}$ and $\mathbf{R}$ more compactly.

Since an R-factor is upper triangular, it is identifiable by just its superdiagonal entries. Subsequently, the symbol $\mathbf{R}$ may either denote (1) the actual R-factor, an $m \times n$ upper-triangular matrix; (2) the leading n rows of the R-factor, an $n \times n$ upper-triangular matrix; or (3) the upper triangle of the R-factor, a data structure of size $n(n+1)/2$. When presenting algorithms, we will prefer convention (2); to obtain an $n \times n$ upper-triangular $\mathbf{R}$ from REC-QR, we agree that BASE-QR (Line 2) returns such an $\mathbf{R}$, and we rewrite the R-factor assembly (Line 9) as

$$\mathbf{R} = \begin{bmatrix}\mathbf{R}_\text{L} & \mathbf{B}_{12}\\ \mathbf{0} & \mathbf{R}_\text{R}\end{bmatrix}.$$

Any unitary matrix $\mathbf{Q}$ can be written as $\mathbf{Q} = \mathbf{I} - \mathbf{VTV}^H$: the matrix pair $(\mathbf{V}, \mathbf{T})$ is called a *basis-kernel representation* [SB95] of $\mathbf{Q}$. If $\mathbf{Q}$ is the Q-factor of a QR decomposition of an $m \times n$ matrix $(m \geq n)$, then there exists such a representation where the *basis* $\mathbf{V}$ is $m \times n$ and the *kernel* $\mathbf{T}$ is $n \times n$.

Modifying REC-QR to use basis-kernel representations, Line 6 becomes

$$\begin{bmatrix}\mathbf{B}_{12}\\ \mathbf{B}_{22}\end{bmatrix} = \begin{bmatrix}\mathbf{A}_{12}\\ \mathbf{A}_{22}\end{bmatrix} - \mathbf{V}_\text{L}\mathbf{T}_\text{L}^H\mathbf{V}_\text{L}^H \begin{bmatrix}\mathbf{A}_{12}\\ \mathbf{A}_{22}\end{bmatrix}, \tag{4}$$

and Line 8 becomes

$$\begin{aligned}\mathbf{V} &= \begin{bmatrix}\mathbf{V}_\text{L} & \begin{bmatrix}\mathbf{0}\\ \mathbf{V}_\text{R}\end{bmatrix}\end{bmatrix}\\ \mathbf{T} &= \begin{bmatrix}\mathbf{T}_\text{L} & -\mathbf{T}_\text{L}\mathbf{V}_\text{L}^H\begin{bmatrix}\mathbf{0}\\ \mathbf{V}_\text{R}\end{bmatrix}\mathbf{T}_\text{R}\\ \mathbf{0} & \mathbf{T}_\text{R}\end{bmatrix},\end{aligned} \tag{5}$$

where $(\mathbf{V}_\text{L}, \mathbf{T}_\text{L})$ and $(\mathbf{V}_\text{R}, \mathbf{T}_\text{R})$ represent $\mathbf{Q}_\text{L}$ and $\mathbf{Q}_\text{R}$.

To simplify the presentation without affecting our asymptotic conclusions, we will not exploit the block-lower-trapezoidal and block-upper-triangular structures of the bases and kernels. With this understanding, it minimizes arithmetic to evaluate the quadruple matrix product in Equation (4) from right to left, and the quadruple product in Equation (5) from inside-out (two possibilities).

When m is close to n, a general basis-kernel representation may require more storage than the explicit $(m \times m)$ Q-factor. The QR decomposition algorithms in (Sca)LAPACK use a variant [Pug92] of *compact WY representation* [SVL89], which we call *Householder representation* in this work. In Householder representation, $\mathbf{V}$ is unit lower trapezoidal and $\mathbf{T}$ is upper triangular. These properties enable an in-place implementation, where $\mathbf{V}$'s strict lower trapezoid and $\mathbf{R}$'s upper triangle overwrite $\mathbf{A}$ and where $\mathbf{T}$ need not be stored, since in this case

$$\mathbf{T} = \left(\text{triu}(\mathbf{V}^H\mathbf{V}, -1) + \text{diag}(\text{diag}(\mathbf{V}^H\mathbf{V}))/2\right)^{-1},$$

using the MATLAB operations 'triu' and 'diag'.

Our algorithms will construct, store, and apply Q-factors in Householder representation. This choice is motivated by our practical goal of integration into the ScaLAPACK library [BCC+97]; from a theoretical standpoint, any basis-kernel representation (with $m \times n$ basis) would yield the same asymptotic costs.

2.4 Elmroth-Gustavson's Approach

The recursive framework of REC-QR is quite general. We will obtain our desired algorithmic costs by following an approach of Elmroth-Gustavson [EG00] (implemented in LAPACK's `_geqrt3`), in which we split $\mathbf{A}$ vertically (roughly) in half, until the number of columns drops below a given

threshold $b \geq 1$. The recursive calls define a binary tree whose $\lceil \log_2(n/b) \rceil$ levels are complete except possibly the last. (We always suppose $b \leq n$; when $b \geq n$, the tree has just one node.) We call this specialized template QR-EG (Algorithm 2); QR-EG utilizes the compact representations as explained in Section 2.3.

Algorithm 2 $(\mathbf{V}, \mathbf{T}, \mathbf{R}) = \text{QR-EG}(\mathbf{A}, b)$

1: **if** $n \leq b$ **then**
2: $(\mathbf{V}, \mathbf{T}, \mathbf{R}) = \text{BASE-QR}(\mathbf{A})$.
3: **else**
4: SPLIT $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$ so $\mathbf{A}_{11}$ is $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$.
5: $(\mathbf{V}_\text{L}, \mathbf{T}_\text{L}, \mathbf{R}_\text{L}) = \text{QR-EG}\left(\begin{bmatrix} \mathbf{A}_{11} \\ \mathbf{A}_{21} \end{bmatrix}, b \right)$.
6: $\mathbf{M}_1 = \mathbf{V}_\text{L}^H \cdot \begin{bmatrix} \mathbf{A}_{12} \\ \mathbf{A}_{22} \end{bmatrix}$.
7: $\mathbf{M}_2 = \mathbf{T}_\text{L}^H \cdot \mathbf{M}_1$.
8: $\begin{bmatrix} \mathbf{B}_{12} \\ \mathbf{B}_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{12} \\ \mathbf{A}_{22} \end{bmatrix} - \mathbf{V}_\text{L} \cdot \mathbf{M}_2$.
9: $(\mathbf{V}_\text{R}, \mathbf{T}_\text{R}, \mathbf{R}_\text{R}) = \text{QR-EG}(\mathbf{B}_{22}, b)$.
10: $\mathbf{V} = \begin{bmatrix} \mathbf{V}_\text{L} & \begin{bmatrix} \mathbf{0} \\ \mathbf{V}_\text{R} \end{bmatrix} \end{bmatrix}$.
11: $\mathbf{M}_3 = \mathbf{V}_\text{L}^H \cdot \begin{bmatrix} \mathbf{0} \\ \mathbf{V}_\text{R} \end{bmatrix}$.
12: $\mathbf{M}_4 = \mathbf{M}_3 \cdot \mathbf{T}_\text{R}$.
13: $\mathbf{T} = \begin{bmatrix} \mathbf{T}_\text{L} & -\mathbf{T}_\text{L} \cdot \mathbf{M}_4 \\ \mathbf{0} & \mathbf{T}_\text{R} \end{bmatrix}$.
14: $\mathbf{R} = \begin{bmatrix} \mathbf{R}_\text{L} & \mathbf{B}_{12} \\ \mathbf{0} & \mathbf{R}_\text{R} \end{bmatrix}$.
15: **end if**

We mention that [EG00] actually proposes a hybrid of the stated approach and an iterative approach, switching between the two for a constant-factor improvement in the arithmetic cost. (It still fits in the REC-QR framework.) While our algorithms can also benefit from this optimization, we omit further discussion in the present work since it does not affect our asymptotic conclusions.

3. COMPUTATION MODEL

We model a parallel machine as a set of P interconnected processors, each with unbounded local memory. Processors operate on local data and communicate with other processors by sending and receiving messages. A processor can perform at most one task (operation/send/receive) at a time. A (data) word means a complex number; operations are the usual field actions, plus complex conjugation and real square roots. Messages are point-to-point and asynchronous. Each operation takes time γ, while sending or receiving a message of w words takes time $\alpha + w\beta$, α being the latency and β the inverse of the bandwidth.

We model an execution as a DAG whose vertices are tasks and whose edges define P paths, one for each processor's task sequence, plus an inter-path edge for each send/receive pair. Weighting vertices by their tasks' durations, we define runtime as the maximum weight of any path. Therefore, if every path includes at most F operations and at most S messages, containing at most W words in total, the runtime is bounded,

$$\text{runtime} \leq \gamma \cdot F + \beta \cdot W + \alpha \cdot S.$$

When multiple processors send/receive messages simultaneously, it can be more efficient to split the messages into more messages of smaller sizes, to coalesce them into fewer, larger messages, or to route them through intermediate processors. These lower-level implementation details are often irrelevant to our asymptotic analyses, motivating us to express our algorithms' communication patterns abstractly, in terms of collectives.

In the rest of Section 3 we define the eight different collectives appearing in this work, collecting in Table 1 their costs.

A general scenario, called an ALL-TO-ALL, is when every processor p initially owns a block of data, containing B_{pq} words, destined for every processor q, including itself. All other collectives we use can be interpreted as special cases of an ALL-TO-ALL. Four of these distinguish a 'root' processor r: SCATTER, where only r's outgoing blocks are nonempty; GATHER, where only r's incoming blocks are nonempty; BROADCAST, a SCATTER where r's outgoing blocks are identical; and REDUCE, a GATHER where r's incoming blocks have same size and are added entrywise. Four others, including ALL-TO-ALL, can be constructed from the first four: ALL-GATHER, P GATHERs with same outgoing blocks but different roots; ALL-REDUCE, P REDUCEs with same outgoing blocks but different roots; ALL-TO-ALL, P GATHERs with different outgoing blocks and different roots; and REDUCE-SCATTER, P REDUCEs with different outgoing blocks and different roots. (Since the three 'REDUCE' collectives perform arithmetic, they are technically not ALL-TO-ALLs.)

LEMMA 1. *There exist algorithms for the eight collectives satisfying the upper bounds in Table 1.*

PROOF. For all but ALL-TO-ALL we use a *binomial-tree* and possibly a *bidirectional-exchange algorithm*: see, e.g., [TRG05, CHPvdG07]. In particular, for (ALL-)GATHER and (REDUCE-)SCATTER we use binary tree algorithms, and for BROADCAST and (ALL-)REDUCE we use whichever of the two minimizes all three costs, asymptotically. For ALL-TO-ALL we use the *(radix-2) index algorithm* [BHK+97], possibly performed twice using the load-balancing approach of [HBJ96]. (For a more detailed proof, see Section A.) □

Note that when the number of processors is not too large w.r.t. the block size, the bidirectional-exchange algorithm for BROADCAST, built from SCATTER+ALL-GATHER, is asymptotically cheaper than the corresponding binary tree algorithm in terms of bandwidth. Similarly, the bidirectional-exchange algorithm for (ALL-)REDUCE), built from REDUCE-SCATTER+(ALL-)GATHER, improves both arithmetic and bandwidth.

4. 3D MATRIX MULTIPLICATION

The key to reducing 3D-CAQR-EG's bandwidth cost below that of previous approaches is *3D matrix multiplication* (3DMM) [ABG+95]. Here, *3D* refers to the parallelization of the operations and distribution of data over a three-dimensional (logical) processor grid. 1D-CAQR-EG will also exploit two special cases, 1DMM, performed on a one-dimensional grid, and MM, performed locally on one processor.

For concreteness, consider multiplying an $I \times K$ matrix $\mathbf{A}$ with a $K \times J$ matrix $\mathbf{B}$ to obtain an $I \times J$ matrix $\mathbf{C}$, via the

	# operations	# words	# messages
SCATTER	0	$(P-1)B$	$\log P$
GATHER	0	$(P-1)B$	$\log P$
BROADCAST	0	$\min(B\log P,\ B+P)$	$\log P$
REDUCE	$\min(B\log P,\ B+P)$	$\min(B\log P,\ B+P)$	$\log P$
ALL-GATHER	0	$(P-1)B$	$\log P$
ALL-REDUCE	$\min(B\log P,\ B+P)$	$\min(B\log P,\ B+P)$	$\log P$
ALL-TO-ALL	0	$\min(BP\log P,\ (B_*+P^2)\log P)$	$\log P$
REDUCE-SCATTER	$(P-1)B$	$(P-1)B$	$\log P$

Table 1: Asymptotic costs of the collectives defined in Section 3. P is the number of processors involved, $B=\max_{p,q} B_{pq}$ is the largest block-size, and $B_*=\max\left(\max_q \sum_p B_{pq},\ \max_p \sum_q B_{pq}\right)$ is the maximum number of words any processor holds before/after. Any P can be replaced by $|\{p,q : B_{pq}>0\}|$ for a possibly smaller (valid) bound.

usual (entrywise) formula:

$$\text{for all } (i,j)\in[I]\times[J],\quad \mathbf{C}_{ij}=\sum_{k\in[K]}\mathbf{A}_{ik}\mathbf{B}_{kj}. \tag{6}$$

We identify each of the IJK (scalar) multiplications with a point (i,j,k) in three-dimensional Euclidean space, so the set of multiplications defines a discrete $I\times J\times K$ brick. We point the reader to [BDKS16, SVdGP16] for further discussions of this geometric terminology.

LEMMA 2. *Suppose input matrices* $\mathbf{A}$ *and* $\mathbf{B}$ *are initially owned by processor* p*, and output matrix* $\mathbf{C}$ *is to be finally owned also by processor* p*.* $\mathbf{C}=\mathbf{A}\cdot\mathbf{B}$ *can be computed with runtime*

$$\gamma\cdot O\left(IJK\right). \tag{7}$$

PROOF. Directly evaluating the sums-of-products in Equation (6) on processor p involves IJK multiplications and $IJ(K-1)$ additions; no communication is necessary. □

LEMMA 3. *Suppose* I*,* J*,* K*, and* P *satisfy*

$$P=O\left(\frac{IJK}{\max(I,J,K)}\right)\quad\text{and}\quad P=O\left(\max(I,J,K)\right).$$

If $K=\max(I,J,K)$*, suppose that matrices* $\mathbf{A}^T$ *and* $\mathbf{B}$ *are initially distributed in matching row-wise layouts where each processor owns* $O(K/P)$ *rows, and that matrix* $\mathbf{C}$ *is to be finally owned by a single processor* r*. Alternatively, if* $I=\max(I,J,K)$*, suppose that matrices* $\mathbf{A}$ *and* $\mathbf{C}$ *are initially/finally distributed in matching row-wise layouts where each processor owns* $O(I/P)$ *rows, and that matrix* $\mathbf{B}$ *is initially owned by a single processor* r*.* $\mathbf{C}=\mathbf{A}\cdot\mathbf{B}$ *can be computed with runtime*

$$\gamma\cdot O\left(\frac{IJK}{P}\right)+\beta\cdot O\left(\frac{IJK}{\max(I,J,K)}\right)+\alpha\cdot O\left(\log P\right), \tag{8}$$

PROOF. In the first case, each processor performs a local MM and then all processors REDUCE to processor r. In the second case, processor r BROADCASTs $\mathbf{B}$ to all processors and then each processor performs a local MM. The hypotheses guarantee that P is not too large for these collectives can leverage the bidirectional-exchange algorithms. (For a more detailed proof, see Section B.) □

The bound of Equation (8) also holds in a third case, when $J=\max(I,J,K)$ and the distributions are symmetric to the second case, but we will not need this result.

LEMMA 4. *Suppose* I*,* J*,* K*, and* P *satisfy*

$$c\cdot\frac{IJK}{\min(I,J,K)^3}\le P\le IJK.$$

There exists a data distribution of $\mathbf{A}$*,* $\mathbf{B}$*, and* $\mathbf{C}$ *such that each processor initially owns at most* $O((I+J)K/P)$ *entries of* $\mathbf{A}$ *and* $\mathbf{B}$*, and finally at most* $O(IJ/P)$ *entries of* $\mathbf{C}$*, where* $\mathbf{C}=\mathbf{A}\cdot\mathbf{B}$ *can be computed with runtime*

$$\gamma\cdot O\left(\frac{IJK}{P}\right)+\beta\cdot O\left(\left(\frac{IJK}{P}\right)^{2/3}\right)+\alpha\cdot O\left(\log P\right). \tag{9}$$

PROOF. Pick $Q=\lfloor I/\rho\rfloor$, $R=\lfloor J/\rho\rfloor$, and $S=\lfloor K/\rho\rfloor$, where $\rho=(IJK/P)^{1/3}$. Under the hypotheses, Q,R,S are positive integers with $QRS\le P$, thus define a valid $Q\times R\times S$ processor grid. Moreover, $QRS=\Omega(P)$. Pick partitions $\{\mathcal{I}_q\}_q$, $\{\mathcal{J}_r\}_r$, and $\{\mathcal{K}_s\}_s$ of $[I]$, $[J]$, and $[K]$ which are *balanced*, meaning their parts differ in size by at most one. Pick a partition $\{\mathcal{A}_{q,r,s}\}_{q,r,s}$ of $[I]\times[K]$ to be a union of balanced R-way partitions of the sets $\mathcal{I}_q\times\mathcal{K}_s$ $((q,s)\in[Q]\times[S])$, and similarly for partitions $\{\mathcal{B}_{q,r,s}\}_{q,r,s}$ and $\{\mathcal{C}_{q,r,s}\}_{q,r,s}$ of $[K]\times[J]$ and $[I]\times[J]$. Distribute entries $\mathcal{A}_{q,r,s}$ of $\mathbf{A}$ to each grid processor (q,r,s), and similarly for $\mathbf{B}$ and $\mathbf{C}$: this distribution satisfies the balance constraint in the theorem statement. The algorithm proceeds with ALL-GATHERs of blocks of $\mathbf{A}$ and $\mathbf{B}$ along processor grid fibers in the Q- and R-directions. then local MMs, then finally REDUCE-SCATTERs of blocks of $\mathbf{C}$ along processor grid fibers in the S-direction. (For a more detailed proof, see Section B.) □

We denote by MM, 1DMM, or 3DMM an algorithm that satisfies Lemma 2, Lemma 3, or Lemma 4, resp.

5. COMMUNICATION-AVOIDING QR

Our new algorithms 1D-CAQR-EG and 3D-CAQR-EG are closely related to *communication-avoiding QR (*CAQR*)* and *tall-skinny QR (*TSQR*)* [DGHL12]: we explore this relationship in Section 8. For now we remark that 3D-CAQR-EG specializes QR-EG to use 1D-CAQR-EG as a base-case, and that 1D-CAQR-EG specializes QR-EG to use TSQR (the variant in [BDG+15]) as a base-case.

On input, the $m\times n$ matrix $\mathbf{A}$ is partitioned across the P processors so that each processor p owns $m_p\ge n$ rows, not necessarily contiguous. Thus we require $\mathbf{A}$ be sufficiently tall and skinny: $m/n\ge P$. A single processor r, which owns $\mathbf{A}$'s n leading rows, is designated as the *root processor*.

On output, the Q-factor is stored in Householder representation ($\mathbf{V}$, $\mathbf{T}$), where $\mathbf{V}$ has the same distribution as $\mathbf{A}$.

Both $\mathbf{T}$ and the R-factor are returned only on the root processor.

LEMMA 5. TSQR*'s runtime is*

$$\gamma \cdot O\left(\max_p m_p n^2 + n^3 \log P\right) + \beta \cdot O\left(n^2 \log P\right) + \alpha \cdot O\left(\log P\right)$$

PROOF. It is crucial to use the TSQR variant in [BDG+15]. (See also Section C for a proof.) □

Recall from Section 3 that when the block-size is sufficiently large, REDUCE and BROADCAST can be performed more efficiently, by REDUCE-SCATTER+GATHER and SCATTER+ALL-GATHER, resp. Unfortunately, TSQR's REDUCE- and BROADCAST-like collectives preclude these optimizations. Next in Section 6, we will show how similar savings are achievable.

6. 1D-CAQR-EG

We now present a new algorithm, 1D-CAQR-EG, an instantiation of the template QR-EG (Algorithm 2). 1D-CAQR-EG effectively reduces TSQR's bandwidth cost by a logarithmic factor, at the expense of increasing its latency cost by a comparable factor.

The input/output data distributions are the same as for TSQR, so we continue notation from Section 5. We specify 1D-CAQR-EG by stepping line-by-line through QR-EG — base case in Section 6.1 and inductive case in Section 6.2 — then prove Theorem 2 in Section 6.3.

6.1 Base Case

1D-CAQR-EG's base-case QR decomposition subroutine (Line 2) is TSQR (Section 5), using the same root processor. Note that $\mathbf{A}$'s distribution satisfies TSQR's requirements, and $\mathbf{V}$, $\mathbf{T}$, and $\mathbf{R}$ are returned distributed as required by 1D-CAQR-EG.

6.2 Inductive Case

Let us walk through the inductive case line-by-line. All algorithmic costs are incurred in the two recursive calls (Lines 5 and 9) and the six matrix multiplications (Lines 6 to 8 and 11 to 13).

(Line 4): the splitting involves no computation nor communication.

(Line 5): the left recursive call is valid since $\left[\begin{smallmatrix}\mathbf{A}_{11}\\ \mathbf{A}_{21}\end{smallmatrix}\right]$ still satisfies the data distribution requirements (only n decreases).

(Line 6): this is a 3DMM with matrix dimensions $I = \lfloor n/2 \rfloor$, $J = \lceil n/2 \rceil$, and $K = m$. We choose a (1D) processor grid with $Q = R = 1$ and $S = P$, thus $T = 0$ and the partitions $\{\mathcal{I}_q\}_q = \{[I]\}$ and $\{\mathcal{J}_r\}_r = \{[J]\}$ are trivial. We pick the partition $\{\mathcal{K}_s\}_s$ to match the distribution of $\mathbf{A}$'s rows, and pick $\{\mathcal{A}_{q,r,s}\}_{q,r,s} = \{[I] \times \mathcal{K}_s\}_s$ and $\{\mathcal{B}_{q,r,s}\}_{q,r,s} = \{\mathcal{K}_s \times [J]\}_s$. Additionally, we set $\mathcal{C}_{q,r,s} = \emptyset$ for all (q, r, s) but the root processor.

(Line 7): this is an MM on the root processor with matrix dimensions $I = K = \lfloor n/2 \rfloor$ and $J = \lceil n/2 \rceil$.

(Line 8): this is a 3DMM with matrix dimensions $I = m$, $J = \lceil n/2 \rceil$, and $K = \lfloor n/2 \rfloor$, followed by a matrix subtraction. We choose a (1D) processor grid with $Q = P$ and $R = S = 1$, thus $T = 0$ and the partitions $\{\mathcal{J}_r\}_r = \{[J]\}$ and $\{\mathcal{K}_s\}_s = \{[K]\}$ are trivial. We pick the partition $\{\mathcal{I}_r\}_r$ to match the distribution of $\mathbf{A}$'s rows, and pick $\{\mathcal{A}_{q,r,s}\}_{q,r,s} = \{\mathcal{I}_r \times [K]\}_r$ and $\{\mathcal{C}_{q,r,s}\}_{q,r,s} = \{\mathcal{I}_r \times [J]\}_r$. Additionally, we set $\mathcal{B}_{q,r,s} = \emptyset$ for all (q, r, s) but the root processor. The row-wise distribution $\{\mathcal{C}_{q,r,s}\}_{q,r,s}$ enables the subsequent matrix subtraction to be performed without further communication.

(Line 9): the second recursive call is valid since $\mathbf{B}_{22}$ still satisfies the data distribution requirements: the number of rows owned by the root processor decreases by the same amount that n does, while all other processors keep the same number.

(Line 10): each processor assembles its local rows of $\mathbf{V}$: no computation nor communication is required.

(Line 11): this is a 3DMM with matrix dimensions $I = \lfloor n/2 \rfloor$, $J = \lceil n/2 \rceil$, and $K = m - \lfloor n/2 \rfloor$; we choose a processor grid and partitions as in Line 6.

(Line 12): this is an MM on the root processor with the same dimensions as Line 7.

(Line 13): this is an MM on the root processor with the same dimensions as Lines 7 and 12.

(Line 14): each processor assembles its local rows of $\mathbf{R}$: no computation nor communication is required.

Verify that $\mathbf{V}$, $\mathbf{T}$, and $\mathbf{R}$ are distributed as desired.

6.3 Concluding the Analysis

1D-CAQR-EG is valid for any $P, m, n, b \geq 1$ such that $P \leq m/n$, and there is no loss of generality to suppose $b \leq n$. When $b = n$, 1D-CAQR-EG reduces to TSQR. As we will see, picking $b < n$ allows us to reduce 1D-CAQR-EG's arithmetic and bandwidth costs — while increasing its latency cost — to appear as if we had used bidirectional exchange REDUCE and BROADCAST algorithms (Section 3) within TSQR, despite the fact that these algorithms are inapplicable, as we lamented at the end of Section 5. We will navigate the tradeoff with a nonnegative parameter ϵ, taking

$$b = \Theta\left(n/(\log P)^{\epsilon}\right). \tag{10}$$

We will show that taking $\epsilon = 1$ yields Theorem 2.

LEMMA 6. *If* $P = O(b^2)$, 1D-CAQR-EG *has runtime*

$$\gamma \cdot \left(\frac{mn^2}{P} + nb^2 \log P\right) + \beta \cdot O\left(n^2 + nb \log P\right) + \alpha \cdot O\left(\frac{n}{b} \log P\right). \tag{11}$$

PROOF. Here we give an expanded proof of Lemma 6.

Let us derive an upper bound $T(m, n)$ on the runtime of an 1D-CAQR-EG invocation. (The unchanging parameters P, b are implicit.) We will now assume a balanced data distribution, meaning the numbers of rows any two processors owns differ by at most one.

When $P = 1$, the runtime of 1D-CAQR-EG for any b is just $\gamma \cdot O\left(mn^2\right)$, which satisfies the conclusion, so we may assume $P > 1$ hereafter.

In the base case ($n \leq b$), the algorithmic cost is the 3D-CAQR-EG call, so by Lemma 5 we conclude that

$$T(m, n) = \gamma \cdot O\left(\frac{mn^2}{P} + n^3 \log P\right) + \beta \cdot O\left(n^2 \log P\right) + \alpha \cdot O\left(\log P\right).$$

In the inductive case, the algorithmic cost is due to the two recursive calls, the three 1DMM calls (Lines 6, 8 and 11, performed on 1D processor grids, and the three (local) MM calls (Lines 7, 12 and 13), performed by the root.

The local MMs have runtime $\gamma \cdot O\left(n^3\right)$.

To apply Lemma 3 to the 3DMM calls, let us now suppose that $P = O\left(n^2\right)$ and $P = O(m)$, Actually, only the first assumption is new: we already know that $P \leq m/n$ for the initial m, n, and when $P > 1$ we see that in any recursive call the current m is within a factor of two of the initial m, hence $P = O(m)$ in any recursive call. Confirming that the data distributions chosen in Section 6.2 match those in the proof of Lemma 3, the 1DMMs' runtime is

$$\gamma \cdot O\left(\frac{mn^2}{P}\right) + \beta \cdot O\left(n^2\right) + \alpha \cdot O\left(\log P\right).$$

Overall, we have found that

$$T(m,n) = T(m, \lfloor n/2) + T(m - \lfloor n/2 \rfloor, \lceil n/2 \rceil) + \gamma \cdot O\left(\frac{mn^2}{P}\right) + \beta \cdot O\left(n^2\right) + \alpha \cdot O\left(\log P\right).$$

By induction we may take $T(m,n)$ to be nondecreasing in both m and n. The former property justifies replacing $m - \lfloor n/2 \rfloor$ by m in the second recursive call. Hence, we will take m to be its initial value in the analysis of every recursive call.

Supposing $n = b2^L$ for a nonnegative integer L, $T(m,n)$ is bounded by Equation (11).

Now observe that $n = b2^{L+1}$ reproduces the asymptotic bound Equation (11), and since $T(m,n)$ is nondecreasing in n, this bound also holds when $b2^L < n < b2^{L+1}$.

Requiring $P = O(b^2)$ suffices to ensure that $P = O(n^2)$ at every inductive case. □

Proof of Theorem 2. Substituting Equation (10) into Equation (11),

$$\gamma \cdot \left(\frac{mn^2}{P}\left(1 + \frac{nP}{m}(\log P)^{1-2\epsilon}\right)\right) + \beta \cdot O\left(n^2\left(1 + (\log P)^{1-\epsilon}\right)\right) + \alpha \cdot O\left((\log P)^{1+\epsilon}\right),$$

thence the hypothesis is $P(\log P)^{2\epsilon} = O\left(n^2\right)$. We conclude by taking $\epsilon = 1$. □

This argument extends to any $\epsilon \geq 0$, assuming $P(\log P)^{2\epsilon} = O(n^2)$, but the asymptotic tradeoff vanishes when $\epsilon > 1$. For $1/2 \leq \epsilon \leq 1$ the tradeoff is only between bandwidth and latency. A sensible interpretation of the case $\epsilon < 0$ is $b = n$, meaning TSQR is invoked immediately. In this case, the costs are given directly by Lemma 5.

7. 3D-CAQR-EG

We now present our second new algorithm, 3D-CAQR-EG, another instantiation of the template QR-EG (Algorithm 2).

On input, the $m \times n$ matrix $\mathbf{A}$ ($m \geq n$) is partitioned across the P processors row-cyclically: thus, each processor owns at most $\lceil m/P \rceil$ rows.

On output, the Q-factor is stored in Householder representation ($\mathbf{V}$, $\mathbf{T}$), where $\mathbf{V}$ has the same distribution as $\mathbf{A}$. Both $\mathbf{T}$ and the R-factor have the same distribution, matching the top $n \times n$ submatrix of $\mathbf{A}$.

After walking through 3D-CAQR-EG in a similar fashion as we did for 1D-CAQR-EG in Section 6, we collect the results to prove Theorem 1.

In the following, $T_{\text{3D-CAQR-EG}}$ denotes an upper bound on the runtime of 3D-CAQR-EG, a function of P, m, n, b.

7.1 Base Case

Recall that b denotes the recursive threshold for 3D-CAQR-EG. 3D-CAQR-EG's base-case QR decomposition subroutine (Line 2) is 1D-CAQR-EG, with a fixed recursive threshold b^*.

To satisfy 1D-CAQR-EG's data distribution requirements, we convert $\mathbf{A}$ from row-cyclic to block-row layout, distributed over

$$P^* = \min\left(P, \lfloor m/n \rfloor\right)$$

processors, ensuring each owns at least n rows and one owns the top n rows (and perhaps others).

Initially, $P' = \min(m, P)$ processors own rows of $\mathbf{A}$. (Clearly $P' \leq P$ with equality just in case $P \leq m$; note further that $P^* \leq P'$ with equality just in case $P \leq m/n$.) Number these processors from 0 to $P'-1$ according to the cyclic layout of $\mathbf{A}$, so that processor 0 owns the top row of A. Deal these processors among P^* groups, so processor 0 goes into group 0, processor 1 goes into group 1, and so on. Represent each group by its lowest-numbered processor and within each group, GATHER $\mathbf{A}$'s rows to the representative. Since each group contains at most $\lceil P'/P^* \rceil$ processors and each processor initially owns at most $\lceil m/P' \rceil$ rows of $\mathbf{A}$, the largest block-size in any GATHER is at most $\lceil m/P' \rceil n$.

Each of the P^* representatives (including processor 0) now owns at least $\lfloor m/P^* \rfloor \geq n$ rows of $\mathbf{A}$, satisfying the first part of 1D-CAQR-EG's data distribution requirements: it remains to ensure processor 0 owns the top n rows of $\mathbf{A}$. These rows are currently owned by the first $P'' = \min(P^*, n)$ representatives. (Clearly $P'' \leq P^*$ with equality just in case $P \leq n$.) We next perform a GATHER over the representatives of groups 0 through $P''-1$, taking processor 0 to be the root so that afterwards it owns the top n rows of $\mathbf{A}$ (and perhaps others). We also perform a SCATTER with the opposite communication pattern so that the overall number of rows per representative is unchanged. The largest block-size in both the GATHER and the SCATTER is at most $\lceil n/P'' \rceil n$.

We can now invoke 1D-CAQR-EG, with parameters P^*, b^*. After it returns, we redistribute $\mathbf{V}$, $\mathbf{T}$, and $\mathbf{R}$ by reversing the preceding GATHERs/SCATTERs, so that $\mathbf{V}$ is (resp., $\mathbf{T}$ and $\mathbf{R}$ are) distributed over all P processors like $\mathbf{A}$ was (resp., $\mathbf{A}$'s first n rows were) initially.

7.2 Inductive Case

Let us walk through the inductive case line-by-line, as we did for 1D-CAQR-EG. All algorithmic costs are incurred in the two recursive calls (Lines 5 and 9) and the six matrix multiplications (Lines 6 to 8 and 11 to 13).

(Line 4): the splitting involves no computation nor communication.

(Line 5): the left recursive call is valid since $\left[\begin{smallmatrix}\mathbf{A}_{11}\\ \mathbf{A}_{21}\end{smallmatrix}\right]$ still satisfies the data distribution requirements (only n decreases).

(Line 6): this is a 3DMM with matrix dimensions $I = \lfloor n/2 \rfloor$, $J = \lceil n/2 \rceil$, and $K = m$. We do not yet specify the processor grid, but we do suppose that 3D-CAQR-EG uses a balanced parallelization and data distribution as in the proof of Lemma 4: this is possible for any processor grid.

To match this data distribution, we perform an ALL-TO-ALL before and after the 3DMM invocation, each time using the two-phase approach [BHK+97]. The first ALL-TO-ALL redistributes the input matrices from column- and row-cyclic to 3DMM layout (the left factor is row-cyclic, transposed); the maximum number of input matrix entries any processor owns before or after this collective is at most

$$\max\left(I\left\lceil\frac{K}{P}\right\rceil+\left\lceil\frac{K}{P}\right\rceil J,\ \left\lceil\frac{\left\lceil\frac{I}{Q}\right\rceil\left\lceil\frac{K}{S}\right\rceil}{R}\right\rceil+\left\lceil\frac{\left\lceil\frac{J}{R}\right\rceil\left\lceil\frac{K}{S}\right\rceil}{Q}\right\rceil\right),$$

where the processor grid is $Q \times R \times S$. The second ALL-TO-ALL converts the output matrix from 3DMM layout to row-cyclic layout; the maximum number of output matrix entries any processor owns before or after this collective is at most

$$\max\left(\left\lceil\frac{\left\lceil\frac{I}{Q}\right\rceil\left\lceil\frac{J}{R}\right\rceil}{S}\right\rceil,\ \left\lceil\frac{I}{P}\right\rceil J\right).$$

(Line 7): this is a 3DMM with matrix dimensions $I = K = \lfloor n/2 \rfloor$ and $J = \lceil n/2 \rceil$. We pick a 3D processor grid and partitions satisfying the same constraints as for Line 6, and perform similar ALL-TO-ALLs before and after, so we can reuse the preceding analysis, substituting I, J, and K.

(Line 8): this is a 3DMM with matrix dimensions $I = m$, $J = \lceil n/2 \rceil$, and $K = \lfloor n/2 \rfloor$, followed by a matrix subtraction. We proceed similarly to Lines 6 and 7, except that the left factor is initially in row-cyclic layout, so the first summand in the first term of the maximum becomes $\lceil I/P \rceil K$ (vs. $I\lceil K/P \rceil$).

(Line 9): the second recursive call is valid since $\mathbf{B}_{22}$ still satisfies the data distribution requirements: in particular, unlike 1D-CAQR-EG there is no requirement that a fixed processor owns the first n rows or that every processor owns at least n rows.

(Line 10): each processor assembles its local rows of $\mathbf{V}$: no computation nor communication is required.

(Line 11): this is a 3DMM with matrix dimensions $I = \lfloor n/2 \rfloor$, $J = \lceil n/2 \rceil$, and $K = m - \lfloor n/2 \rfloor$; we choose a processor grid, partitions, and ALL-TO-ALLs as in Line 6.

(Line 12): this is a 3DMM with the same dimensions, processor grid, partitions, and ALL-TO-ALLs as Line 7.

(Line 13): this is a 3DMM with the same dimensions, processor grid, partitions, and ALL-TO-ALLs as Lines 7 and 12.

(Line 14): each processor assembles its local rows of $\mathbf{R}$: no computation nor communication is required.

Verify that $\mathbf{V}$, $\mathbf{T}$, and $\mathbf{R}$ are distributed as desired.

7.3 Concluding the Analysis

3D-CAQR-EG is valid for any $P, m, n, b, b^* \geq 1$, and there is no loss of generality to suppose $b^* \leq b \leq n$. Taking $b = n$ simplifies 3D-CAQR-EG to 1D-CAQR-EG with parameters P^*, b^* and additional data redistributions. As in the case of 1D-CAQR-EG, picking $b < n$ allows us to reduce 3D-CAQR-EG's arithmetic and bandwidth costs, while increasing its latency cost.

We will navigate this tradeoff with two nonnegative parameters δ, ϵ, taking

$$b = \Theta\left(n/\left(nP/m\right)^{\delta}\right), \qquad b^* = \Theta\left(b/(\log P)^{\epsilon}\right). \tag{12}$$

We prove Theorem 1 with $\delta \in [1/2, 2/3]$ and $\epsilon = 1$.

LEMMA 7. *If $P = O(b^2)$ and $P^* = O(b^{*2})$, 3D-CAQR-EG has runtime*

$$\begin{aligned} T(m,n) = \gamma \cdot \left(\frac{mn^2}{P} + nb^{*2}\log P\right) \\ + \beta \cdot O\left(\frac{mn}{P} + nb + nb^*\log P + \left(\frac{mn^2}{P}\right)^{2/3}\right. \\ \left. + \left(\left(\frac{mn}{P}+n\right)\log\frac{n}{b} + \frac{nP^2}{b}\right)\log P\right) \\ + \alpha \cdot O\left(\frac{n}{b^*}\log P\right). \end{aligned} \tag{13}$$

PROOF. Here we give an expanded proof of Lemma 7.

Let us derive an upper bound $T(m,n)$ on the runtime of a 3D-CAQR-EG invocation. (The unchanging parameters P, b, b^* are implicit.)

When $P = 1$, the runtime of 3D-CAQR-EG is just $\gamma \cdot O\left(mn^2\right)$, which satisfies the conclusion, so we may assume $P > 1$ hereafter.

In a base case ($n \leq b$), the algorithmic costs are due to the 1D-CAQR-EG invocation and the four communication phases.

The first communication phase, involving P^* independent GATHERs, has runtime bounded by

$$\begin{aligned} &\beta \cdot O\left((\lceil P'/P^* \rceil - 1)\lceil m/P' \rceil n\right) + \alpha \cdot O\left(\log\lceil P'/P^* \rceil\right) \\ &\qquad = \beta \cdot O\left(mn/P + n^2\right) + \alpha \cdot O\left(\log P\right), \end{aligned}$$

and the same bound applies for the last phase (matching SCATTERs). (Recall that $P^* = \min(P, \lfloor m/n \rfloor)$ and $P' = \min(P, m)$.) The fact that no communication happens when $m \geq nP$ (and thus $P^* = P$) is evident in the first bound but not the second.

The second communication phase, a simultaneous GATHER/SCATTER, has runtime bounded by

$$\begin{aligned} &\beta \cdot O(\left((P'' - 1)\lceil n/P'' \rceil n\right) + \alpha \cdot O\left(\log P''\right) \\ &\qquad = \beta \cdot O\left(n^2\right) + \alpha \cdot O\left(\log P\right), \end{aligned}$$

and the same bound applies for the third phase (matching SCATTER/GATHER). (Recall that $P'' = \min(P^*, n)$.)

A runtime bound for the 1D-CAQR-EG invocation is given by Theorem 2, supposing now that $P^* = O(b^{*2})$. Actually we use the more refined bound of Equation (11), substituting P^*, b^* for P, b.

Altogether, in a base case,

$$\begin{aligned} T(m,n) = \gamma \cdot O\left(\frac{mn^2}{P} + nb^{*2}\log P\right) \\ + \beta \cdot O\left(\frac{mn}{P} + n^2 + nb^*\log P\right) + \alpha \cdot O\left(\frac{n}{b^*}\log P\right) \end{aligned}$$

In the inductive case ($n > b$), the algorithmic cost is due to the two recursive calls, the six 3DMMs, performed on 3D processor grids, and the twelve ALL-TO-ALLs, performed before and after each 3DMM.

To apply Lemma 4 to the 3DMMs, let us now suppose that $P \geq (3c)^{3/4}$ for some $c > 1$ and $b \geq 2P^{1/3}$: since $m \geq n$ at every recursive call and since $n \geq b+1$ in the inductive case, $c \cdot IJK/\min(I,J,K)^3 \leq P \leq IJK$ for each 3DMM. Thus the

3DMMs' overall runtime is

$$\gamma \cdot O\left(\frac{mn^2}{P}\right) + \beta \cdot O\left(\left(\frac{mn^2}{P}\right)^{2/3}\right) + \alpha \cdot O\left(\log P\right).$$

Moreover, the inequalities derived at the beginning of Lemma 4's proof also yield the following upper bound on the overall runtime of the ALL-TO-ALLs,

$$\beta \cdot O\left(\left(\frac{mn}{P} + n + P^2\right)\log P\right) + \alpha \cdot O\left(\log P\right).$$

Overall, we have found that

$$\begin{aligned} T(m,n) = T(m,\lfloor n/2\rfloor) + T(m-\lfloor n/2\rfloor, \lceil n/2\rceil) \\ + \gamma \cdot O\left(\frac{mn^2}{P}\right) + \beta \cdot O\Bigg(\left(\frac{mn^2}{P}\right)^{2/3} \\ + \left(\frac{mn}{P} + n + P^2\right)\log P\Bigg) + \alpha \cdot O\left(\log P\right). \end{aligned}$$

By induction we may take $T(m,n)$ to be nondecreasing in both m and n. The former property justifies replacing $m-\lfloor n/2\rfloor$ by m in the second recursive call. Hence, we will take m to be its initial value in the analysis of every recursive call.

Supposing $n = b2^L$ for a nonnegative integer L, $T(m,n)$ is bounded by Equation (13).

Now observe that $n = b2^{L+1}$ reproduces the asymptotic bound of Equation (13), which thus holds when $b2^L < n < b2^{L+1}$ since $T(m,n)$ is nondecreasing in n.

In conclusion, 3D-CAQR-EG's runtime $T(m,n)$ satisfies Equation (13) if $P \geq 3$, $b \geq 2P^{1/3}$, $P = O(b^2)$, and $P^* = O(b^{*2})$. □

PROOF OF THEOREM 1. Consider picking b, b^* as in Equation (12). The constraints relating P and b are satisfiable if

$$P = O\left(m^{\frac{2\delta}{1+2\delta}} \cdot n^{\frac{2-2\delta}{1+2\delta}}\right), \tag{14}$$

and the constraint relating P^* and b^* is satisfiable if

$$P \cdot (\log P)^{\frac{2\epsilon}{1+2\delta}} = O\left(m^{\frac{2\delta}{1+2\delta}} \cdot n^{\frac{2-2\delta}{1+2\delta}}\right), \tag{15}$$

a stronger condition than Equation (14).

Substituting Equation (10) into Equation (11),

$$\begin{aligned} \gamma \cdot \left(\frac{mn^2}{P}\left(1 + \left(\frac{nP}{m}\right)^{1-2\delta}(\log P)^{1-2\epsilon}\right)\right) \\ + \beta \cdot O\left(\frac{n^2}{\left(\frac{nP}{m}\right)^{\delta}}\left(1 + (\log P)^{1-\epsilon}\right) + W\right) \\ + \alpha \cdot O\left((nP/m)^{\delta}(\log P)^{1+\epsilon}\right), \end{aligned}$$

where W denotes the sum of three terms associated with the ALL-TO-ALLs,

$$\frac{mn}{P}\log\frac{nP}{m}\log P,\ n\log\frac{nP}{m}\log P,\ P^2\left(\frac{nP}{m}\right)^{\delta}\log P,$$

plus a term $n^2/(nP/m)^{2/3}$ associated with the 3DMMs. We obtain the stated arithmetic and latency costs by taking $\delta \geq 1/2$ and $\epsilon = 1$. To suppress the bandwidth term W, it suffices to require that there exists $\delta' \in (0, 1-\delta)$, hence $\delta < 1$, such that

$$\begin{aligned} P \,/\, (\log P)^{\frac{\epsilon}{1-\delta-\delta'}} &= \Omega\big(m \,/\, n\big), \\ P \cdot (\log P)^{\frac{\epsilon}{\delta+\delta'}} &= O\left(m \cdot n^{\frac{1-\delta-\delta'}{\delta+\delta'}}\right), \\ P \cdot (\log P)^{\frac{\epsilon}{2+2\delta}} &= O\left(m^{\frac{\delta}{1+\delta}} \cdot n^{\frac{1-\delta}{1+\delta}}\right). \end{aligned} \tag{16}$$

The 3DMMs' bandwidth cost cannot be reduced, but it is lower-order if $\delta \leq 2/3$.

The hypotheses of Theorem 1, $\delta \in [1/2, 2/3]$ (and tacitly $\epsilon = 1$) and Equation (2), imply Equations (14) to (16). □

This argument extends to a larger range of nonnegative δ, ϵ. Assuming fixed ϵ, for $\delta > 2/3$ the 3DMM invocations dominate the bandwidth cost, whose bound remains as if $\delta = 2/3$, no longer a tradeoff, at least asymptotically. In the case $0 \leq \delta < 1/2$, the additive term in the arithmetic cost, due to the small (mostly) triangular matrix operations on TSQR's critical path, possibly dominates. (A sensible interpretation of the case $\delta \leq 0$ is $b = n$, in which case 1D-CAQR-EG is invoked immediately.) Assuming fixed δ, the tradeoffs due to varying $\epsilon \in [0,1]$ are just as in the proof of Theorem 2, except now the factor in the arithmetic cost is suppressed by increasing δ.

8. DISCUSSION

We have presented two new algorithms, 1D-CAQR-EG and 3D-CAQR-EG (Sections 6 and 7), for computing QR decompositions on distributed-memory parallel machines. Our analysis (e.g., Equations (11) and (13)) demonstrates tradeoffs between arithmetic, bandwidth, and latency costs, governed by the choice of one (1D-CAQR-EG) or two (3D-CAQR-EG) block sizes. We navigated these tradeoffs in Theorems 1 and 2 by asymptotically minimizing arithmetic, as well as bandwidth in Theorem 2.

8.1 Comparison With Similar Algorithms

Here we compare the two new algorithms with four other instances of the REC-QR framework, deriving Tables 2 and 3. Let us review the other algorithms.

An early and well-known instance of REC-QR (Algorithm 1) was proposed by Householder [Hou58]. It features a splitting strategy (Line 4) where $\mathbf{A}_{11}$ is $b \times b$ (*right-looking*) or $n-b \times n-b$ (*left-looking*); a base case threshold (Line 1) asserting $n \leq b$; and a base case subroutine (Line 2) generating a product of b Householder reflectors. Householder's proposal was right-looking and *unblocked*, meaning $b = 1$ (vs. *blocked*, $b > 1$).

Let 1D-HOUSE and 2D-HOUSE denote the un/blocked right-looking variants, specialized to use compact representations (Section 2.3); their costs are summarized in the first rows of Tables 2 and 3. 2D-HOUSE invokes 1D-HOUSE as its base case. For 1D-HOUSE we use a 1D processor grid and for 2D-HOUSE we use a 2D processor grid. For 1D-HOUSE we distribute matrices similar to 1D-CAQR-EG and for 2D-HOUSE we distribute matrices (2D-) block-cyclically with $b \times b$ blocks: the distribution block size matches the algorithmic block size. We parallelize 1D-HOUSE and the base case of 2D-HOUSE to match the distribution of $\mathbf{A}$, analogous to 1D-CAQR-EG. We parallelize 2D-HOUSE's inductive-case matrix multiplications to match the output matrix distribution (a 2D parallelization). In the case of 1D-HOUSE we assume

algorithm	*# operations*	*# words*	*# messages*
2D-HOUSE	mn^2/P	$n^2/(nP/m)^{1/2}$	$n \log P$
CAQR	mn^2/P	$n^2/(nP/m)^{1/2}$	$(nP/m)^{1/2}(\log P)^2$
3D-CAQR-EG	mn^2/P	$n^2/(nP/m)^{\delta}$	$(nP/m)^{\delta}(\log P)^2$

Table 2: Comparison of approaches for square-ish matrices ($m/n = O(P)$). The algorithms and the assumptions that support these bounds are explained in Section 8.1. (In line 3, δ varies from 1/2 to 2/3.)

algorithm	*# operations*	*# words*	*# messages*
1D-HOUSE	mn^2/P	$n^2 \log P$	$n \log P$
TSQR	$mn^2/P + n^3 \log P$	$n^2 \log P$	$\log P$
1D-CAQR-EG	$mn^2/P + n^3(\log P)^{1-2\epsilon}$	$n^2(\log P)^{1-\epsilon}$	$(\log P)^{1+\epsilon}$

Table 3: Comparison of approaches for tall/skinny matrices ($m/n = \Omega(P)$). The algorithms and the assumptions that support these bounds are explained in Section 8.1. (In line 3, ϵ varies from 0 to 1.)

$P = O(m)$. In the case of 2D-HOUSE, we choose an $r \times c$ processor grid with $c = \Theta((nP/m)^{1/2})$ and $r = \Theta(P/c)$, and we choose $b = \Theta(1)$. Assuming $P = \Omega(m/n)$ and $P \cdot (\log P)^2 = O(m \cdot n)$, these choices are valid and simultaneously minimize all three costs, asymptotically.

CAQR [DGHL12] modifies 2D-HOUSE to invoke TSQR (Section 5) in the base case. Our algorithms make crucial use of the TSQR enhancements in [BDG+15]; additionally, we use that paper's improved CAQR in the following comparison. We parallelize and distribute data for TSQR as discussed in Section 5, and for CAQR's inductive case as we did for 2D-HOUSE's. TSQR and CAQR's costs are summarized in the second rows of Tables 2 and 3. In the case of TSQR we assume $P \leq m/n$. In the case of CAQR we use the same $r \times c$ grid as for 2D-HOUSE but now pick $b = \Theta(n/(nP/m)^{1/2})$. Assuming $P/(\log P)^2 = \Omega(m/n)$ and $P \cdot (\log P)^2 = O(m \cdot n)$, these choices are valid and simultaneously minimize all three costs, asymptotically.

The costs of the new algorithms, 1D-CAQR-EG and 3D-CAQR-EG appear in the third rows of Tables 2 and 3. To make the comparison between TSQR and 1D-CAQR-EG more clear, for the latter we use $b = \Theta(n/(\log P)^{\epsilon})$ in Theorem 2's proof, allowing the parameter ϵ to vary over $[0, 1]$, justified by the stronger constraint $P(\log P)^{2\epsilon} = O\left(n^2\right)$. For 3D-CAQR-EG we follow Theorem 1's proof and hypotheses.

8.2 Elimination By Blocks

Tiskin [Tis07], working in the BSP model [Val90], proposed an algorithm outside of the REC-QR framework which demonstrates a similar bandwidth/latency tradeoff as 3D-CAQR-EG. Tiskin's algorithm was designed only for square matrices, but it has been extended to rectangular matrices, using the original algorithm as a black-box [SBDH17]. This extension achieves BSP bandwidth cost $O\left(n^2/(nP/m)^{\delta}\right)$ and BSP synchronization cost $O\left((nP/m)^{\delta}(\log P)^2\right)$. Despite the fact that, in BSP, 3D-CAQR-EG achieves these same communication costs, we still believe 3D-CAQR-EG is a valuable contribution for multiple reasons. Defining a data distribution is a nontrivial and crucial step in developing a distributed-memory implementation using, e.g., MPI. The aforementioned BSP algorithms do not (and need not) explicitly specify their data distributions. Second, our algorithms are based on Householder's algorithm and use Householder representation. Thus, they are readily assembled from robust, tuned subroutines in standard libraries like (P)BLAS and (Sca)LAPACK. Additionally, all interprocessor communication in our algorithms is expressed in terms of standard MPI collectives. Lastly, we feel that Tiskin's recursive scheme, based on a slope-2 wavefront and 'pseudopanels', is much more demanding from an implementation perspective than Elmroth-Gustavson's (QR-EG). To our knowledge no one has implemented Tiskin's algorithm.

8.3 Lower Bounds

Let us continue the notation from Section 8.1. The algorithms studied there are all subject to an arithmetic lower bound of $\Omega(mn^2/P)$ [DGHL12].

In the tall-skinny case, we have bandwidth and latency bounds $\Omega(n^2)$ and $\Omega(\log P)$ [CHPvdG07, BCD+14]. 1D-HOUSE attains the arithmetic lower bound, but misses the bandwidth and latency lower bounds by $\Theta(\log P)$ and $\Theta(n)$. TSQR attains the arithmetic lower bound assuming $P \log P = \Omega(m/n)$, but misses the bandwidth and latency lower bounds both by $\Theta(\log P)$. 1D-CAQR-EG attains the latency lower bound when $\epsilon = 0$, the arithmetic lower bound when $\epsilon \leq 1/2$, and the bandwidth lower bound when $\epsilon \geq 1$.

In the (close to) square case, we have bandwidth and latency bounds $\Omega(n^2/(nP/m)^{2/3}$ and $\Omega((nP/m)^{1/2})$ [BCD+14]. We restrict parameters so that both CAQR and 3D-CAQR-EG attain the arithmetic lower bound, like 2D-HOUSE. 2D-HOUSE and CAQR exceed the bandwidth lower bound both by a factor of $\Theta((nP/m)^{1/6})$ and they exceed the latency lower bound by factors of $\Theta\left(n/(nP/m)^{1/2} \log P\right)$ and $\Theta\left((nP/m)^{1/6}(\log P)^2\right)$, resp. 3D-CAQR-EG attains the bandwidth lower bound when $\delta = 2/3$, and exceeds the latency lower bound by just $\Theta((\log P)^2)$ when $\delta = 1/3$.

We did not prove that 3D-CAQR-EG's bandwidth-latency product is optimal — i.e., that the tradeoff is inevitable — although we conjecture this to be the case. Our intuition is based on bandwidth/latency tradeoffs observed in computations whose dependence graphs have similar diamond-shaped substructures: see, e.g., [PU87, SCKD16].

8.4 Limitations and Extensions

Our main upper bound Theorem 1 is substantially limited by its restrictions on permissible parallelism: see Equation (2). 3D-CAQR-EG's ALL-TO-ALLs are responsible for these constraints: if we supposed the ALL-TO-ALLs had zero cost, Equation (2) could be weakened to Equation (15). We make three remarks about improving this aspect of our work.

First, the bound used is worst-case; our knowledge of (and control over) data distribution could lead to stronger bounds. Second, it may be that the index algorithm is suboptimal for the data distribution, e.g., many $B_{pq} = 0$ and a specialized algorithm would perform less communication, or at least yield sharper cost bounds. Third and more generally, we should optimize for the data distribution before and after each subroutine. The constraints are the balance assumptions to invoke Lemma 4 and 1D-CAQR-EG. This is a difficult combinatorial problem.

The constants hidden in our asymptotic analysis are practically important, and the precise choices of parameters for particular machines warrants further study. We have also omitted a number of practical optimizations that do not affect our asymptotic analysis. For example, recall from Section 2.3 that $\mathbf{T}$ can be reconstructed from $\mathbf{V}$. If the full $\mathbf{T}$ is not desired, by replacing the top level of recursion with a right-looking iterative QR-EG variant, we can avoid ever computing superdiagonal blocks of $\mathbf{T}$; this does, however, restrict the available parallelism [EG00].

Acknowledgments

G. Ballard was supported by the National Science Foundation (NSF) Grant No. ACI-1642385. The work of L. Grigori was supported by the NLAFET project as part of the European Union's Horizon 2020 research and innovation program under grant 671633. Support for M. Jacquelin was provided in part through the Scientific Discovery through Advanced Computing (SciDAC) program funded by the US Department of Energy (DOE), Office of Science, Advanced Scientific Computing Research under Contract No. DE-AC02-05CH11231.

9. REFERENCES

[ABG+95] Ramesh C Agarwal, Susanne M Balle, Fred G Gustavson, Mahesh Joshi, and P Palkar. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development*, 39(5):575–582, 1995.

[BCC+97] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, USA, May 1997. Also available from http://www.netlib.org/scalapack/.

[BCD+14] Grey Ballard, E Carson, J Demmel, M Hoemmen, Nicholas Knight, and Oded Schwartz. Communication lower bounds and optimal algorithms for numerical linear algebra. *Acta Numerica*, 23:1–155, 2014.

[BDG+15] Grey Ballard, James Demmel, Laura Grigori, Mathias Jacquelin, Nicholas Knight, and Hong Diep Nguyen. Reconstructing Householder vectors from tall-skinny QR. *Journal of Parallel and Distributed Computing*, 85:3–31, August 2015.

[BDKS16] Grey Ballard, Alex Druinsky, Nicholas Knight, and Oded Schwartz. Hypergraph partitioning for sparse matrix-matrix multiplication. *ACM Transactions on Parallel Computing (TOPC)*, 3(3):18, 2016.

[BHK+97] Jehoshua Bruck, Ching-Tien Ho, Shlomo Kipnis, Eli Upfal, and Derrick Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on parallel and distributed systems*, 8(11):1143–1156, 1997.

[CHPvdG07] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007.

[DGHL12] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM Journal on Scientific Computing*, 34(1):A206–A239, 2012.

[EG00] Erik Elmroth and Fred G Gustavson. Applying recursion to serial and parallel qr factorization leads to better performance. *IBM Journal of Research and Development*, 44(4):605–624, 2000.

[HBJ96] David R Helman, David A Bader, and Joseph JáJá. Parallel algorithms for personalized communication and sorting with an experimental study. In *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, pages 211–222. ACM, 1996.

[Hou58] Alston S Householder. Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM (JACM)*, 5(4):339–342, 1958.

[PU87] Christos H Papadimitriou and Jeffrey D Ullman. A communication-time tradeoff. *SIAM Journal on Computing*, 16(4):639–646, 1987.

[Pug92] Chiara Puglisi. Modification of the Householder method based on the compact WY representation. *SIAM Journal on Scientific and Statistical Computing*, 13(3):723–726, 1992.

[SB95] Xiaobai Sun and Christian Bischof. A basis-kernel representation of orthogonal matrices. *SIAM journal on matrix analysis and applications*, 16(4):1184–1196, 1995.

[SBDH17] Edgar Solomonik, Grey Ballard, James Demmel, and Torsten Hoefler. A communication-avoiding parallel algorithm for the symmetric eigenvalue problem. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '17, pages 111–121, New York, NY, USA, 2017. ACM.

[SCKD16] Edgar Solomonik, Erin Carson, Nicholas Knight, and James Demmel. Trade-offs between synchronization, communication, and computation in parallel linear algebra computations. *ACM Transactions on Parallel Computing (TOPC)*, 3(1):3, 2016.

[SVdGP16] Martin D Schatz, Robert A Van de Geijn, and Jack Poulson. Parallel matrix multiplication: A systematic journey. *SIAM Journal on Scientific Computing*, 38(6):C748–C781, 2016.

[SVL89] Robert Schreiber and Charles Van Loan. A storage-efficient WY representation for products of Householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 10(1):53–57, 1989.

[Tis07] A. Tiskin. Communication-efficient parallel generic pairwise elimination. *Future Generation Computer Systems*, 23(2):179–188, 2007.

[TRG05] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[Val90] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

APPENDIX

A. PROOF OF LEMMA 1

In Section A we give a more detailed proof of Lemma 1. First in Section A.1 we review the *binomial-tree algorithms* for SCATTER, GATHER, BROADCAST, REDUCE, and ALL-REDUCE. Then in Section A.2 we review the *bidirectional-exchange algorithms* for REDUCE-SCATTER, ALL-GATHER, BROADCAST, REDUCE, and ALL-REDUCE. Finally in Section A.3 we review the *(radix-2) index algorithm* for ALL-TO-ALL, and a two-phase variant that admits sharper bounds when the block sizes vary widely. For the four collectives with two algorithmic variants, we report the smaller of the two upper bounds in Table 1.

We do not claim the chosen algorithms are optimal in our model or in practice. When developing a high-performance implementation we would invoke the corresponding subroutines in a tuned MPI library.

A.1 Binomial Tree Algorithms

The binomial-tree SCATTER algorithm proceeds as follows. The algorithm terminates immediately when $P = 1$; when $P > 1$, we split the P processors into two sets, of size $\lceil P/2 \rceil$ and $\lfloor P/2 \rfloor$, and pick a processor r' in the set not containing processor r. Processor r sends all blocks it owns destined for the opposite subset to processor r'. Processors r and r' then become the roots of two smaller SCATTERs among their respective processor subsets, which proceed in parallel. An upper bound on the runtime satisfies

$$T(P, B) \leq T(\lceil P/2 \rceil,\ B) + \beta \cdot \lceil P/2 \rceil B + \alpha \cdot 2,$$

which simplifies to

$$\beta \cdot O\left((P-1)B\right) + \alpha \cdot O\left(\log P\right). \tag{17}$$

The binomial tree BROADCAST algorithm is an optimization of the SCATTER algorithm when all blocks are identical: each message contains exactly one block, so the upper bound simplifies to

$$\beta \cdot O\left(B \log P\right) + \alpha \cdot O\left(\log P\right). \tag{18}$$

The binomial tree GATHER algorithm reverses the SCATTER's communication pattern, using head- (vs. tail-) recursion. That is, after the recursive GATHERs with roots r and r' complete, processor r' sends processor r all blocks it owns. An upper bound on the runtime satisfies the same recurrence as before, reproducing Equation (17)

The binomial tree REDUCE algorithm is an optimization of the GATHER algorithm when blocks are added as soon as they are received: each message contains exactly one block, so the upper bound simplifies to

$$\gamma \cdot O\left(B \log P\right) + \beta \cdot O\left(B \log P\right) + \alpha \cdot O\left(\log P\right). \tag{19}$$

The binomial tree ALL-REDUCE algorithm is a REDUCE followed by a BROADCAST, so its runtime also satisfies Equation (19).

A.2 Bidirectional Exchange Algorithms

The bidirectional exchange REDUCE-SCATTER algorithm proceeds as follows. The algorithm terminates immediately when $P = 1$; when $P > 1$, we split the P processors into two sets, of size $\lceil P/2 \rceil$ and $\lfloor P/2 \rfloor$, and pair each processor with a processor in the other set. Each processor sends its paired processor all blocks it owns destined for the other set. If the sets differ in size, one processor p in the smaller set is paired with two processors q, q' in the larger set: processor p only sends to one of the two, but receives from both. After each exchange, each processor adds the blocks it receives to the ones it already owns with the same destinations. Each set then performs a smaller REDUCE-SCATTER. An upper bound on the runtime satisfies

$$\begin{aligned} T(P, B) \leq\ & T(\lceil P/2 \rceil,\ B) \\ & + \gamma \cdot 2\lfloor P/2 \rfloor + \beta \cdot (\lceil P/2 \rceil + 2\lfloor P/2 \rfloor) + \alpha \cdot 3, \end{aligned}$$

which simplifies to

$$\gamma \cdot O\left((P-1)B\right) + \beta \cdot O\left((P-1)B\right) + \alpha \cdot O\left(\log P\right).$$

The bidirectional exchange ALL-GATHER algorithm reverses the REDUCE-SCATTER's communication pattern, using head- (vs. tail-) recursion. That is, after the recursive ALL-GATHERs complete, each processor sends its paired processor all blocks it owns destined for the other set, and if the sets differ in size, processor p only receives from one of q, q', but sends to both. Since blocks with the same sources are identical, each processor only sends only one copy of each. An upper bound on the runtime satisfies the same recurrence as before, except without the arithmetic cost, reproducing Equation (17).

When B is sufficiently large with respect to P, it is beneficial implement BROADCAST, REDUCE, and ALL-REDUCE with the preceding bidirectional exchange algorithms, splitting the original blocks into new blocks of size at most $\lceil B/P \rceil$. The bidirectional exchange BROADCAST algorithm is a SCATTER followed by an ALL-GATHER, with runtime

$$\beta \cdot O\left((P-1)\lceil B/P \rceil\right) + \alpha \cdot O\left(\log P\right). \tag{20}$$

The bidirectional exchange REDUCE algorithm is a REDUCE-SCATTER followed by a GATHER, with runtime

$$\begin{aligned} \gamma \cdot O\left((P-1)\lceil B/P \rceil\right) + \beta \cdot O\left((P-1)\lceil B/P \rceil\right) \\ + \alpha \cdot O\left(\log P\right). \end{aligned} \tag{21}$$

The bidirectional exchange ALL-REDUCE algorithm is a REDUCE-SCATTER followed by an ALL-GATHER, so its runtime also satisfies Equation (21).

A.3 All-to-All

We consider an ALL-TO-ALL algorithm called the *radix-2 index algorithm* [BHK+97]. Processors are numbered from 0 and $P-1$, and each block is labeled $q-p \bmod P$, where p is the source processor number and q is the destination processor number. Block labels are encoded in binary, as bit-strings of length $d = \lceil \log_2 P \rceil$, starting from their least significant bit. For each step $i = 0, 1, \ldots, d-1$, each processor p sends processor $p+2^i \bmod P$ a single message containing all blocks it currently owns whose labels' i-th bits are nonzero, at most $\lceil P/2 \rceil$ blocks. After d steps, all blocks have arrived at their destination. The runtime is thus

$$\beta \cdot O\left(BP \log P\right) + \alpha \cdot O\left(\log P\right).$$

All ALL-TO-ALLs in this work use a two-phase approach [HBJ96] that explicitly addresses variable block-sizes. Each processor p starts by balancing its outgoing blocks, dealing the B_{pq} elements of its original block destined for each processor q into new blocks destined for processors $p+q$, $p+q+1$, ..., and so on, cyclically. The processors then perform two ALL-TO-ALLs, the first to route the new blocks to intermediate processors, and the second to route elements to their original destinations. The largest block-sizes B' and B'' in the first and second ALL-TO-ALLs are bounded,

$$B' \leq \frac{P-1}{2} + \max_q \sum_p \frac{B_{pq}}{P}$$

$$B'' \leq \frac{P-1}{2} + \max_p \sum_q \frac{B_{pq}}{P}.$$

The overall runtime is thus bounded by

$$\beta \cdot O\left(\left(B_* + P^2\right) \log P\right) + \alpha \cdot O\left(\log P\right),$$

where $B_* \leq BP$ is the maximum number of words any processor holds before or after the collective,

$$B_* = \max\left(\max_q \sum_p B_{pq},\ \max_p \sum_q B_{pq}\right).$$

B. PROOF OF LEMMAS 2, 3, AND 4

Here we provide a more comprehensive analysis of matrix multiplication than in Section 4.

B.1 Generic Algorithm

Consider multiplying an $I \times K$ matrix $\mathbf{A}$ with a $K \times J$ matrix $\mathbf{B}$ to obtain an $I \times J$ matrix $\mathbf{C}$, via the usual (entrywise) formula: for all $(i,j) \in [I] \times [J]$,

$$\mathbf{C}_{ij} = \sum_{k \in [K]} \mathbf{A}_{ik} \mathbf{B}_{kj}.$$

We identify each of the IJK (scalar) multiplications with a point (i,j,k) in three-dimensional Euclidean space, so the set of multiplications defines a discrete $I \times J \times K$ brick. We will parallelize the multiplications by arranging the processors in a three-dimensional grid and assigning each processor a sub-brick.

A complication is that the admissible processor grids depend on the integer factorizations of the number P of processors: for flexibility, we will allow some processors to remain idle. That is, writing $P = QRS + T$ for any positive integers Q, R, and S and nonnegative integer T, we arrange QRS processors in a $Q \times R \times S$ grid, indexing each by a triple (q,r,s), and set the remaining T processors aside.

Having fixed a processor grid, let $\{\mathcal{I}_q\}_{q \in [Q]}$, $\{\mathcal{J}_r\}_{r \in [R]}$, and $\{\mathcal{K}_s\}_{s \in [S]}$ denote partitions of $[I]$, $[J]$, and $[K]$. Assign each grid processor (q,r,s) the sub-brick $\mathcal{I}_q \times \mathcal{J}_r \times \mathcal{K}_s$, meaning the (sub-) matrix product

$$\mathbf{Z}_{\mathcal{I}_q, \mathcal{J}_r, s} = \mathbf{A}_{\mathcal{I}_q, \mathcal{K}_s} \cdot \mathbf{B}_{\mathcal{K}_s, \mathcal{J}_r}.$$

Slices of the $I \times J \times S$ tensor $\mathbf{Z}$ are partial sums of the output matrix: in particular, for all $(q,r) \in [Q] \times [R]$,

$$\mathbf{C}_{\mathcal{I}_q, \mathcal{J}_r} = \sum_{s \in [S]} \mathbf{Z}_{\mathcal{I}_q, \mathcal{J}_r, s}.$$

We will parallelize these remaining $IJ(S-1)$ additions as part of data redistribution.

The initial distributions of $\mathbf{A}$ and $\mathbf{B}$, and the final distribution of $\mathbf{C}$, are chosen as follows. For each $(q,s) \in [Q] \times [S]$, $\mathbf{A}_{\mathcal{I}_q, \mathcal{K}_s}$ is partitioned entrywise across the grid processors $(q, \cdot, s)$, and, for each $(r,s) \in [R] \times [S]$, $\mathbf{B}_{\mathcal{K}_s, \mathcal{J}_r}$ is partitioned entrywise across the grid processors $(\cdot, r, s)$. We identify the matrix entries that grid processor (q,r,s) owns with the sets $\mathcal{A}_{q,r,s} \subseteq \mathcal{I}_q \times \mathcal{K}_s$ and $\mathcal{B}_{q,r,s} \subseteq \mathcal{K}_s \times \mathcal{J}_r$.

During the computation, each of the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ undergoes a redistribution, using ALL-GATHERs or REDUCE-SCATTERs along fibers of the processor grid.

First, for each $(q,s) \in [Q] \times [S]$, we perform an ALL-GATHER among the grid processors $(q, \cdot, s)$ so that each obtains a copy of $\mathbf{A}_{\mathcal{I}_q, \mathcal{K}_s}$; these QS ALL-GATHERs, each involving a disjoint set of R processors, occur simultaneously. Next, for each $(r,s) \in [R] \times [S]$, we perform an ALL-GATHER among the grid processors $(\cdot, r, s)$ so that each obtains a copy of $\mathbf{B}_{\mathcal{K}_s, \mathcal{J}_r}$; similarly, these RS ALL-GATHERs, each involving a disjoint set of Q processors, occur simultaneously.

At this point, each grid processor (q,r,s) can evaluate its local matrix product, obtaining the matrix $\mathbf{Z}_{\mathcal{I}_q, \mathcal{J}_r, s}$ as explained above.

Finally, to construct the output matrix, for each $(q,r) \in [Q] \times [R]$, we perform a REDUCE-SCATTER among the grid processors $(q,r,\cdot)$, computing $\mathbf{C}_{\mathcal{I}_q, \mathcal{J}_r}$ while partitioning it entrywise across these processors; these QR REDUCE-SCATTERs, each involving a disjoint set of S processors, can occur simultaneously. We identify the matrix entries that grid processor (q,r,s) owns after these collectives with the set $\mathcal{C}_{q,r,s} \subseteq \mathcal{I}_q \times \mathcal{J}_r$.

B.2 Analysis

We denote by MM the multiplication of an $I \times K$ matrix with a $K \times J$ matrix, both stored locally on one processor: this is the special case of 3DMM on a $1 \times 1 \times 1$ processor grid. Using the conventional algorithm, which involves IJK multiplications and $IJ(K-1)$ additions, MM's runtime is bounded,

$$T_{\text{MM}}\left(I,J,K\right) = \gamma \cdot O\left(IJK\right).$$

(This was Lemma 2.)

The algorithmic costs of 3DMM are due to the MMs, ALL-GATHERs, and REDUCE-SCATTERs. The costs of these subroutines, in turn, depend on the splitting $P = QRS + T$ and the six partitions,

$$\begin{array}{llll} \{\mathcal{I}_q\}_q & \text{of} & [I], & \{\mathcal{A}_{q,r,s}\}_{q,r,s} \quad \text{of} \quad [I] \times [K], \\ \{\mathcal{J}_r\}_r & \text{of} & [J], & \{\mathcal{B}_{q,r,s}\}_{q,r,s} \quad \text{of} \quad [K] \times [J], \\ \{\mathcal{K}_s\}_s & \text{of} & [K], & \{\mathcal{C}_{q,r,s}\}_{q,r,s} \quad \text{of} \quad [I] \times [J]. \end{array}$$

LEMMA 8. 3DMM's *runtime is bounded by*

$$\begin{aligned} & \max_{q,r,s} T_{\text{MM}}\left(|\mathcal{I}_q|, |\mathcal{J}_r|, |\mathcal{K}_s|\right) \\ + & \max_{q,s} T_{\text{ALL-GATHER}}\left(\{\mathcal{A}_{q,r,s}\}_r\right) \\ + & \max_{r,s} T_{\text{ALL-GATHER}}\left(\{\mathcal{B}_{q,r,s}\}_q\right) \\ + & \max_{q,r} T_{\text{REDUCE-SCATTER}}\left(\{\mathcal{C}_{q,r,s}\}_s\right). \end{aligned}$$

Upper bounds on $T_{\text{ALL-GATHER}}$ and $T_{\text{REDUCE-SCATTER}}$ are given in Lemma 1. However, those results are pessimistic when the block-sizes vary. For example, on a one-dimensional processor grid, an ALL-GATHER or REDUCE-SCATTER simplifies to a BROADCAST or REDUCE and sharper bounds apply. This situation arises in 1D-CAQR-EG, in whose analysis we will additionally constrain the number of processors to ensure that the arithmetic and bandwidth costs of the BROADCAST/REDUCE are independent of P. Lemma 3 is a corollary of Lemma 8 that summarizes this special case.

PROOF OF LEMMA 3. Suppose first that $K = \max(I, J, K)$. Pick $Q = R = 1$ and $S = P$, which is a valid $Q \times R \times S$ processor grid since Q, R, S are positive integers with $QRS \leq P$. Fix any balanced partitions $\{\mathcal{I}_q\}_q$, $\{\mathcal{J}_r\}_r$, and $\{\mathcal{K}_s\}_s$. Take the partition $\{\mathcal{A}_{q,r,s}\}_{q,r,s}$ (resp., $\{\mathcal{B}_{q,r,s}\}_{q,r,s}$) to be a union of balanced R-way (resp., Q-way) partitions of the sets $\mathcal{I}_q \times \mathcal{K}_s$ ($(q,s) \in [Q] \times [S]$), resp., $\mathcal{K}_s \times \mathcal{J}_r$ ($(s,r) \in [S] \times [R]$). Finally, take $\mathcal{C}_{q,r,s} = \emptyset$ for all but one (q,r,s). In this scenario, the ALL-GATHERs are trivial (involving one processor each), and the single REDUCE-SCATTER is just a REDUCE. The arithmetic cost (MMs and REDUCE) is

$$\gamma \cdot O\left(IJ\left\lceil\frac{K}{P}\right\rceil + \min(IJ\log P, IJ + P)\right) = \gamma \cdot O\left(\frac{IJK}{P}\right).$$

The bandwidth cost (REDUCE) is

$$\beta \cdot O\left(\min(IJ\log P, IJ + P)\right) = \beta \cdot O\left(IJ\right).$$

The latency cost (REDUCE) is $\alpha \cdot O(\log P)$.

Now suppose that I (resp., J) $= \max(I, J, K)$ Pick $R = S = 1$ and $Q = P$ (resp., $Q = S = 1$ and $R = P$). Again take any balanced partitions $\{\mathcal{I}_q\}_q$, $\{\mathcal{J}_r\}_r$, and $\{\mathcal{K}_s\}_s$. Similarly to before, take the partitions $\{\mathcal{A}_{q,r,s}\}_{q,r,s}$ and $\{\mathcal{C}_{q,r,s}\}_{q,r,s}$ (resp., $\mathcal{B}_{q,r,s}$ and $\mathcal{C}_{q,r,s}$) to be unions of balanced R- (resp., Q-) and S-way partitions, while taking take $\mathcal{B}_{q,r,s}$ (resp., $\mathcal{A}_{q,r,s}$) $= \emptyset$ for all but one (q,r,s). In this scenario, the ALL-GATHERs of the left (resp., right) factor and the REDUCE-SCATTERs are trivial (involving one processor each), and the single ALL-GATHER of the right (resp., left) factor is just a BROADCAST. The arithmetic cost is $\gamma \cdot O\left(\lceil I/P \rceil JK\right)$, resp., $\gamma \cdot O\left(I \lceil J/P \rceil K\right)$, which $= \gamma \cdot O\left(IJK/P\right)$. Similar to the first case, the bandwidth cost is $\beta \cdot O\left(JK\right)$, resp., $\beta \cdot O\left(IK\right)$, and the latency costs are both $\alpha \cdot O\left(\log P\right)$. □

Lemma 4 is another corollary of Lemma 8, applicable for matrix and processor grid dimensions sufficiently large to admit parallelizations where each processor is assigned a roughly cubical sub-brick of the $I \times J \times K$ computation brick. This situation arises in 3D-CAQR-EG.

PROOF OF LEMMA 4. Pick $Q = \lfloor I/\rho \rfloor$, $R = \lfloor J/\rho \rfloor$, and $S = \lfloor K/\rho \rfloor$, where $\rho = (IJK/P)^{1/3}$. Then

$$(1 - 1/c)^3 P \leq QRS \leq P$$

$$1 \leq \rho \leq \frac{I}{Q}, \frac{J}{R}, \frac{K}{S} \leq \frac{c}{c-1}\rho.$$

This defines a valid $Q \times R \times S$ processor grid, since Q, R, S are positive integers with $QRS \leq P$. Take any balanced partitions $\{\mathcal{I}_q\}_q$, $\{\mathcal{J}_r\}_r$, and $\{\mathcal{K}_s\}_s$. Take the partition $\{\mathcal{A}_{q,r,s}\}_{q,r,s}$ to be a union of balanced R-way partitions of the sets $\mathcal{I}_q \times \mathcal{K}_s$ ($(q,s) \in [Q] \times [S]$), and similarly for the partitions $\{\mathcal{B}_{q,r,s}\}_{q,r,s}$ and $\{\mathcal{C}_{q,r,s}\}_{q,r,s}$. The arithmetic cost (from MMs and REDUCE-SCATTERs) is

$$\begin{aligned} \gamma \cdot O\left(\left\lceil\frac{I}{Q}\right\rceil\left\lceil\frac{J}{R}\right\rceil\left\lceil\frac{K}{S}\right\rceil + (S-1)\left\lceil\frac{\left\lceil\frac{I}{Q}\right\rceil\left\lceil\frac{J}{R}\right\rceil}{S}\right\rceil\right) \\ = \gamma \cdot O\left(\frac{IJK}{P}\right), \end{aligned}$$

where a left-to-right reading of this equality introduces the assumption on P from the theorem statement and fixes Q, R, S as in the preceding paragraph. The bandwidth cost (from ALL-GATHERs/REDUCE-SCATTERs) is

$$\begin{aligned} \beta \cdot O\Bigg((R-1)\left\lceil\frac{\left\lceil\frac{I}{Q}\right\rceil\left\lceil\frac{K}{S}\right\rceil}{R}\right\rceil + (Q-1)\left\lceil\frac{\left\lceil\frac{J}{R}\right\rceil\left\lceil\frac{K}{S}\right\rceil}{Q}\right\rceil \\ + (S-1)\left\lceil\frac{\left\lceil\frac{I}{Q}\right\rceil\left\lceil\frac{J}{R}\right\rceil}{S}\right\rceil\Bigg) = \beta \cdot O\left(\left(\frac{IJK}{P}\right)^{2/3}\right), \end{aligned}$$

where a left-to-right reading is as before. The latency cost (from ALL-GATHERs/REDUCE-SCATTERs) is

$$\alpha \cdot O\left(\log R + \log Q + \log S\right) = \alpha \cdot O\left(\log P\right).$$

□

C. PROOF OF LEMMA 5

Here we provide relatively self-contained description of TSQR. To establish our asymptotic claims it suffices to use a simplified version of TSQR, which we present in Sections C.1 and C.2 and analyze in Section C.3. Proofs of correctness and numerical stability can be found in [BDG+15].

Recall that on input, the $m \times n$ matrix $\mathbf{A}$ is partitioned across the P processors so that each processor p owns $m_p \geq n$ rows, not necessarily contiguous. Thus we require $\mathbf{A}$ be sufficiently tall and skinny: $m/n \geq P$. A single processor r, which owns $\mathbf{A}$'s n leading rows, is designated as the *root processor*.

On output, the Q-factor is stored in Householder representation ($\mathbf{V}$, $\mathbf{T}$), where $\mathbf{V}$ has the same distribution as $\mathbf{A}$. Both $\mathbf{T}$ and the R-factor are returned only on the root processor.

Looking just at its communication pattern, TSQR resembles a REDUCE followed by a BROADCAST, the distinction being the local arithmetic performed before and after each exchange.

C.1 Upsweep

At the start of TSQR each processor p performs a QR decomposition of $\mathbf{A}_p$, its $m_p \times n$ submatrix of $\mathbf{A}$.

$$\left(\mathbf{V}_p^{(0)}, \mathbf{T}_p^{(0)}, \mathbf{R}_p^{(0)}\right) = \text{LOCAL-QR}\left(\mathbf{A}_p\right).$$

The subroutine LOCAL-QR is unspecified other than that it computes a QR decomposition of an $\mu \times \nu$ matrix ($\mu \geq \nu$), stored locally, in $O(\mu\nu^2)$ operations, and returns the Q-

and R-factors in the compact representations described in Section 2.3.

Next, the processors perform a REDUCE using the binomial tree algorithm (Section A.1) with root r and blocks $\{\mathbf{R}_p\}_p$, meaning the block-size is $n(n+1)/2$. However, instead of adding the blocks elementwise after each exchange, we perform local QR decompositions:

$$\left(\mathbf{V}_p^{(\ell)}, \mathbf{T}_p^{(\ell)}, \mathbf{R}_p^{(\ell)}\right) = \text{LOCAL-QR}\left(\begin{bmatrix}\mathbf{R}_p^{(\ell-1)}\\ \mathbf{R}_q^{(\ell-1)}\end{bmatrix}\right),$$

where $\mathbf{R}_p^{(\ell-1)}$ is processor p's R-factor from its previous QR decomposition and $\mathbf{R}_q^{(\ell-1)}$ is the R-factor it just received from some other processor $q \neq p$. Each processor p keeps the Q-factors (in basis-kernel representation) it produces — they will be used subsequently — but the R-factors are destroyed once they are sent on to another processor. At the end of the REDUCE, each processor stores between 1 and $L = \lceil \log_2 P \rceil$ intermediate Q-factors, and processor r also stores $\mathbf{R}_r^{(L)}$, an R-factor of a QR decomposition of $\mathbf{A}$.

C.2 Downsweep

In principle, we can recover the Q-factor (in Householder representation) almost directly from $\mathbf{A}$ and the R-factor: see [BDG+15, Section 4.4] for a survey of approaches. However, numerical issues motivate taking an additional step, recovering the leading n columns of the Q-factor, which can be done stably and efficiently by applying the tree of Q-factors to a set of n identity columns [DGHL12]. This resembles a BROADCAST using the binomial tree algorithm (Section A.1) with root r and block-size n^2, reversing the communication pattern of the REDUCE. Unlike a typical BROADCAST, however, the block's contents change each time it is sent to another processor. That is, whenever processor p received a block from processor q during the REDUCE, processor p now computes,

$$\begin{bmatrix}\mathbf{B}_p^{(\ell-1)}\\ \mathbf{B}_q^{(\ell-1)}\end{bmatrix} = \left(\mathbf{I} - \mathbf{V}_p^{(\ell)}\mathbf{T}_p^{(\ell)}(\mathbf{V}_p^{(\ell)})^H\right)\begin{bmatrix}\mathbf{B}_p^{(\ell)}\\ \mathbf{0}\end{bmatrix}$$

and then sends $\mathbf{B}_q^{(\ell)}$ to processor q. To start, the root processor r sets $\mathbf{B}_r^{(L)} = \mathbf{I}$.

Next, each processor p computes

$$\mathbf{W}_p = \left(\mathbf{I} - \mathbf{V}_p^{(0)}\mathbf{T}_p^{(0)}(\mathbf{V}_p^{(0)})^H\right)\begin{bmatrix}\mathbf{B}_p^{(0)}\\ \mathbf{0}\end{bmatrix}.$$

The matrix $\mathbf{W}$, defined by the submatrices $\{\mathbf{W}_p\}_p$ just as $\mathbf{A}$ is defined by $\{\mathbf{A}_p\}_p$, is the rightmost $m \times n$ submatrix of the Q-factor associated with the R-factor obtained by the REDUCE.

It remains to recover a Householder representation of the Q-factor from $\mathbf{W}$. For numerical stability, we exploit the non-uniqueness of a QR decomposition $(\mathbf{Q}, \mathbf{R})$ of $\mathbf{A}$: $(\mathbf{QZ}, \mathbf{Z}^H\mathbf{R})$ is also a QR decomposition of $\mathbf{A}$ for any unitary matrix $\mathbf{Z}$ with 2×2 block-diagonal structure whose leading block is an $n \times n$ diagonal matrix. Processor r row-reduces $\mathbf{X}$, the upper $n \times n$ submatrix of $\mathbf{W}_r$, in the usual approach — working left-to-right, eliminating each column below the diagonal by premultiplying with a unit lower triangular matrix — but with a modification to simultaneously populate an $n \times n$ diagonal matrix $\mathbf{S}$. For each $j \in [n]$, before the j-th column is eliminated, letting $\hat{\mathbf{X}}$ denote the current partially reduced matrix, compute $\mathbf{S}_{jj} = \text{sgn}(\hat{\mathbf{X}}_{jj})$, add $\mathbf{S}_{jj}$ to $\hat{\mathbf{X}}_{jj}$, and then proceed as usual. (Here $\text{sgn}(z)$ means $z/|z|$ for $z \neq 0$ and an arbitrary unit complex number when $z = 0$.) Since $\hat{\mathbf{X}}_{jj} + \mathbf{S}_{jj} \neq 0$, pivoting is unnecessary to avoid breakdown, thus this procedure terminates with both $\mathbf{S}$ and a matrix pair $(\mathbf{L}, \mathbf{U})$, where $\mathbf{L}$ is unit lower triangular, U is upper triangular, and $\mathbf{LU}$ is invertible. It is shown in [BDG+15, Lemma 6.2] that $\mathbf{X} + \mathbf{S} = \mathbf{LU}$ and, moreover, that partial pivoting is obviated (or performed implicitly): at each step, the diagonal element's magnitude is at least that of each element in the column below. Processor r then computes $\mathbf{T} = \mathbf{US}^H\mathbf{L}^{-H}$, $\mathbf{V}_r = \mathbf{L}$, and $\mathbf{R} = -\mathbf{S}^H\mathbf{R}_r^{(L)}$. The processors then perform a BROADCAST of $\mathbf{U}$, with root r, after which each recipient processor $p \neq r$ computes $\mathbf{V}_p = \mathbf{W}_p\mathbf{U}^{-1}$.

We are done: the Q-factor's basis $\mathbf{V}$ is partitioned as $\{\mathbf{V}_p\}_p$ across the processors commensurately with $\mathbf{A}$, and the root processor stores both the kernel $\mathbf{T}$ and the R-factor $\mathbf{R}$.

C.3 Concluding the Analysis

PROOF OF LEMMA 5. The runtime of the LOCAL-QRs, performed in parallel, is $\gamma \cdot O\left(\max_p m_p n^2\right)$, and the runtime of the subsequent 'REDUCE' is

$$\gamma \cdot O\left(n^3 \log P\right) + \beta \cdot O\left(n^2 \log P\right) + \alpha \cdot O\left(\log P\right).$$

Computing $\mathbf{W}$ asymptotically matches the current total runtime. Computing $\mathbf{U}$ on the root processor adds $\gamma \cdot O\left(n^3\right)$, then the subsequent BROADCAST and computing $\mathbf{V}$ matches computing $\mathbf{W}$. □

Note that in Section C.2 it is possible to avoid the second BROADCAST and computing most of $\mathbf{W}$ by starting the first BROADCAST with $\mathbf{B}_r^{(L)} = \mathbf{U}^{-1}$ [BDG+15].